Experience-Based Behavior Adaptation of Kinematically-Complex Robots

von Alexander Dettmann

Dissertation

zur Erlangung des Grades eines Doktors der
Ingenieurwissenschaften
- Dr.-Ing. -

Vorgelegt im Fachbereich 3 (Mathematik & Informatik)
der Universität Bremen
im Dezember 2020

Gutachter: Prof. Dr. Dr. h.c. Frank Kirchner (Universität Bremen)
Prof. Dr. Tim Tiedemann (HAW Hamburg)

Datum des Kolloquiums: 25.02.2021

Zusammenfassung

In den letzen Jahrzehnten entwickelten sich Roboter von blind agierenden Maschinen zu intelligenten Agenten, die mit einer Vielzahl von Aktuatoren und Sensoren sowie mit leistungsfähigen Prozessoren ausgestattet sind. Diese kinematisch komplexen Roboter besitzen die nötige Flexibilität und Mobilität, um sowohl präzise Aktionen wiederholt ausführen als auch mit Hilfe von komplexen Kontrollansätzen auf geänderte Umgebungsbedingungen reagieren zu können. Dabei kommen oft reaktive und/oder planende Bewegungssteuerungen zum Einsatz, die je nach Parametrierung Lösungen für einen limitierten Kontextbereich generieren, d.h. für begrenzte Kommandos, Umgebungseigenschaften oder Zustände des Roboters. Zudem können verschiedene Konfigurationen der Bewegungssteuerung die gleiche Aktion realisieren, aber mit unterschiedlichen Leistungsmerkmalen wie Präzision, Stabilität oder Effizienz. Daher muss ein möglichst generischen Roboter, der Aktionen in unterschiedlichen Situationen mit einer anwendungsspezifisch optimale Leistung ausführen soll, seine Bewegungssteuerung kontextabhängig zur Laufzeit anpassen.

Das Ziel dieser Arbeit ist, den Kontextbereich in dem ein Roboter agieren kann zu erhöhen, indem dieser selbstständig seine Bewegungssteuerung und somit sein Verhalten auf Basis von aktuellen Kontextinformationen und gesammelten Erfahrungen anpasst. In dieser Arbeit wird ein Konzept präsentiert, wie Erfahrungen - d.h. Werte für anwendungsspezifische Evaluierungskriterien eines ausgeführten Verhaltens in einem detektierten Kontext - erzeugt, verwaltet und für die Verhaltensadaptation genutzt werden. Der entwickelte Ansatz ermöglicht die kontinuierliche Einbindung von neuen Erfahrungen, wodurch die Grundlage für die autonome Verhaltensanpassung mit der Lebenszeit des Roboters wächst und somit mögliche Zustandsänderungen wie Verschleiß berücksichtigt werden. Für die Ableitung des vermeintlich besten Verhaltens für den aktuellen Kontext auf Basis von Erfahrungen werden zwei unterschiedliche Methoden implementiert und miteinander verglichen. Der erste Ansatz basiert auf fallbasiertem Schließen während der zweite einen modellbasierter Ansatz implementiert, welcher eine inkrementelle Gaußprozessregression als internes Simulationsmodell für die Suche nach einem optimalen Verhalten zu Grunde legt.

Die erfahrungsbasierte Verhaltensanpassung wird zur echtzeitfähigen Adaptation von Laufverhalten am Beispiel von zwei kinematisch komplexen Robotern angewandt, d.h. am Hexapoden SpaceClimber und am vierbeinigen Laufroboter Charlie. Zur Erzeugung von Laufverhalten wird eine generische Laufsteuerung präsentiert, welche je nach Konfiguration unterschiedliche Posen und Laufmuster für verschiedenartige Laufroboter erzeugt. Ebenso werden Bewertungskriterien und Metriken zur Bestimmung des Kontextes definiert und angewandt. Durch manuelles und automatisiertes Testen von vielen Verhaltensparameterkonfigurationen werden Wissensdatenbanken generiert. Die experimentelle Auswertung zeigt, dass die gesammelten Erfahrungen in Kombination mit dem vorgestellten Ansatz eine erfolgreiche Verhaltensanpassung für die Zielsysteme erlauben. Beide untersuchten Anpassungsstrategien zeigen Vor- und Nachteile, die je nach Anwendung ihren jeweiligen Einsatz rechtfertigen. Der fallbasierte Ansatz hat seine Stärken, wenn mit Hilfe von wenig Erfahrungen viele Verhaltensaparametern angepasst werden müssen, während der modellbasierte Ansatz gerade für feingranulare Anpassungen bei großen Wissensdatenbanken zu empfehlen ist.

Abstract

In the last decades, robots have increasingly evolved from blindly acting machines towards more and more intelligent agents that feature many actuators and sensors in combination with advanced processing units. Besides precisely repeating actions, these kinematically complex robots provide the flexibility and mobility to react to changing conditions with the help of sophisticated control approaches. Thereby, reactive and/or planning-based motion controls are usually used that need to be configured to generate solutions for a limited context range, i.e. for specific commands, environmental properties, or conditions of the robot. In addition, different configurations of the motion control may fulfill the same action but with different scopes of performance, e.g. precision, stability, or efficiency. Consequently, to obtain a generic robot that can perform actions in various situations while maintaining an application-specific optimal performance, the robot must be able to adapt its motion control during runtime according to the context.

The goal of this thesis is to increase the contextual range in which a robot can act by autonomously adapting its motion control, and thus its behavior, on the base of context information and gained experiences. In this thesis, a concept is presented how experiences - i.e. values for application-specific evaluation criteria of an executed behavior in a detected context - can be generated, managed, and used for behavior adaptation. The presented approach allows the continuous integration of new experiences, whereby the foundation for autonomous behavior adaptation grows with the lifetime of the robot and thus considers possible condition changes such as wearout. To derive the supposedly best behavior for the current context based on experience, two different methods are implemented and compared to exploit their pros and cons. The first approach is based on case-based reasoning, while the second implements a model-based approach that uses an incremental Gaussian process regression as an internal simulation model to search for an optimal behavior.

The experience-based behavior adaptation is analyzed for the use case of adapting walking behaviors during runtime using two kinematically complex robots as examples, the hexapod SpaceClimber and the four-legged walking robot Charlie. Therefore, a generic locomotion control is developed that generates walking behaviors for various types of robots, i.e. it generates different poses and walking patterns with their advantages and disadvantages depending on its configuration. In addition, evaluation criteria and metrics to determine the context are defined and applied. Knowledge bases are generated through manual and automated testing of different behavior parameter configurations. The experimental evaluation shows that the gathered experiences in combination with the presented approach allow a successful behavior adaptation for the target systems. Both investigated adaptation strategies show advantages and disadvantages that justify their respective utilization depending on the application. The case-based reasoning approach has its strengths, if behavior parameters need to be adapted on the foundation of few experiences. The model-based approach is best suited for large knowledge bases to derive fine-grained adaptations.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Dr. Frank Kirchner for guiding me through my thesis over the past years. In addition to the opportunity to work as a researcher in multiple scientific projects, he has given many important advices and a path to follow to finally complete my thesis. Especially the possibility to participate in the annual PhD retreats has had a very positive influence on the content and outcome of my work. The scientific and philosophical discussions together with the other PhD students in casual and inspiring atmosphere widened my horizon and helped me to look at my topic and research in general from different perspectives. Many thanks also go to my second supervisor Prof. Dr. Tim Tiedemann for reviewing my thesis.

Many of my fellow colleagues from the German Research Center for Artificial Intelligence, to be precise from the Robotics Research Center (RIC), and the University of Bremen have contributed useful suggestions and perspectives throughout the years. Special thanks goes to Dr. Sebastian Bartsch, who has accompanied me as mentor for many years, starting with the supervision of my master thesis in 2008, then becoming my project leader, and finally my team leader. He always supported my work closely, gave scientific and administrative advice, and all in all provided an efficient but gentle atmosphere that, on the one side, enabled me to concentrate on specific details while, on the other side, promoted team work to achieve amazing group results.

A warm thanks also goes to my friends and colleagues Dr. Malte Langosz, Dr. Daniel Kühn, and Dr. Florian Cordes. I had constructive discussions about my work (and other things) with all of them throughout my career in projects, work groups, at conferences and fairs, simply during every day office live, and especially during the final writing phase. Simple phrases like "diamonds are formed under pressure" relativized some stressful situations in an amusing manner that have always been keeping up the motivation. They are perfect examples for the friendly and constructive atmosphere within the RIC. I actually could have had mentioned many more. Also many students helped to finish my thesis, both directly involved or even by taking some of the project work at RIC, so that I could concentrate on PhD-related issues. Nevertheless, I especially would like to thank my former students Anna Born, Shuo Jiang, Lukas Ruhe, and Jannik Klemm for their excellent support in mastering many technical time-consuming details.

Most of the work carried out at DFKI is depending on the funding agencies' constant effort to push the frontiers of scientific research. In my case, all work was carried out in projects funded by the German Aerospace Center. In the projects SpaceClimber and LIMES, the development of the hexapod SpaceClimber and the multifunctional robot Mantis as well as the required software developments to control and learn behaviors for walking robots was funded. In addition, the projects iStruct and VIPE made the development of the quadruped Charlie possible, thus proving a different target platform to analyze the research goals of my thesis. Finally, the project TransFit provided sufficient resources to finalize my thesis.

And last but not least, everyone knows immediate family has to endure the load of anybody finishing his or her PhD. In my case, special thanks goes to my beloved wife Sandra, who went together with me through all ups and downs of my life, my career, and of course throughout my dissertation. Even though she heard me saying too often "I only need one or two more years and then I'm done", Sandra always encouraged me in my decisions and at the same time bore the one or the other extra time I needed over the evenings or weekends. My children Jenny and Mila also contribute greatly to my work. Seeing them growing up, making their first experiences, trying new things, learning from failures, and successfully acquiring new skills also inspired my work and motivated some decisions I took. The final thanks goes to my parents Doris and Gunther as well as my sister Xenia who supported me and my decisions during my whole life. I dedicate this work to you, my beloved family.

Contents

Chapter 1

Introduction

This chapter introduces the thesis starting with the motivation of this work. Then, in order to avoid misinterpretations, some terms and definitions are provided before coming to the overall goal of this thesis. In order to reach the goal of the thesis, several contributions to the field of robotics are made, which are summarized in Section 1.4. Finally, an overview of the thesis structure is provided.

1.1 Motivation

Since the third industrial revolution started in the second half of 20^{th} century, small scale electronics, computers, and robots helped to rise an era of high-level automation. Through the possibility to control strong, fast, and precise robots, heavy-duty tasks could be accomplished and repeated numerous times maintaining the same quality. Looking at the development in robotics over the last decades, one can find more and more complex robots with increasing sensor and motor disposition, smaller but faster computational devices, as well as more stable and light-weight mechanical designs. The capability, to sense and react in realtime helped the robots to escape the strictly controlled industrial manufacturing environment. These days, advances in control and artificial intelligence (AI) are now augmenting purely automated systems to autonomous embodied agents that act on their own reducing the need of pre-programming tasks and supervising every single action.

Thus, more and more complex tasks can be solved by robots not only indoors but also outdoors in almost every environment which led to manifold application possibilities. Robot vacuums and mower show already today how successful robotic solutions can be integrated in every day live. Especially in hard-to-reach regions, robots play a growing role to lower risks and costs, e.g. for inspecting under water constructions like oil platforms or off-shore wind power plants [Christensen et al., 2015]. Also in space, numerous successful robotic missions have been executed, e.g in-orbit servic-

ing with the Canadarm [Hiltz et al., 2001] or MEV-1 spacecraft[1] and surface exploration with the Mars Science Laboratory [Grotzinger et al., 2012].

In the past, the robot's hardware and software were usually designed to fulfill only their application-specific task in order to reduce complexity for being efficient and reliable. Consequently, their specialized and minimal selection of sensors and motors as well as hard-coded software limit them from doing a variety of tasks. Nowadays, the development and reuse of core functionalities helps to manage controlling kinematically complex systems. A steady increase of integrated sensors and controlled degrees of freedom (DOF) improves a single robot's flexibility and extends its potential fields of application. Thus, many different tasks can be achieved with one single robot instead of several single-purpose robots, which can safe effort and costs especially in hard-to-reach areas. For instance, instead of sending several application-specific robots to the Moon, one or two mobile multi-purpose robots could effectively investigate the landing side, explore areas of scientific interest, gather soil samples, deploy scientific instruments, or build up infrastructure. In addition, it is more likely, that a kinematically complex robot provides the flexibility to realize many upcoming tasks that may arise during mission execution.

Having a versatile robotic hardware that is theoretically capable of realizing various tasks in diverse applications, is not sufficient to actually accomplish them. The robot's software needs to be as generic and flexible as its hardware counterpart to exploit the robot's full potential. In the past, major effort was already put in the development of AI approaches to successfully accomplish a single task autonomously, e.g. a learned dynamic motion primitives (DMP) or different kinds of motion planners can be used to generate arm trajectories for variable start and target poses. The question that needs to be answered is when to use which approach. A configured DMP, for instance, can generalize over a certain range of target poses. Nevertheless, for specific commands a better performance may be reached with another DMP configuration due to higher accuracy or less energy consumption. So, when is the best point to switch between different DMP configurations? Or in which situation should I rather use a motion planner instead? Similar questions arise in other robotic domains like locomotion, e.g. how should I configure my blind reactive locomotion controller for a specific terrain or should I rather use a planning-based approach that carefully places every foot based on a map information that is costly generated from exteroceptive sensor data? Also rovers that support different locomotion modes, e.g. [Cordes et al., 2011], have to decide which one to use in which situation. In any case, integrating manifold solutions for a specific task on a single system and using them within different situations can increase the robot's operational context range.

[1] https://www.northropgrumman.com/space/space-logistics-services/mission-extension-vehicle/, accessed on 11/11/2019

Continuously adapting the executed behavior while performing the same action is also common in biological systems. The walking gait of a human is a vivid example. To traverse relatively flat ground, a central pattern generator (CPG) that is located in the spinal cord [Cummings, 2001] generates rhythmic foot trajectories, which are modulated by sensor-input-dependent reflexes to react on small irregularities. A fast reactive control loop is realized, because no deliberative decisions from the brain need to be involved. However, internal adjustments are taken to generate trajectories that match varying situations. Minor adaptations like adapting the CPG are needed when the surface interaction changes, e.g. from walking in shoes over solid ground to walking without shoes at a sandy beach. Major adaptations may also include changing the control strategy. For instance, to dryly traverse a field of puddles will probably cause the integration of visual information to avoid stepping into a puddle. Thus, the purely reactive control approach is replaced by a more costly one requiring the processing of visual input and additional brain activity. Besides environmental changes, the walking pattern selection is also depending on individual preferences. Usually, a decent speed paired with energy efficiency are the dominating concerns. In contrast, the puddle example shows that avoiding wet feet, which might result in a cold in the long run, is prioritized over maintaining the energy efficiency or desired walking speed. This might change again, e.g. when time is short to catch the next bus back home, where wet feet may be tolerated to keep up the speed, thus using again the fast and efficient purely rhythmic walking pattern.

Transferring this to robotics reveals that the control strategy itself cannot foresee all the different circumstances and adapt itself. Conscious decisions are taken by higher control levels that are based on manifold complex coherences, some of them derived from past experiences. This walking gait example shows that the walking behavior is adapted when the environment, the internal state, or the prioritization of specific action properties change. As a consequence, the internal parameters are adapted, e.g. awareness of specific sensor input and shape of limb trajectories, resulting in different behaviors that satisfy relevant performance measures on the costs of others.

Adapting the behavior, i.e. the way how the robot acts, is challenging for kinematically complex robots. Usually, the overall robot control consists of a network of diverse generic processing modules that need to work jointly together. Each of them has to be configured for every specific task which also may have side effects on others. Many times, an expert or the developer him or herself is needed for the adjustments. In cases where the overall circumstances change within the execution of an action, even online adaptations need to performed like switching information sources, tuning control parameters, or exchanging algorithms. These changes often need to be done simultaneously, which can bring even an experienced operator to his or her limits. In

many applications, this can lead to severe operation failures. Thus, the importance that a robot adapts its behavior autonomously to context changes is mandatory for future robotic applications.

1.2 Definitions

This work takes a more general view on robot control and sets upon several research fields, which all have their own understanding of terms and definitions. Using a certain one from one domain can be interpreted differently in another domain. To avoid misinterpretations, often used terms are defined here as they are used within this work. Thereby, their specific definitions also bound this work on the made assumptions and interpretations.

Behavior

According to the title, this thesis deals with the adaptation of behaviors. But what exactly is a behavior? Behavior is a term that is widely used in many fields of research. Hency, many often contradicting definitions can be found. [Levitis et al., 2009] collected and analyzed definitions of the term behavior from behavior-focussed societies and tried to formulate a consistent definition: "Behaviour is the internally coordinated responses (actions or inactions) of whole living organisms (individuals or groups) to internal and/or external stimuli, excluding responses more easily understood as developmental changes". Condensing and transferring this definition to robotics results in a definition that is used within this thesis: "A behavior is an executed movement of a robotic system with respect to its internal state and external state of the environment." Consequently, through providing actuator outputs, a behavior realizes an action. The way of doing this, is defined by the utilized motion control, i.e. the composition of algorithms, their configurations, and their interconnections.

To abstract from the actual implementation of the motion control, a behavior within this thesis is defined by a parameter vector $\boldsymbol{b} \in \mathbb{R}^B$ with B being the number of parameters. Thus $\mathbb{R}^B$ is representing all possible configuration options of a motion control. It is assumed that the activation of a specific (sub)algorithm is also encoded in one parameter within $\boldsymbol{b}$. Changing any parameter has an influence of how a robot behaves and at the end how an action is executed even though its influence is highly depending on the current situation.

Action

When a robotic system is executing a mission, it is supposed to accomplish certain consecutive tasks. Thereby, each task can be divided into several actions. Imagine, a futuristic Moon or Mars mission, where a robot should explore the surrounding to find a proper location for a base camp and to build up infrastructure. This mission can be assembled by the task of building up a map, the task of testing the surface conditions of flat spots, and numerous tasks of getting equipment from the mission lander and to install them properly. One example for the latter can be to get a box of tools, which can be decomposed into the following actions:

1. move to lander
2. locate box
3. grasp box
4. move to target location
5. drop box at target location

Usually, a state task planner derives the actions, monitors the accomplished goals and initiates the transition from one action to the next. By instantiating a typ of action, specific target values are set, which define the goal state, e.g. "move to lander" is realized by `move_to(target_pose)` with `target_pose` being the desired target position and rotation that is derived from the known or estimated pose of the lander. The granularity of actions that the actual execution engine is able to execute may also differ, so that actions are further decomposed into several sub-actions, e.g. the planning action `move_to(target_pose)` can consist of `plan_path(target_pose)` and `follow_path(path)`, which might be further decomposed until atomic actions are derived. Finally, an atomic action with numeric target values is derived that needs to be realized by the action execution engine, e.g. a motion command like `move(velocity_x, velocity_y, velocity_rot)` that provides longitudinal, lateral, and rotational target velocities.

Within this work, it is assumed that the target values are represented by a vector of real numbers, each of them describing a specific part of the action, also called action features. In addition, each action may have some side constraints or meta targets, which specify the way how the action should be realized. These meta target features do not need to be commanded explicitly. They are intrinsically defined by the application. For example, if a robot should move forward, the desired velocity would be the explicit action target value, whereas realizing the locomotion with minimal power consumption can be represented by a meta target value that is fixed for the robot-specific minimum power consumption. However, in order to abstract from specific action implementations, an action shall be defined in this work as the vector $\boldsymbol{a} \in \mathbb{R}^A$ with A being the number of target and meta target values.

Behavior Performance

A behavior causes a movement of the robot that can be measured and evaluated. The performance of a behavior can be rated with the fulfillment of an action, e.g. when the desired action defines a certain movement speed, one can measure the actual speed. The difference between desired and actual value can be used as a success measure. Thus, performance measures that characterize the execution of desired actions are further referred to as action performance features.

Especially when considering kinematically complex robots, the same action can be performed in many different manners without effecting the action performance, e.g. moving at a certain speed can be realized with plain walking, running, sneaky, jumping, in high knee running, etc. Consequently, meta performance features are introduced, that characterize other relevant performance measures, e.g. stability or energy efficiency. At the end, these features help to distinguish between behaviors which have the same action performance. Thus, the overall behavior performance is defined as vector $\boldsymbol{p} \in \mathbb{R}^A$ of action and meta performance features out of all available performance measures. Within this work, $\boldsymbol{p}$ has A elements as the desired action $\boldsymbol{a}$ because it is assumed that every desired feature has a measurable counterpart in order to be able to evaluate a behavior execution. Vice-versa, every performance feature is an additional item that can be used as action target to adapt the autonomous action execution for the current application.

Performance Prioritization

In general, many performance features can be imagined, computed, and used to evaluate a behavior. Their importance may vary not only from application to application, but also during execution of an action. Referring back to the introductory example of a human traversing a field of puddles, he or she can use different walking gaits depending on the importance of maintaining the desired speed and energy efficiency or the prioritization of maintaining dry feet. Thereby, there is no need for defining a strict hierarchy of performance features. It is desirable to have the opportunity to vary the influence of any performance feature separately. Thus, the performance prioritization $\boldsymbol{w} \in \mathbb{R}^A$ is introduced as a vector of weighting factors, one for every performance feature.

Context

If a behavior is executing an action as expected, the necessity to adapt the behavior can only arise, when something changes. Changes can abruptly occur in a discrete way, continuously vary, or almost not noticeably creep over a long period of time. An example for the latter would be the continuous wearout of systems which may lead to a varying behavior performance throughout the lifetime of a robot. In contrast, any changes that have an instant noticeable effect on the selection of a behavior are referred to as context changes within this thesis. Context itself is a highly discussed term. The definition of [Dey, 2001] quite nicely meets the understanding of context within the scope of this thesis: "Context is any information that can be used to characterize the situation of an entity ...". The entity can be interpreted as a robot whose behavior needs to be adapted. A robot is an entity that is supposed to perform an action and is situated in the world. Consequently, the context can be divided into the controllable action context which is here defined as the desired action with performance prioritization, and the uncontrollable state context that characterizes the situation of the robot. The latter surely has an influence on the performance of a behavior, e.g. fast running on flat ground will result in a high velocity, whereas the same behavior on uneven terrain will probably result in stumbling and finally slower velocity. This is an example of the influence of the external environment, but also internal aspects like the current battery level or damaged joints may be relevant for a proper behavior selection. Thus, within this work, the Z relevant and measurable criteria which have an influence on the performance of a behavior are represented by the vector $\boldsymbol{z} \in \mathbb{R}^Z$ which is further referred to as state context.

1.3 Goal

The goal of this thesis is to develop a generic approach that autonomously adapts the behavior of a robot, independently of its morphology and control approach, but with respect to the current context, i.e. the desired action to be executed as well as the state of the environment and the robot itself. It is intended, that the performance $\boldsymbol{p}$ of the executed behavior $\hat{\boldsymbol{b}}$ minimizes the difference towards the desired action $\boldsymbol{a}$ according to an error function $\boldsymbol{err}()$ and a variable performance prioritization $\boldsymbol{w}$

$$\hat{\boldsymbol{b}} = \underset{\boldsymbol{b}}{\operatorname{argmin}}\, \boldsymbol{w}^T \boldsymbol{err}(\boldsymbol{a}, \boldsymbol{p}). \tag{1.1}$$

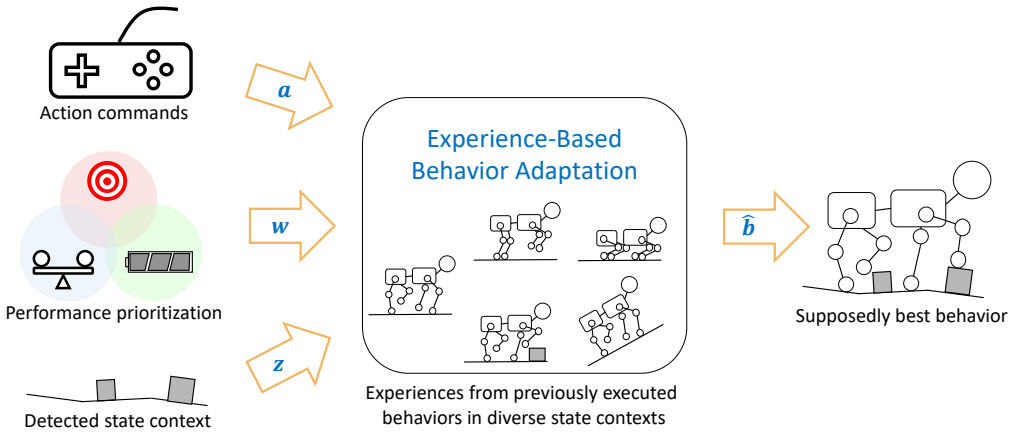


Figure 1.1: Sketch of the overall goal on the example of a robot that adapts its walking behavior depending on the control input with considering the performance prioritization, the terrain characteristics, and previously made experiences.

The approach has to consider that the performance of the executed action is not only depending on the chosen behavior $\boldsymbol{b}$ but also on the current state context $\boldsymbol{z}$

$$\boldsymbol{p} = f(\boldsymbol{b}, \boldsymbol{z}), \tag{1.2}$$

with $f()$ being some unknown, probably non-linear, function. Therefore, the objective of this work is to find a mapping $m()$ from desired action, state context, and performance prioritization to the supposedly best robot behavior

$$\hat{\boldsymbol{b}} = m(\boldsymbol{a}, \boldsymbol{z}, \boldsymbol{w}), \tag{1.3}$$

as exemplarily depicted in Figure 1.1.

The overall goal to adapt the robot behavior to context changes comes with further requirements. The first is the realtime capability. This metric is rather subjective to the field of application because the application defines how frequent context changes appear and how time critical a proper adaptation of the motion control is required. However, it is mandatory that the additional computational and temporal effort to compute $\hat{\boldsymbol{b}}$ should not make its execution obsolete.

The second criterion is the capability to transfer this approach to different control schemas, robots, or fields of application. Thus, hard-coded solutions for single use cases are not sufficient. Instead, a data-driven approach is required that learns from experiences to be independent from the implementation of the actual behavior-producing control implementation. It is desired that the robot is able to learn throughout its life time in order to continuously improve its adaptation capa-

bility but also to incorporate long-term changes like wearout that may make initially obtained experiences outdated.

The scope of this thesis is not only to develop an approach capable of achieving the ambitious goal with its side constraints but also to verify it in an extensive experimental evaluation. The selected target application comprises the online adaptation of a reactive locomotion control. Here, continuously varying motion commands and rapidly changing surface conditions pose high demands on the realtime capability of the behavior adaptation. The six-legged SpaceClimber [Bartsch et al., 2012] and the four-legged Charlie [Kuehn et al., 2017] are the target systems to analyze the different aspects of behavior adaptation in this domain. To efficiently generate experiences without damaging the hardware, a realtime rigid body simulation is used. Also experiments with the physical robots are conducted to show its applicability and transferability from simulation to the real world.

1.4 Contributions

This section briefly summarizes the thesis' contributions and links to relevant authored or co-authored publications. The developed experience-based behavior adaptation, is a generic holistic approach that augments an existing control structure through its autonomous configuration of the motion control for changing contexts. It is a unique approach that incorporates context changes that come from changing robot and environmental conditions, desired commands, and adjustable performance prioritizations. Through the abstraction of the concrete motion control, it provides an action-specific, robot-independent interface for the planning and navigation layer [Dettmann et al., 2015a]. In addition, it offers the opportunity to hande multiple control approaches for the same action.

The overall data-driven concept provides the opportunity to allow the integration and management of both, manually designed as well as learned behaviors. In addition, new experiences are continuously integrated during application to maintain an up-to-date model to derive behaviors for long-term autonomous systems [Dettmann et al., 2014, Langosz et al., 2013]. A case-based reasoning approach [Dettmann et al., 2015b] and an online incremental learning approach based a novel online Gaussian Process regression derivative are implemented for the behavior inference and are experimentally evaluated.

The autonomous behavior adaptation was successfully tested in simulation and on the physical counterparts in complex locomotion scenarios. To generate locomotion behaviors for kinematically different walking systems, a reactive modular central pattern generator approach is proposed which is implemented and used on Charlie [Dettmann et al., 2020, Dettmann et al., 2018a,

Dettmann et al., 2018b, Kuehn et al., 2020, Kuehn et al., 2018], on Mantis [Brinkmann et al., 2020, Brinkmann et al., 2019, Bartsch et al., 2016], and on SpaceClimber [Dettmann et al., 2015b]. Multiple metrics to characterize the environmental conditions as well as the performance of an executed behavior are implemented and experimentally validated.

1.5 Structure of the Thesis

The objective of this work is to extend existing control architectures by a module that adapts the robot-specific control parameters, i.e. the robot's behavior, for varying contexts. Therefore, as shown in Figure 1.2, a literature review on behavior adaptation in robotics is provided after this introductory chapter. Different methods are analyzed and a summary of their pros and cons is provided motivating the approach developed within this thesis.

Since this work is not only focussing on theory but also aims to implement and evaluate the proposed general approach, the robotic use case of adapting the locomotion behaviors for two different walking robots is selected. Therefore, Chapter 3 introduces this robotic exemplary applications and provides details on the utilized motion control. With a solid background knowledge of how the motion control works and which advantages and disadvantages are present, the foundations for understanding the behavior adaptation requirements and the experimental evaluation are laid. In addition, possible behavior performance and state context features are introduced, that are utilized within the experimental evaluation.

Chapter 4 describes in detail the overall concept of experience-based behavior adaptation and how it is applied on existing robot controls. The heart of this approach is the data-driven mapping from action and state context to behavior parameters. For this challenge, a lazy and an eager learning strategy are investigated to evaluate their strengths and weaknesses. Both, further referred to as case-based and a model-based inference, are described in detail in Chapter 5 and Chapter 6, respectively. Their concepts and implementations are presented and experimentally evaluated.

Chapter 7 presents the experimental evaluation of the proposed experience-based behavior adaptation on the example of adapting the walking gait for Charlie. Especially the difference between the proposed case-based and model-based behavior inference strategy is analysed.

Finally, Chapter 8 concludes the thesis. It summarizes the approach, analyses its evaluation, and discusses the results. At the end, an outlook on suggested directions for future work is provided. Additional miscellaneous information and material can be found in the appendix.

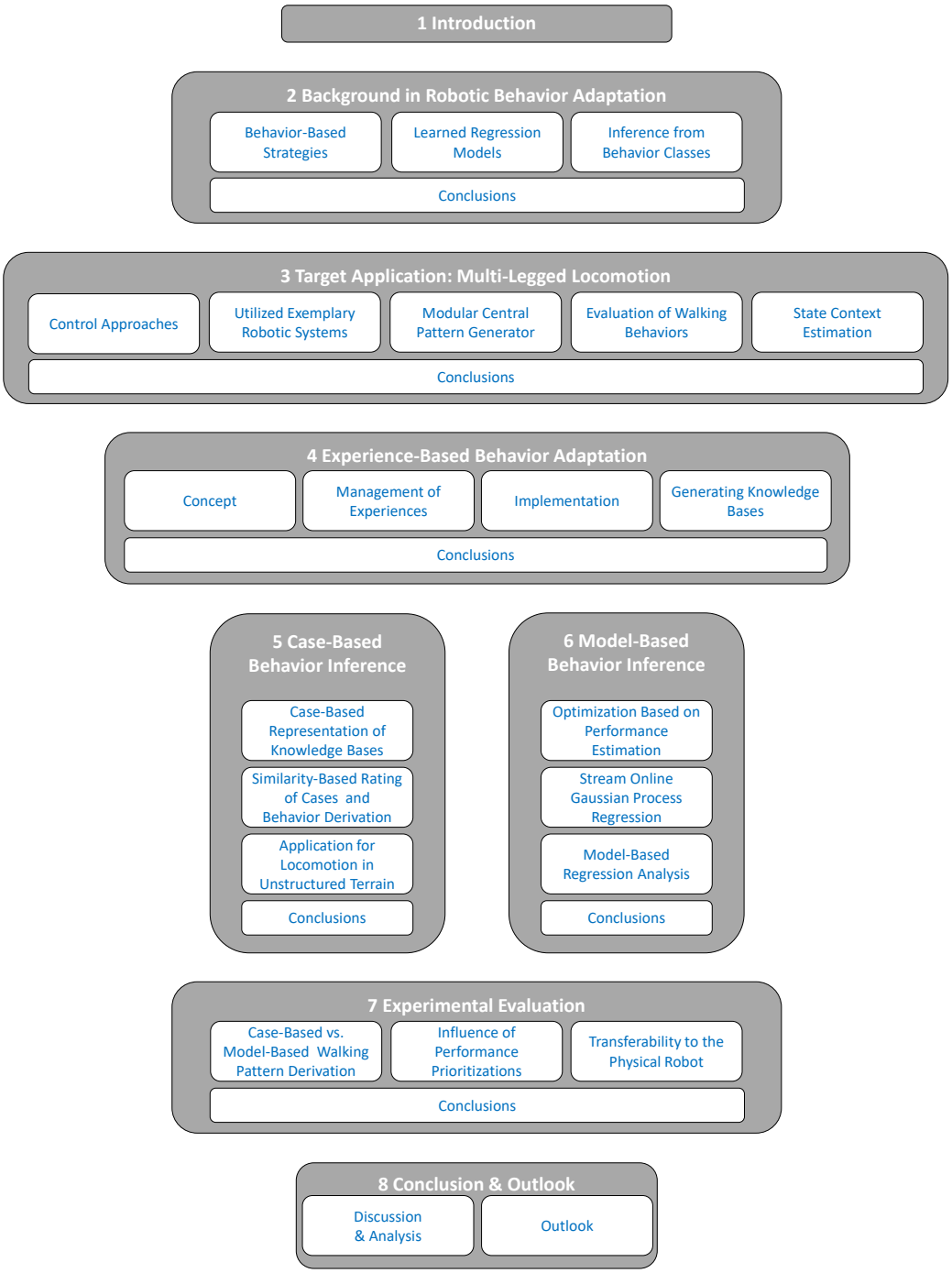


Figure 1.2: Structure of the thesis

Chapter 2

Background in Robotic Behavior Adaptation

In literature, behavior adaptation is a widely used term. It is interpretativ in various ways due to the different understandings of the term behavior. However, this section summarizes different approaches which have the same goal - a robot that autonomously changes its way of acting on the base of sensed or commanded input. First, an overview of behavior-based systems is provided with focussing on approaches that try to adapt the behavior-generating networks of behavior modules. Second, different regression techniques are analyzed. Even though they are more utilized to learn behaviors, their generalizing data-driven nature of learning a mapping from inputs to outputs offers great potential to use them within the experience-based behavior adaptation. Third, strategies from literature are presented that derive a behavior for the current context from available classes, that are given by expert knowledge or learned prior to execution. Finally, a conclusion is drawn that leads to the approach that was developed within this thesis.

2.1 Adaptation within Behavior-Based Systems

Behavior-based adaptation strategies base on modular behavior-based architectures, that consist of interconnected behavior producing modules that all contribute to the overall system behavior. In literature, the behavior modules are also referred to as behavior functional modules, behavior producing modules, or short behaviors.

This type of control architecture started to become famous in the mid 80s with the subsumption architecture [Brooks, 1986]. Instead of defining functional modules, this approach introduced goal-orientated behavior modules which directly receive sensor inputs and create actuator outputs. Originally, they are hierarchically ordered. Higher prioritized layers can inhibit or subsume signals from lower lay-

ers. The resulting modular architecture is built beginning from lowest layer, testing it, then advancing to the next layer, testing both, and so on. Thus, by adding new layers, new and more complex functionalities are added. The winer-takes-all arbitration scheme clearly defines which behavior produces the actual output signals. Also cooperative merging strategies are common to emerge a joint actuator output based on parallel acting behavior modules, e.g. potential fields within the motor schema architecture of [Arbib, 1992]. However, modifications and extensions led to more modern behavior-based architectures to control walking robots, e.g. [Cruse et al., 1998, Spenneberg and Kirchner, 2007, Albiez, 2007, Bartsch, 2014] or to simultaneously handle several actions with whole body control approaches as originally introduced by [Sentis and Khatib, 2006], which opened up a new field of research.

As long as the behavior network comprises several behavior modules, the developed behavior networks are still manually manageable. However, the more functionality is needed, predefining all behavior modules and interconnections as well as defining weights for the arbitration becomes difficult and impractical. Newly added behavior modules require more coordination effort and their influence is less predictable. Consequently, a way to organize many behavior modules becomes necessary. In addition, the output of a behavior network depends on the inputs and the internal state. Thus, influences which have an effect but are not considered in the control architecture may lead to misbehavior and usually demands to rebuild the behavior-based system from scratch. To account for this, a behavior adaptation process is desirable that intelligently activates behavior modules or modulates their signals if needed. Existing approaches from literature can be categorized into (1) decentralized approaches, i.e. where every single module decides when and how to adapt, and (2) the introduction of supervision modules that can influence every single behavior module.

One example for the former is that each behavior independently tries to find out if it is correlated to positive feedback and under which conditions it becomes reliable, i.e. the conditions in which it rather produces positive than negative feedback. [Maes and Brooks, 1990]. So, by defining positive and negative feedback functions, each of the utilized behaviors can learn from experience when it is appropriate to become active. The learned correlations can be stored and associated to the utilized feedback functions. The feedback functions are derived from application-specific needs and can be exchanged together with the learned correlations to adapt for different scenarios.

Even though collecting experiences in form of positive and negative feedback in a decentralized approach is an interesting attempt, it faces practical difficulties. First, to extend an existing control structure with such a functionality, every single mod-

ule needs to be augmented by the proposed learning capability including a kind of knowledge management. And second, the stored correlations are only valid for a distinct network of behavior modules which implicates the necessity that every module knows the overall network topology in order to select the valid learned correlation.

The motivated behavioral architecture [Michaud, 2002] is an example for supervised adaptation of behavior networks. It uses motivational modules to derive the robot's intentions and to recommend the selection of behavior module sets that define the activation or inhibition of certain behavior modules. An activation module determines which behavior modules are to be activated according to the recommendations of the motivational modules. These are divided into three categories: instinctual, rational, and emotional. Instinctual motivation modules provide basic operations, e.g. safe point to point movements, whereas rational modules are related to deliberative processes were reasoning is involved. The emotional motivation modules monitor conflicting tasks or transitional situations and try to resolve them with appropriately activated behaviors. Based on the situation-specific recommendations of the motivational modules, the way how the robot behaves changes.

In [Burattini and Rossi, 2010], the impact of emotional modulation on behavior-based systems is studied. Here, the underlying architecture is based on periodic behaviors modelled as perceptual and motor schema, derived from Schema Theory [Arbib, 1992]. In order to introduce a context-dependent adaptation capability, an emotional module is introduced, that can adapt the execution period of behavior modules based on specific events. Thus, the focus on specific sensor values can be increased or decreased resulting in a changed emergent overall behavior. The state of the emotional module itself depends on the correlation between specific sensor readings and behavior outputs, e.g. when the input stays quite constant but a behavior module is continuously generating outputs at a high update rate, it will reduce the module's frequency.

Even though all of the above examples have interesting attempts to adapt a robot's behavior, every one of them is bound to its underlying behavior module representation and can hardly be transferred to other applications. Thus, only partial aspects can be utilized within the scope of the thesis, like keeping track of positive and negative influences and the consideration of non-functional external and internal factors for the behavior adaptation.

2.2 Adaptation through Learned Regression Models

As stated in the mathematical goal description (Section 1.3), a non-linear mapping between multiple inputs (context) and outputs (behavior) is needed according to Equation (1.3). Due to its complexity, pure analytical models can hardly be derived

and if, they cannot be transferred to another domain without carefully obtaining every single model variable. In contrast, supervised machine learning techniques, such as learning from demonstration, are capable to learn from provided samples that are likely to be generated in many fields of application [Argall et al., 2009]. This data-driven approach has a good transferability, as long as similar complexity and processing capabilities are present and, thus, is a potential key concept for this thesis. Learning from demonstration involves several steps. First, an action is demonstrated and the corresponding samples of inputs and outputs are collected. Then, after collecting plenty of samples, i.e. having enough training data, a generalizing model can be trained. Finally, when an appropriate model is learned, it can be used to predict outputs for new given inputs. The term classification is used when discrete outputs are predicted, whereas regression refers to the prediction of continuous outputs, which is needed to derived real-numbered behavior parameters.

To provide a solid background on regression, the principle of regression is introduced in the following by the example of linear regression. Afterwards, kernel-based methods are summarized that are superior for modeling unknown, non-linear input-output correlations. Then, incremental learning approaches are summarized that approximate the standard regressions for the sake of applicability in real-life applications. Finally, an approach to combine model learning with parameter optimization is presented.

Classical Regression Approaches

A regression model predicts a continuous output $y_* \in \mathbb{R}$ based on the current inputs $\boldsymbol{x}_* \in \mathbb{R}^N$ and a given data set $D = \{y_s, \boldsymbol{x}_s\}_{s=1}^S$, which is a collection of S samples, each consisting of an output y_s and an input vector $\boldsymbol{x}_s$. A vector of multiple outputs can also be predicted, either by using multiple regression models or by using a multivariate regression. However, for the sake of simplicity, only the prediction of one scalar outputs is considered here.

A linear regression assumes that a vector $\boldsymbol{m}$ exists, that maps all input samples $\boldsymbol{X} \in \mathbb{R}^{S \times N}$ to the corresponding output samples $\boldsymbol{y} \in \mathbb{R}^S$ with an unknown error $\boldsymbol{\epsilon}$

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{m} + \boldsymbol{\epsilon}. \tag{2.1}$$

To estimate the parameter vector $\boldsymbol{m}$, a large number of procedures have been developed. Ordinary least squares, for instance, minimizes the sum of the squared residuals

$$c_{ols}(\boldsymbol{m}) = ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{m}||^2 \tag{2.2}$$

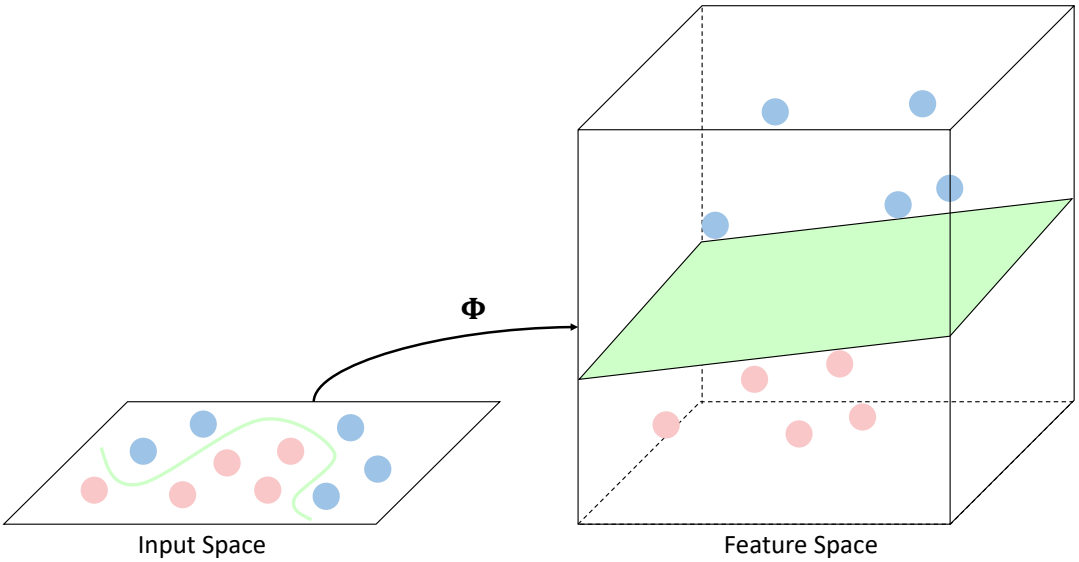


Figure 2.1: Transforming the input variables to higher dimensional feature space may yield a linear mapping for a non-linear mapping in the original space.

leading to the closed-form solution for $\boldsymbol{m}$

$$\boldsymbol{m} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{2.3}$$

The final prediction step results in

$$y_* = \boldsymbol{x}_*^T\boldsymbol{m}. \tag{2.4}$$

The linear combination of the parameter and input variables can also be used to model non-linear mappings by transforming the input variables by $\boldsymbol{\Phi}(\boldsymbol{x})$

$$y_* = \boldsymbol{m}^T\boldsymbol{\Phi}(\boldsymbol{x}_*), \tag{2.5}$$

e.g. by squaring the inputs and adding these as additional elements.

The transformation into higher dimensional spaces allows to find a linear mapping within this space for a non-linear mapping in the original space (Figure 2.1). Polynomial regression, for example, can fit the estimated output to any arbitrary function. Overfitting the input data and loosing the generalization capability can be a problem and should be avoided especially when noisy output data is expected. Thus, many variants, such as lasso regression (L1-norm) and ridge regression (L2-norm), use extra information in form of a regularization term in the cost function to prevent unreasonable solutions, e.g. by penalizing the complexity of the solution.

Even though the transformation to a higher dimension helps to find an accurate model, the selection of a proper transformation $\boldsymbol{\Phi}(\boldsymbol{x})$ can be cumbersome and has to be repeated for every new problem. In addition, the model derivation requires usually the computation of multiple dot products between the inputs of samples to find their similarity, i.e. $\boldsymbol{\Phi}(\boldsymbol{x}_m)^T\boldsymbol{\Phi}(\boldsymbol{x}_n)$ between sample m and n, which is computationally intensive especially for higher polynomials.

Kernel-Based Regression

Kernel methods use the fact that dot products in the original space raised to a power are equal to dot products in the higher dimensional space avoiding the evaluation of higher polynomials, which is commonly known as the kernel trick:

$$\boldsymbol{\Phi}(\boldsymbol{x}_m)^T\boldsymbol{\Phi}(\boldsymbol{x}_n) = k(\boldsymbol{x}_m, \boldsymbol{x}_n). \tag{2.6}$$

Through the kernel trick, a linear algorithm operating in a higher dimensional feature space will behave non-linearly in the original input space [Schölkopf et al., 2002].

A well-designed kernel function $k(\boldsymbol{x}_m, \boldsymbol{x}_n)$ can find similarities in even infinite dimensional spaces, without performing computations in the infinite space, e.g. the Gaussian kernel as an example for a Radial Basis Function (RBF) kernel:

$$k(\boldsymbol{x}_m, \boldsymbol{x}_n) = \sigma_f^2 \exp\left(-\frac{||\boldsymbol{x}_m - \boldsymbol{x}_n||^2}{2l^2}\right), \tag{2.7}$$

where $||\boldsymbol{x}_m - \boldsymbol{x}_n||^2$ represents the Euclidean distance between the two input feature vectors. The hyperparameter l allows to balance the kernel between overfitting the data or a plain linear projection. The hyperparameter σ_f scales the Gaussian curve which makes the regression more adaptive to the signal variance. Kernel Ridge Regression, for instance, combines the kernel trick with ridge regression, a linear least square variant as loss function using L2-norm regularization to penalize the complexity of the model avoiding a fitting to the noise of the data.

Support Vector Regression (SVR), or specifically ε-SVR [Drucker et al., 1997], uses a loss function that ignores errors that are within a given distance (ε) and computes the loss towards that ε-boundary while obtaining a possibly flat curve. The prediction only depends on the utilized support vectors that are on the ε-boundary resulting in a sparse model that accelerates the prediction time. As it is difficult to select an appropriate ε, υ-SVR [Schölkopf et al., 1999] is a variant that provides an estimate of ε by controlling the number of support points and training errors. Hence, given the same parameters, the regressive function automatically adapts to the noise.

Gaussian Process Regression (GPR) [Rasmussen and Williams, 2006] is the kernel version of Baysian Linear Regression, both assuming an additive Gaussian noise. The regression function is a linear combination of weighted kernel functions, evaluated on tests points:

$$y_* = \mu_* = \sum_{s=1}^{S} \alpha_s k(\boldsymbol{x}_s, \boldsymbol{x}_*) = \boldsymbol{k}_*^T \boldsymbol{\alpha}, \tag{2.8}$$

where

$$\boldsymbol{\alpha} = (\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{y}, \tag{2.9}$$

with σ_n being the signal noise, $\boldsymbol{I}$ the identity matrix and $\boldsymbol{K} = \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X})$ the covariance matrix or kernel matrix, i.e. it holds the outcome of the kernel function for every combination of samples. $\boldsymbol{k}_* = \boldsymbol{k}(\boldsymbol{X}, \boldsymbol{x}_*)$ is the kernel function applied between the current input and the respective inputs of the kernel elements.

Every element of $\boldsymbol{\alpha}$ can be interpreted as derived prediction value from a sample point that is influenced by its own noise and by all other sample outputs weighted by their input similarity. Through $\boldsymbol{k}_*^T$, these values are weighted by the similarity between query point and corresponding sample point.

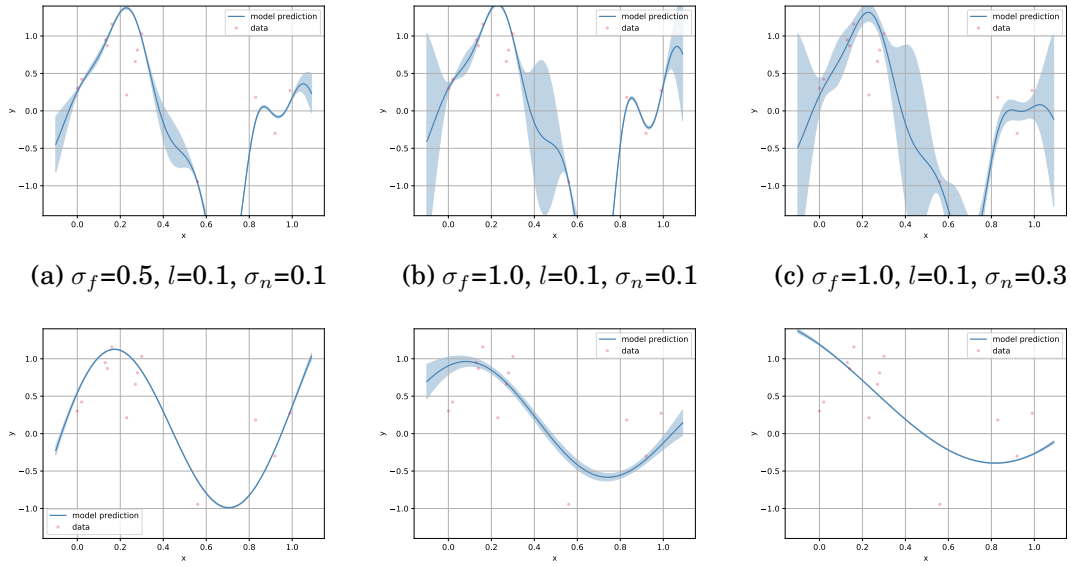
One benefit of GPR is, that the variance σ_*^2 of the Gaussian Process can be used as uncertainty measure u_* for the estimated output y_*

$$u_* = \sigma_*^2 = k_{**} - \boldsymbol{k}_*^T (\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{k}_*, \tag{2.10}$$

where $k_{**} = k(\boldsymbol{x}_*, \boldsymbol{x}_*)$ is the kernel function applied to the input and itself yielding the input samples' variance.

Figure 2.2 shows the influence on the prediction and uncertainty estimation of different GPR hyperparameter combinations using a RBF kernel. For model training served 15 samples derived from a sine signal with additional 50% white noise, which are plotted with red dots. Figure 2.2d shows that GPR is capable of predicting the target function with few samples despite the noisy input data. However, the data could also have been recorded from another signal source. With adapting the GPR hyperparameters, various signal types can be predicted. Thus, prior knowledge of the expected signal can help to tune the model. Taking Figure 2.2d as reference, decreasing l increases the fitting of the prediction curve to the target values (Figure 2.2b), whereas high values flattens the possible prediction curve (Figure 2.2f). With decreasing σ_f, less signal variance is allowed (Figure 2.2a). Increasing σ_n, reduces the need of predicting the exact value at a sample input, thus generates smoother curves but also increases the uncertainty (Figure 2.2c) and (Figure 2.2e).

Despite good estimation accuracy, training any model (or its variants) mentioned above involves always the whole data set. Whenever new data arrives, the whole



(a) σ_f=0.5, l=0.1, σ_n=0.1 (b) σ_f=1.0, l=0.1, σ_n=0.1 (c) σ_f=1.0, l=0.1, σ_n=0.3

(d) σ_f=1.0, l=0.5, σ_n=0.1 (e) σ_f=1.0, l=0.5, σ_n=0.3 (f) σ_f=1.0, l=1.0, σ_n=0.1

Figure 2.2: GPR prediction and uncertainty (shaded region) for noisy sine data utilizing a RBF kernel with different hyperparameter combinations

batch is used for training resulting in constantly growing training time, which is especially severe for approaches like GPR that have a $O(s^3)$ complexity. Such a computational effort is unacceptable within the scope of long-term autonomy, where the training process is running almost infinitely and new data is continuously incoming.

Incremental Learning Approaches

Long-term autonomous robots actually never finish training, i.e. there is no point in time where the model is final and exclusively used for prediction. Thus, incremental learning variants have been developed that allow retraining the model only with the use of the latest sample which has the advantage that the need to store the history of raw data is eliminated.

Locally Weighted Projection Regression (LWPR) [Vijayakumar and Schaal, 2000] is an incremental online learning method that uses the weighted mean of several local linear regression models to predict outputs for high-dimensional inputs. The weights are determined by a distance metric between query point and location of the linear model. Due to the small models, it maintains a reasonable update time for realtime applications. LWPR is used in many applications, e.g. [Grollman and Jenkins, 2008] uses it to train skills for an AIBO robot and [Stulp et al., 2013] use LWPR to learn DMP parameters for demonstrated arm trajectories with the small-scale humanoid iCup robot [Metta et al., 2008]. In the lat-

ter example, obstacles of different heights were placed in the way between start and goal position and a different trajectory was demonstrated for each obstacle height. In the end, a proper trajectory was generated depending on the obstacle height.

Similarly, [Nguyen-Tuong et al., 2009] have proposed a local version of GPR, Local Gaussian Process regression (LGP), where the regression model is divided into several smaller local Gaussian models which are segmented by input distance. The output is calculated as the weighted superposition of these local model predictions. In [Ranganathan et al., 2011], an online sparse matrix Gaussian processes method is presented, which is based on maintaining and updating the sparse Cholesky factor of the kernel matrix using Givens rotations. It is shown that with local support, the Kernel matrix is typically sparse, which makes the update very efficient.

Another category of online Gaussian process approximation is targeting at intelligently selecting a subset of the training set to present a larger data region, e.g. [Csató and Opper, 2002], [Lawrence et al., 2003], or [Seeger et al., 2003]. The drawback of this active set approximation is that reselecting the active set causes non-smooth fluctuations in the marginal likelihood and its gradients, thus, increasing the difficulty to derive optimal hyperparameters. [Snelson and Ghahramani, 2006] suggest to generate pseudo inputs to bias the computation. The active set is generated by gradient-based optimization and is not constrained to be a subset of the data. The method was then generalized by [Naish-Guzman and Holden, 2008]. Nevertheless, the extra optimization step to generate pseudo inputs turns out to be impractical for high-dimensional input spaces. [Bijl et al., 2017] propose a Sparse ONlIne Gaussian process regression (SONIG) and implemented it in a non-linear black-box system modeling. The algorithm selects inducing input points for training. Their results show that this regression algorithm can incorporate noisy measurements while maintaining high accuracy.

However, the aforementioned online versions of algorithms are only trying to decrease the complexity order, which is not sufficient for unlimited stream data regression. To ensure constant computation time, the kernel matrix size needs to be limited. In order to avoid the oblivion of new data when the kernel size is reached, a strategy is needed to remove a kernel matrix entry for the sake of inserting a new one. The general aim is that the elements stored in the limited kernel matrix cover the relevant state space. For this reason, [Nguyen-Tuong and Peters, 2011] developed Sparse Incremental Gaussian Process Regression (SIGPR) which computes an independence measure for every kernel matrix element that indicates which element provides least information and can be replaced next. The insertion of new kernel elements is only considered when the new independence measure is higher than the least of the kernel elements. Even though this method provides a good coverage over the data space, handling noisy data is not considered. This can lead to the problem,

that an outlier may be selected as kernel element that will lead to the discard of every sample within its vicinity resulting in an outlier-biased regression.

Also online versions of SVR exist that avoid batch learning and thus provide the capability to adapt the model with newly incoming samples. They either iteratively add new support vectors on the costs of growing computational complexity [Laskov et al., 2006, Parrella, 2007] or simultaneously remove old support vectors when new ones are added that slightly lowers the accuracy [Van Vaerenbergh et al., 2010, Fabisch et al., 2015, Krell, 2018].

In contrast to kernel methods that learn a representation of the data by assigning weights on training points and reusing them for prediction, using an artificial neural network (ANN) is an alternative approach. Here, the model is pre-defined by a network of interconnected artificial neurons or perceptrons [Rosenblatt, 1958], i.e. from biological neurons abstracted processing nodes that transforms multiple inputs to one output by summing up the inputs, adding a bias, and feeding this to a (usually non-linear) activation function. A weight is assigned to each connection between neurons defining the importance of a neurons output. Thus, by iteratively adapting the weights when seeing data enables the network to learn the input output correlations. The network is typically organized in layers, i.e. an input layer, an output layer, and variable size of hidden layers in between each consisting of a variable number of neurons. Networks that only feature connections from one layer to the next are called Feedforward Neural Network (FNN). In contrast, networks that allow connections to previous layers are called recurrent networks introducing a memory effect and allowing to also learn correlations between subsequent input data. The introducing of the back-propagation algorithm [Rumelhart et al., 1985] with a gradient descent scheme to learn the weights between neurons opened up the applicability of ANN in many machine learning applications. The drawback of classical neural networks is that the performance heavily depends on the initialization of the weight matrix. Poor initialization sometimes prevents a successful training which finally would lead to unexpected autonomous robot decisions. Nowadays, Deep Neural Networks, i.e. network architectures with multiple hidden layers, are very popular to solve complex learning problems. The learning of tremendous amount of weights is possible through the ability to efficiently compute neuron outputs in parallel on dedicated hardware such as graphical processing units, the availability of large data sets, and the improvements of modelling and learning techniques such as using improved activations functions and Reinforcement Learning (RL) for efficient training [Goodfellow et al., 2016]. Especially, the introduction of convolutional layers allows to process time-series and visual data very efficiently providing solutions for many applications, such as electroencephalogram data processing [Schirrmeister et al., 2017], speech recognition [Amodei et al., 2016], and image

recognition [Wu et al., 2015]. Nevertheless, the huge amount of required training data is a drawback in many applications. In addition, to determine the number of neurons and layers requires a lot of expert knowledge.

In all of the presented approaches, only one performance criteria is considered which is mainly the error between predicted outputs for test inputs and their respective known or measured targets. They have to be embedded in a larger framework if multiple performance criteria are used and an online adaptable performance prioritization is desired, as required to meet the goal of this thesis.

A Posteriori Optimization

In the examples above, repeating the demonstrated actions as close as possible is the only performance metric. Whenever different performance measures come into consideration, adapting a behavior is also often understood as optimizing or re-optimizing a behavior under changed conditions, i.e. environmental changes, new targets, or their prioritizations.

In [Calinon et al., 2013] for instance, pan cake flipping with a 7-DOF robotic arm is demonstrated with kinesthetic teaching. Due to the fact that the physical guidance interferes the task itself, the initially acquired trajectory is optimized afterwards in terms of flying height of the pancake, landing in the middle of the pan, and landing on the opposite side. Here, RL is used to optimize the initial solution over several iterations. Whenever a discrete context change happens, e.g. a new pancake with more weight, a new optimization needs to be triggered.

Even though there are RL approaches that try to minimize the amount of iterations to derive the optimal solution, triggering an iterative adaptation when context changes appear rather frequently is not desired, because the adapted behavior would be obsolete before the adaptation process has ended. In addition, exploring new solutions from an old optima over potentially local minima towards the new optimum may be not applicable or dangerous in some use cases.

2.3 Adaptation through Inference from Behavior Classes

In many applications, a situation change may require an instantaneous behavior adaptation to efficiently achieve a certain task. A fast and reliable way would be to derive a solution from a finite set of working, already tested options. Different variants with this underlying procedure can be found in literature. In the following, approaches are analyzed that infer a solution from (1) pre-defined classes, (2) example cases, and (3) a behavior-performance map.

Inference from Pre-Defined Behavior Classes

In robot control and operating systems, the common way of developing robot software is to implement general configurable modules that can be utilized and configured for different target applications, as done in ROS[1] or ROCK[2]. In order to use a specific module, it is usually first connected to other modules, sensors, or actuators, then configured for the current use case, and finally started. Within the configuration step, different module options and parameters are set to tune the algorithm for optimal performance. Consequently, a set of parameters, which corresponds to a behavior in this thesis, is selected from several, pre-given parameter sets. The available options are usually defined by the systems engineer who sets up the robot software for different use cases and the configuration step is usually done before the action is executed. A skillful operator would be needed to make any further modifications within the action execution.

One example where the online adaptation of parameters is useful, is to select the right parameters for a motion planner. Depending on the situation, a good parametrization may increase the success to plan a trajectory or to minimize the required planning effort. This is shown by [Denny et al., 2018] within the domain of sample-based planning. They propose to manually divide the manipulation area into regions of different difficulty. For every specific region, a different sampling-based planner with specific configuration is defined that optimally works within this context. Searching for a global path triggers the efficient local planning within every specified region with the region-specific parameters. Then, the local plans are connected to generate one efficient global path. This approach efficiently generates global paths on the costs of manually dividing the planning space by an expert for every new situation.

In [Morales et al., 2005], the authors avoid the manual intervention of defining regions with using a trained classifier that detects the region based on features that describe the planning space and detected obstacles. Therefore, in an offline step, example planning spaces are iteratively divided until three different classes can be assigned: free space, corridor, or unstructured. For every class, a planner configuration is defined that usually has good performance for the specific problem class with respect to the application goals. During the planning process, the type of region is automatically detected with respect to the measured features, and several sub plans are efficiently generated and afterwards connected. Unfortunately, the generation of labeled data is needed to train the classifier, i.e. different example problems for every class need to be generated before the approach can be applied.

[1] ros.org, accessed on 10/11/2019
[2] rock-robotics.org, accessed on 10/11/2019

Other approaches define discrete classes and solutions for them, but go beyond a simple selection procedure. Fuzzy logic, for instance, selects rules on how to solve a problem when specific criteria are met. The selection process is soften by taking into consideration how much a rule is applicable, thus inferring continuous outputs between pre-defined solutions. In [Gassmann, 2007], a fuzzy logic is implemented, that links surface properties to specific behavior parameters of a walking robot. Even though the fuzzy logic is creating a smooth behavior adaptation, the setup of rules including their influence intervals is hard-coded. It requires much expert knowledge and carefully performed tests to tune the corresponding hyperparameters which has to be repeated for every new system and does not consider performance mitigation during a system's life time.

To conclude, inference from classes is an efficient way to generate a behavior for changing contexts. Nevertheless, relying on pre-defined classes does not fulfill the requirement of transferability to other applications. Consequently, methods that derive the utilized classes from data are more appropriate.

Case-Based Reasoning

One data-driven approach to find a mapping between any inputs and outputs is case-based reasoning (CBR). It is an artificial intelligence paradigm and computational problem-solving method that solves new problems by adapting previous solutions for similar problems. The underlying assumptions are that (1) past solved problems reoccur in future and (2) the same action under the same conditions will likely create the same or at least a similar outcome [Leake et al., 1996]. Thus, small changes in the situation require small changes in the solution, although old solutions would also work with less performance. A case represents a problem-solution pair, i.e. past experiences. Multiple cases are collected in a case base, or also called memory.

According to [Kolodner, 1992], the CBR working cycle can be divided into four major phases:

1. Case Retrieval: For any incoming problem formulation, the best matching case or a few candidates for further investigation are searched in the case base. The retrieval is usually based on a similarity measurement that compares features characterizing the problem.

2. Case Adaptation: Since new situations rarely match old ones exactly, retrieved case solutions need to be adapted according to the difference of the current problem and the one from the case.

3. Case Evaluation: The adapted solution is evaluated. This can be done before its application, also called criticize, or after the solution has been applied.

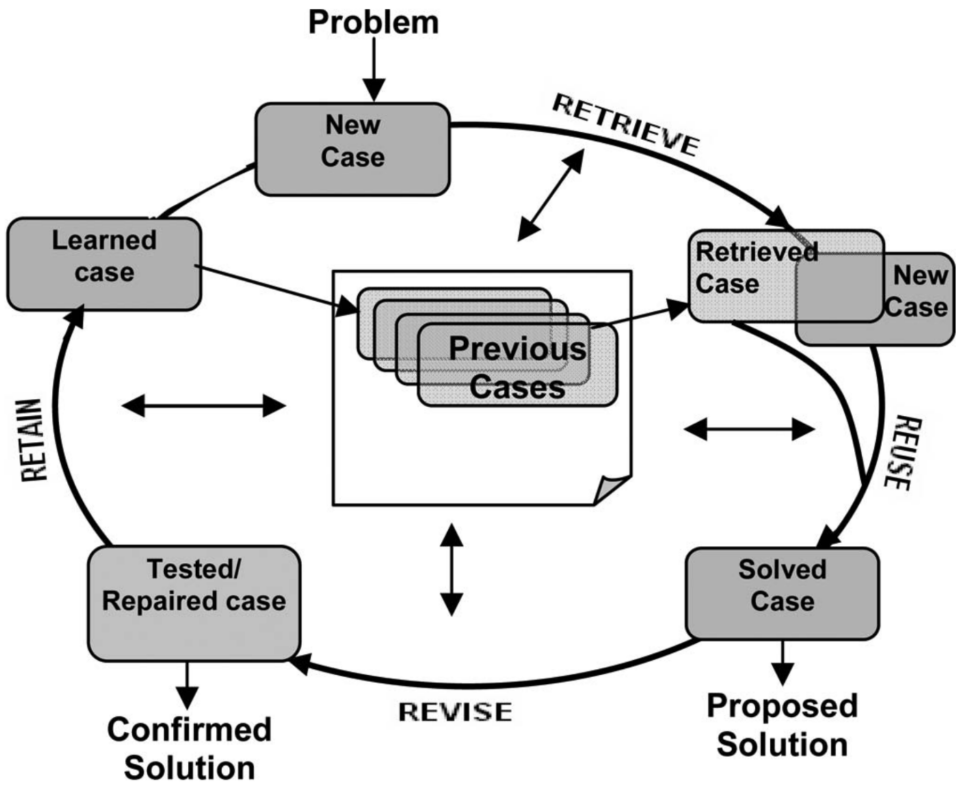


Figure 2.3: "4 REs" CBR cycle [Aamodt and Plaza, 1994]

4. Memory Update: If the solution is evaluated, the new problem-solution pair can be stored as a new case in the case base, thus, being available for upcoming solution queries.

These steps form a cycle and are also called the "4 REs": Retrieve, Reuse, Revise, and Retain [Aamodt and Plaza, 1994]. They are also in some sense recursive creating smaller loops within the big loop, e.g. case criticism or evaluation may lead to a new case retrieval or adaptation (Figure 2.3).

The main advantage of CBR is that no explicit domain models are needed that would require time-consuming expert knowledge elicitation. CBR provides a reasoner that quickly proposes solutions in domains that not necessarily need to be understood completely [Hüllermeier, 2007]. Solutions are derived when no algorithmic method is available. They will not always be optimal, but especially in domains, where the problem space is only sparsely covered, plausible solutions are generated. By evaluating and storing "good" or "bad" experiences, the CBR system can incrementally learn and improve over time. Furthermore, the implementation effort to apply a reasoner to another application is reduced to identify significant problem-describing features and to build up a case base.

It has been argued that the CBR process is also common in human problem solving [Leake et al., 1996]. For example, considering a doctor that faces a patient with an unusual combination of symptoms. If he or she has seen similar symptoms before with another patient, he or she probably remembers the old case and proposes the old diagnosis as a solution for the new problem. Of course, his or her hypothesis might be incorrect and the doctor must still validate the assumption. Nevertheless, it is a good starting point that might avoid a time-consuming analysis from scratch. Other daily examples are easy to find, e.g a car mechanic that needs to repair a car with certain malfunctions will likely recall a previous repair of another car with similar symptoms, or in court, the explicit usage of precedence cases to derive the interpretation or prioritization of laws to decide for subsequent cases is very prominent.

Expert systems using CBR are developed for above mentioned use cases and other domains, e.g. estimation of real-estate prices. Furthermore, CBR is used as a powerful method for computer reasoning, e.g. for reactive navigation [Urdiales et al., 2006], for plan selection in real-time strategic games [Aha et al., 2005], or adapting the game play of soccer robots [Ros et al., 2006].

A major challenge when designing a case-based reasoner is to define an appropriate adaptation rule. Often, a fixed rule is used that is hard-coded by a domain expert. In some approaches, e.g. [McDonnell and Cunningham, 2006] and [Jalali and Leake, 2013], adaptation rules are learned by comparing cases to infer how differences in the problem descriptions suggest solution differences. CBR may be problematic on dynamic problem domains. A frequent shift of how problems are solved leads easily to an outdated data base. In addition, the linearly growing memory costs and retrieval time with each case can be problematic in high-dimensional problem spaces. Also noisy problem characteristics can lead to a fast growing case base without adding more knowledge, because the same case is more or less processed several times. Consequently, appropriate methods are required to decide which cases are memorized and which need to be forgotten [Kira and Arkin, 2004]. Also, a proper memory management and efficient case retrieval is needed [Pantic, 2005]. [Leake and Sooriamurthi, 2001] propose to combine multiple smaller case bases to speed up the case retrieval process. When applying a case-based reasoner, one has to consider, that the generalization capability is highly coupled with the input data. Especially sparse problem space coverage may lead to reasoning based on anecdotal evidence. Thus, statistical relevance should be considered to obtain reliable solutions.

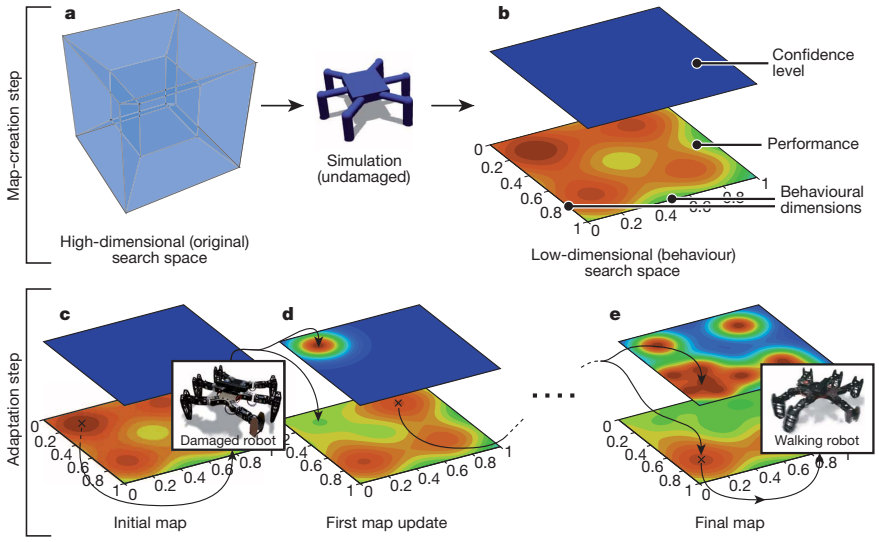


Figure 2.4: Behavior performance map approach [Cully et al., 2015]

Behavior Performance Map

Another sophisticated behavior selection approach is to infer a parameter set for the current action based on a behavior performance map, which is a repertoire of evaluated behaviors. In [Cully et al., 2015], the authors generate a behavior performance map for a six-legged walking robot (among other examples) by testing millions of behavior parameter sets in simulation (Figure 2.4). To reduce the state-space complexity, they abstract the high-dimensional parameter space to a low-dimensional behavior description space that characterized the behavior execution, in this specific case how long each foot has ground contact during a step cycle. Only the best performing parameter set is stored as representative behavior for the discretized abstract behavior description space. Its performance, which is a scalar computed from an application-specific fixed fitness function, is stored in the behavior performance map. The map itself is modeled as a Gaussian process, which can estimate the performance of any given behavioral descriptor. A second map describes the uncertainty which is initialized with a constant value.

After this offline phase, the robot can be deployed in reality or even with a broken body part, both changing the real still unknown behavior performance map. Nevertheless, the proposed intelligent trial and error approach is now testing the supposedly best behavior which is computed from the offline-generated behavior performance map plus uncertainty. It is likely, that it does not perform well, because the

simulated robot behaves differently due to the simulation reality gap. However, the execution and evaluation generates new information that are used to update the corresponding value in the behavior performance map. In addition, the corresponding entry in the uncertainty map is set to zero. The update in both maps affects the neighboring behaviors as well through the Gaussian process modelling. If a performance drop is encountered, the new overall best behavior is likely to be found in another region of the behavior performance map. So, a different behavior has now the highest performance value and is applied next. This is potentially repeated a couple of times until a behavior is found that has not suffered from a large performance drop, thus being appropriate for the current context. In other words, as soon as the context changes, the behavior performance map is explored in an efficient way, to generate suitable test behaviors in few steps.

This is very efficient when the behavioral description fits to the application, e.g. stance time of a foot for an application where a leg is later injured and cannot be used for locomotion. So, prior domain knowledge is needed for an efficient mapping from parameter to behavior description. Another drawback is the behavioral abstraction layer in between because it prevents reasoning in the original parameter space. Consequently, only already tested parameter sets can be selected and the generation of new ones is impossible. In addition, many evaluation results are discarded, because they have a worse performance than the representative parameter set for the corresponding behavior description. But maybe a discarded parametrization might be helpful for an upcoming situation or sometimes bad examples might give relevant information about what to avoid. The usage of a GPR provides a powerful way to model the non-linear behavior performance map. Due to the offline performance computation of one scalar value based on weighted features, an online performance prioritization is not applicable that limits the flexibility of this approach.

2.4 Conclusions

In this chapter, different behavior adaptation approaches are analyzed. Summarizing their strengths and weaknesses with respect to the goals of this thesis, non of the approaches alone is appropriate to reach all of them (Figure 2.1).

Behavior-based adaptation strategies rely heavily on the underlying representation of the behavior modules and are difficult to transfer to other systems, especially when no component-based architecture is used. In contrast, learning a model from demonstrated samples to model an unknown relationship between inputs and outputs seems to be a promising way to realize the desired mapping between context and behavior parameters. Several regression methods including their incremental versions show promising results. Especially, GPR seems to be a powerful tool, that

Table 2.1: Summary of common behavior adaptation approaches, where green depicts positive attributes, red represents negative attributes that are in conflict with the goals of the thesis, and orange refers to attributes that are in between.

Approach	Adaptation Type	Adaption to Context Change	Life-Long Learning	Online Performance Prioritization
Behavior-based strategies	Intrinsic activation of behavior-producing modules	Continuous	Yes	Possible
Incremental Learning Approaches	Training a model based on demonstrated target values	Continuous	Yes	No
Learning from Demonstration + A Posteriori Optimization	Optimizing a learned model after discrete context change	Discrete	Yes	Yes
Inference from Pre-Defined Behavior Classes	Inference of solutions based on similarity to pre-defined classes	Discrete or Continuous	No	Possible
Case-Based Reasoning	Inference from example cases	Continuous	Yes	Yes
Behavior Performance Map	Selection in reduced space based on performance map	Discrete	Yes	Possible

can provide accurate predictions and also uncertainty estimations while only few hyperparameters need to be tuned. However, most applications focus on reproducing the demonstrated action as close as possible. Only few approaches consider multiple performance measures and if, they would need many iterations to adapt to different performance prioritizations.

Solution inference of already stored solutions has the potential of using multiple performance criteria while yielding the possibility to quickly derive a potential solution for any currently changed context. Nevertheless, using pre-defined behavior classes has a lack of flexibility, i.e. huge effort in order to apply the approach on different systems and no adaptation capability towards any short- or long-term changes of the robot. CBR is a very general approach that can continuously create solutions for incoming problems while improving the underlying knowledge base throughout the life-time of the robot. If proper methods for case retrieval, adaptation, evaluation, and memory storage can be found, an application might be successful to map from context to behavior. Also, the approach of using a behavior-performance map is a promising way to go unless a dimensionality reduction from the original input space can be avoided. Extending it for continuous context changes and considering a variable performance prioritization can make this approach an elegant way to fulfill the goals of this thesis.

To conclude, in order to realize a transferable, online, context-depending adaptation strategy, case-based and model-based approaches are most promising. The former correspond more or less to lazy learning approaches, which avoid a complete function approximation prior to runtime, i.e they have time-efficient update procedures but need to sustain most of the training data. Considering only the data around the query point during execution may be efficient but also has no extrapolating capability. In contrast, eager model-based approaches compile their knowledge for the whole data space prior to runtime, which requires larger training effort but maintains an efficient prediction step and does not require to keep the entire training data. In fact, a question that needs to be answered is, whether a lazy or eager learning technique, or a combination of both is the best choice to generate working solutions from past experiences while supporting life-long learning. Consequently, the analysis of both ways and multiple target applications should be considered.

Chapter 3

Target Application: Multi-Legged Locomotion

The developed experience-based behavior adaptation for kinematically complex robots is a holistic approach to autonomously adapt a robot's behavior for varying context changes in terms of varying desired action targets, performance prioritizations, environmental conditions, and state of the robot itself. The challenge to correctly parameterize a motion control to optimize the action execution is faced in many robotic applications, e.g. to specify the type of motion planner and configure it according to the difficulty to reach a target pose or to determine prioritization constraints of a whole body control (WBC). The behavior adaptation of the motion control for walking robots is also an ideal target application to test its applicability. Multi-legged walking machines are kinematically complex robots that feature many actuated DOF and are equipped with multi-modal sensors to sense the state of the robot and the environment. A sophisticated motion control is needed that turns all sensor information to new joint targets to follow given motion commands while preserving stability. A high degree of mobility can be reached with walking systems, if the utilized motion control exploits the full potential of the locomotor system. When traversing real outdoor environments, changing terrain types can be encountered that require different walking gaits for a successful traversal. Consequently, quick parameter adaptations are needed and due to the kinematical complexity of the robots, a diversity of movement patterns can be generated. Thus, it is a challenging and interesting application to apply autonomous behavior adaptation.

The first section of this chapter further introduces the reader into the field of locomtion by providing an overview of utilized control approaches. Then, the exemplary target systems that are used within this thesis are described. For every application where autonomous behavior adaptation should be applied, the following questions need to be answered:

- What are the behavior parameters b that should autonomously be adapted?
- What is the action context a and, simultaneously, what are the action and meta performance features p?
- What state context z influences the performance of a behavior and how can it be estimated?

Consequently, the third section describes in detail the control approach to understand the parameters that generate the overall walking behavior. A unifying motion control approach is presented, that allows to control various walking robots, such as SpaceClimber and Charlie. Then, performance features are introduced that characterize executed walking behaviors and allow a quantitative comparison between them. Furthermore, a selection of state context features is provided that help to characterize the robot's situation. Finally, a chapter conclusion is provided to summarize the underlying application-specific parameters and metrics that are utilized by the proposed experience-based behavior adaptation.

3.1 Control Approaches for Legged Locomotion

The challenge to control a walking robot is manifold, i.e. it depends on the robot's morphology that determines the kinematic and dynamic properties, the type of actuators, the available sensors, and the environment that has to be traversed. A robot needs to solve multi-contact constraints for a coherent and expedient motion, at the same time maintaining stability and keeping efficiency in mind. This results in a challenging problem, even more when considering that only incomplete and most likely inaccurate world information is available. In literature, three main approaches can be found in current systems: model-based control, machine learning approaches, and biologically-inspired control.

Model-based approaches try to continuously plan a consistent and stable trajectory for the robot according to the desired motion command and the sensed state of the robot. To avoid instability, trajectories are planned and dynamic stability criteria like centroidal momentum, capture point, or zero moment point (ZMP) are continuously compared with the sensed counterpart on the base of current sensor readings and accurate kinematic and dynamic models of the robot, e.g. [Kuindersma et al., 2016, Herzog et al., 2015, Englsberger et al., 2011, Kajita et al., 2003]. The latest humanoids, e.g. ATLAS[1], Valkyrie [Radford et al., 2015], and Toro [Englsberger et al., 2014] and quadrupeds, such as ANYmal, Spot[2], and Mini Cheetah [Katz et al., 2019], feature torque control

[1] https://www.bostondynamics.com/atlas, accessed on 10/15/2019
[2] https://www.bostondynamics.com/spot, accessed on 10/15/2019

for compliant actuation. The joint targets are usually computed by a WBC framework that sees the control problem as an optimization problem. In WBC, which originates from [Sentis and Khatib, 2006], a multi-objective motion is divided into reactive constraints each describing a sub-goal of the overall action.By solving the resulting optimization problem, torques but also other targets like positions, velocities, or accelerations are generated that satisfy the constraints as good as possible. In order to handle contradicting constraints, a hierarchy is defined, which is solved via null space projection. This guarantees that the highest priority constraints are always executed whereas lower prioritized constraints will only be executed if enough kinematical redundancy is available. This way, kinematically complex robots with many DOF can simultaneously consider several action-related, safety, or physical constraints, e.g. as done in [de Gea Fernández et al., 2017]. The outputs of equally weighted constraints is averaged, potentially failing both constraints. Thus, defining coherent constraints and deriving a clear hierarchy are the main challenges. Their prioritization has a large influence on the overall robot behavior and should be adapted for different contexts.

The complete control or different parts, i.e. trajectory generation and actuator control, can also be learned by machine learning techniques. In [Peng et al., 2017], for instance, the authors propose to learn one ANN with RL to map joint states and incoming motion commands to actuator targets for a simulated bipedal robot. Evolutionary methods are used in [Langosz, 2018] to develop entire control structures for quadrupeds and hexapods. Machine learning approaches are also often used to improve other control approaches, e.g. by learning dynamic models [Nguyen-Tuong and Peters, 2011] or by optimizing control parameters [Römmermann et al., 2008]. The difficulty in utilizing machine learning techniques is the huge amount of training data. Thus, dynamic rigid body simulations are usually employed to generate the training data. The simulation reality gap is a major drawback in this case, resulting in possibly non-performing walking behaviors on the physical system. By producing behaviors that generalize over variable models or by reducing the simulation reality gap, machine learning approaches are getting more and more attractive. In [Hwangbo et al., 2019], for instance, a ANN is trained in simulation and applied on the physical four-legged ANYmal without noticeable performance drop due to a previously trained actuator net that minimizes the gap between simulated and physical system.

Hexapods, such as LAURON V [Rönnau et al., 2014] or HECTOR [Schneider et al., 2011], come with the stability advantage of having at least three legs on the ground during locomotion which creates a large support polygon. Consequently, they are a good choice for exploring unstructured terrain, e.g. for search and rescue applications. Although, they usually have a higher

kinematical complexity in terms of more DOF, less control complexity is needed through the intrinsic stability property. Biologically-inspired control approaches that harmonize with the insect-like design, are commonly used to control hexapods or octopods. Instead of using complex internal models, aspects of decentralized control and reactive modules are utilized. Reflex-chains, for instance, studied from the motion of stick insects [Cruse et al., 1998] among others, derive a walking gait from the consecutive activity of diverse reflexes. Here, the output of one reflex triggers the next which finally generates a periodic movement. Another approach is to use a central coordinated neural oscillator, or CPG, that generates rhythmic patterns without the need of external stimuli. The generated open-loop trajectories are then superposed by reactive behavior modules that modulate the original set values, thus incorporating the sensed state of the robot in order to adapt to the surface. The core idea is that the outputs of many simple behavior producing modules can be merged to generate an emergent overall robot behavior that is more than its single components. Thus, a high degree of mobility with comparably few processing costs can be realized, as shown for example with the Scorpion robot [Spenneberg and Kirchner, 2007]. However, in both aforementioned biologically-inspired concepts, adapting one model or rule will have side effects on the overall robot behavior. By adding new ones, the complexity can quickly grow. Consequently, the influence of adaptations can hardly be foreseen beforehand.

3.2 Exemplary Robotic Systems Used in this Thesis

Within this thesis, two kinematically different systems are selected in order to evaluate the experience-based behavior adaptation: SpaceClimber and Charlie. SpaceClimber [Bartsch et al., 2012] (Figure 3.1) is a six-legged walking robot that uses a CPG-based control approach to traverse unstructured terrain including steep slopes, sandy surfaces, and rough terrain. Four joints per leg allow SpaceClimber to walk in diverse gaits with different stability and energy characteristics. To distribute the control effort, each joint consists of a motor and a Field Programmable Gate Array, as well as position, velocity, and current sensors for local actuator control. Every leg is attached to the body via a force-torque sensor to be able to detect ground contact. Inside the body, the main processing units, power electronics, and communication devices are integrated. In addition, an inertia measurement unit (IMU) for measuring the orientation and acceleration is placed in the front part. The pose information are used together with the measured positions of the limbs with ground contact to compute a contact-based odometry [Schwendner and Joyeux, 2011]. SpaceClimber's sensor head houses a camera, a horizontal two-dimensional laser scanner, and a servo. The latter continuously sweeps the laser scanner to generate three-dimensional (3D)

Figure 3.1: SpaceClimber in an artificial Mars environment

point clouds of the environment. Thus, the generation of maps is supported that can be used to estimate the state context and finally to trigger a behavior adaption for improved autonomy and mobility.

Charlie [Kuehn et al., 2017] (Figure 3.2) is an ape-like quadruped that features 39 active, locally position controlled DOF to generate multiple locomotion behaviors for even and unstructured terrain. Charlie has a force-torque sensor mounted on each wrist and ankle, respectively, to measure ground contact. In addition, Charlie has two unique features in comparison to other quadrupeds found in literature:

- an active artificial spine [Kuehn et al., 2011], which can be used to support walking
- active ankle joints [Fondahl et al., 2012], that improve four-legged walking and, even more important, allow Charlie to stand and walk on his rear legs.

Furthermore, Charlie contains an IMU in its chest to compute a contact-based odometry to provide position and velocity estimations. All proprioceptive sensor data is incorporated by the CPG-based reactive control approach. The exteroceptive sensor information from a time of flight camera and a stereo camera in Charlie's head are used to generate 3D point clouds, which also allow simultaneous localization and mapping (SLAM) [Dettmann et al., 2018a].

Quadrupeds can theoretically walk in different gaits, e.g. static walking (i.e. at least three feet with simultaneous ground contact), dynamic walking (i.e. at least on leg with ground contact), and dynamic running or jumping including full flight

Figure 3.2: The ape-like robot Charlie

phases. Thereby, each gait has different performances in terms of stability, energy efficiency, and maximum velocity which also vary with the type of surface or gravity as analyzed in [Kolvenbach et al., 2018]. For Charlie, who features stiff position-controlled joints, static and quasi-static walking gaits are most suited. Nevertheless, the intelligent gait selection based on experienced gait performances for different walking speeds is an interesting use case for the context-sensitive behavior adaptation that can be evaluated with Charlie.

3.3 Modular Central Pattern Generator for Multi-Legged Walking Robots

SpaceClimber and Charlie are selected as example systems to analyze the autonomous adaptation of a locomotion control. Both are controlled via a similar behavior-based control approach [Kühn, 2016, Bartsch, 2014]. A CPG generates a clock, which triggers consecutive leg movements. The resulting open-loop Cartesian foot trajectories are then superposed by reactive modules to adapt to the surface. Finally, inverse kinematics are used to generate the target values for the position-controlled joints.

Unfortunately, both had different implementations, used different names for equal parameters, and vice-versa different interpretations for equally labeled parameters. Furthermore, both implementations incorporated robot-specific knowledge that limited them from being utilized for other robots. Their stand-alone software was also not meant to be integrated within larger robot operating systems, such as ROS or ROCK. Consequently, a uniform implementation was desired, that combines the advantages of both former implementations. A modular, generic implementation would allow its usage on multiple walking machines and provide a uniform interface to robotic frameworks. In addition, when using a common code base, the improvement of a module for one robot will also bring benefits for others.

All of these reasons led to the development of the modular, generic CPG approach that generates walking trajectories regardless of the exact morphology of the target robot. Even though the experience-based adaptation concept is abstracting this control layer as a black box, for the understanding of the experimental evaluation of this thesis, it is important to know the general concept and the influence of specific behavior parameters. Thus, the developed control approach is described in the following in more detail. The behavior parameters are described and the subset being adapted within the experimental evaluation are highlighted in `teletype`.

3.3.1 General Concept

The overall concept follows key principles of a behavior-based control architecture, where the overall robot behavior is generated by the interaction of multiple processing units, also called behavior modules. Two basic guidelines are followed to generate a generic and reusable control architecture. First, the overall control is split into general-applicable behavior modules, which are required for every system, and robot-specific behavior modules that extend the general architecture and exploit the capability of the target system. Second, behavior modules are grouped to larger behavior modules to implement a hierarchical structure that efficiently reuses functionalities.

Every behavior module provides a specific functionality contributing to the overall robot behavior. On the top layer, in- and outputs of the overall control as well as the main behavior modules are defined, e.g. the *CPG* behavior module to derive gait-dependent step cycles for the limbs and for the body, a *posture controller*, several *limb controllers* for every motion-supporting arm or limb, a *head controller*, as well as data processing nodes, which provide module-specific information of the current robot state (Figure 3.3). The architecture is a mixture of open-loop trajectory-generating modules and trajectory-adapting reactive modules to adapt to the environment.

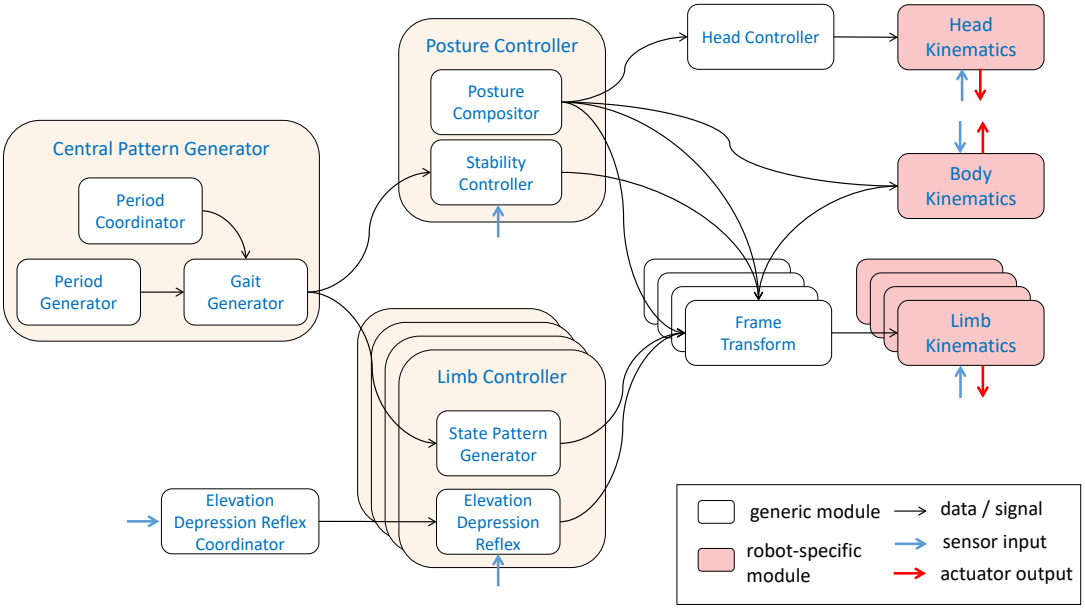


Figure 3.3: Abstract overview of the modular central pattern generator for a robot with four limbs, thus having four instances of limb controllers, frame transforms, and limb kinematics (data processing nodes are left out for clarity)

To abstract from the robot's morphology, all behavior parameters are defined in an imaginary frame, i.e. a coordinate system that is not fixed to any body part, the so called $LocoFrame$ (Figure 3.4). It is parallel to the ideal ground and is the representative frame for any navigation tasks. Within this frame, posture and limb controllers generate desired end effector targets that are superposed to generate the desired end effector poses ${}^{Loco}_{EE_l}\boldsymbol{T}$, where l denotes the respective limb name, e.g. for a hexapod FrontLeft (FL), FrontRight (FR), MiddleLeft (ML), MiddleRight (MR), RearLeft (RL), and RearRight (RR). They are converted by the respective *frame transform* modules into the desired limb base coordinate systems in two steps. First, every end effector pose is transformed from $LocoFrame$ to the robot coordinate system ($RobotFrame$), which is fixed to the root of the kinematical robot model

$$ {}^{Robot}_{EE_l}\boldsymbol{T} = {}^{Loco}_{Robot}\boldsymbol{T}^{-1}\, {}^{Loco}_{EE_l}\boldsymbol{T}, \tag{3.1} $$

where ${}^{Loco}_{Robot}\boldsymbol{T}$ denotes the transformation from $LocoFrame$ to $RobotFrame$ based on the outputs of the posture controller.

The Cartesian target poses can now be transformed into joint targets, e.g. by using inverse kinematics or a model-based WBC approach. For the position-controlled SpaceClimber and Charlie, the inverse kinematics approach is utilized. Therefore, the highly-complex overall robot kinematics is split into body and respective limb kinematic chains, which facilitates the problem of solving forward and inverse kine-

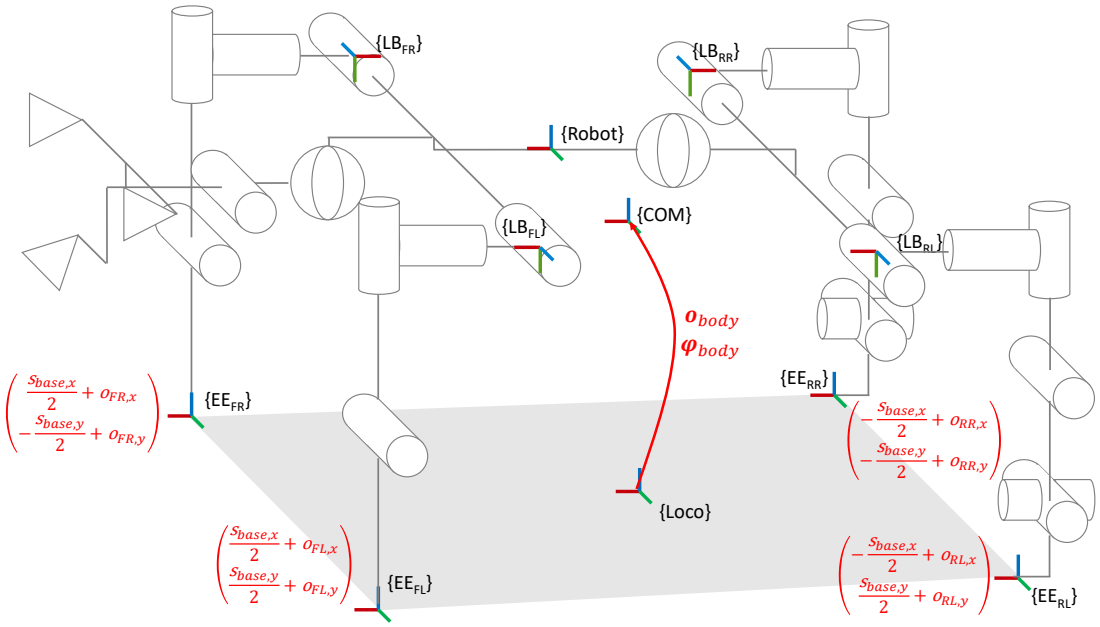


Figure 3.4: Coordinate systems and posture parameters on the example of Charlie

matics. Thus, the end effector target poses in the $RobotFrame$ ${}^{Robot}_{EE_l}\boldsymbol{T}$ are transformed into the respective limb base coordinate systems (LB_lFrame) via

$$ {}^{LB_l}_{EE_l}\boldsymbol{T} = {}^{Robot}_{LB_l}\boldsymbol{T}^{-1}\, {}^{Robot}_{EE_l}\boldsymbol{T}, \tag{3.2} $$

where ${}^{Robot}_{LB_l}\boldsymbol{T}$ is computed from the forward kinematics of the body joints.

Finally, the limb inverse kinematics are used to compute the joint targets from the target poses ${}^{LB_l}_{EE_l}\boldsymbol{T}$. For both target systems, the inverse kinematics of arms and legs are efficiently solved in closed form. This way of abstracting the behavior modules from the kinematical morphology (except for the robot-specific kinematics) facilitates the transferability. Only their configuration needs to be adapted to fit the robot's range of motion and dynamic properties.

Biologically inspired Graph-Based Language (BAGEL) [Langosz, 2018, Langosz et al., 2013] was chosen for the implementation of the modular CPG control. It is an open-source software[3]. BAGEL has the advantages, among others, of supporting hierarchical architectures as well as arbitration strategies to merge multiple signals on one input, both facilitating the implementation of a behavior-based architectures. In the following, the key behavior modules are described to learn about the influence of their configuration parameters.

[3] https://github.com/dfki-ric/bagel_wiki/wiki, accessed on 12/03/2019

3.3.2 Central Pattern Generator

The CPG is responsible for the cyclic walking gait generation. It generates a periodic clock for every limb, so that each limb controller can be timed whether to support the body or to swing to the next support point. The module consists of a *period generator*, a *period coordinator*, and a *gait generator*.

The *period generator* derives a periodic, normalized, saw-like curve from zero to one representing the internal robot clock. Thus, the internal time step increment t_{inc} depends on the desired step cycle time t_{cycle}(`t_cycle`) and the update period of the control t_{period}:

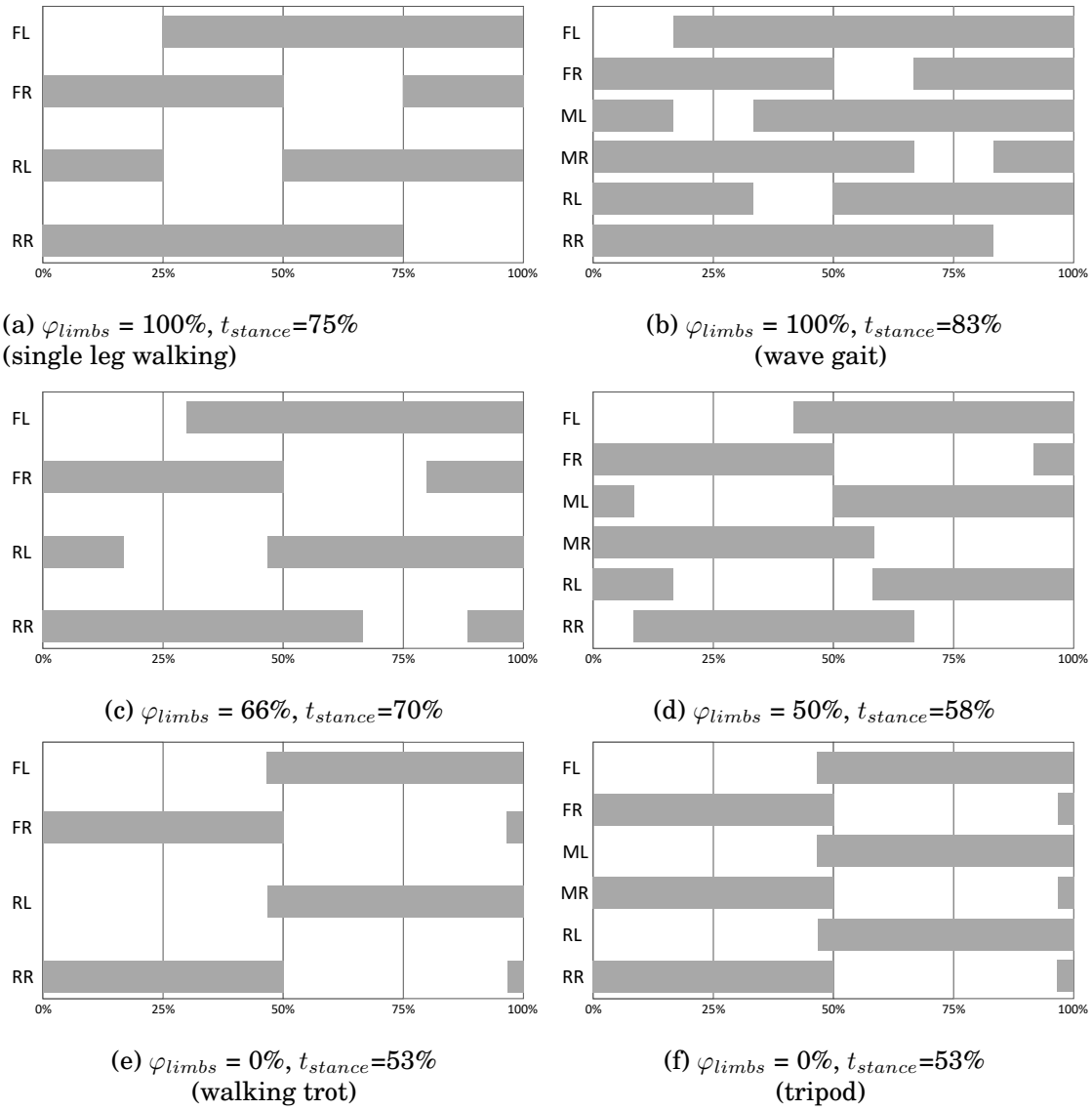
$$t_{inc} = \frac{t_{period}}{t_{cycle}}. \tag{3.3}$$

The *period coordinator* starts and stops the internal clock depending whether a movement is desired or not. If the robot is requested to stop, the period coordinator forces the step cycle to continue, i.e. the period counter further increases, until all legs are placed on the ground to hold in a stable stance. During this stopping phase, no further leg is allowed to start the lifting movement. The period counter finally stops when all legs are in stance. In addition, the period counter is reset to the point when the stop signal was emitted in the first place. Thus, the walking cycle continues with the leg originally scheduled for liftoff, if a robot movement is desired again. This way, no swing phase of a leg is missed and a prolonged stance phase is avoided, which could lead to target positions that violate kinematical limits.

The *gait generator* defines at what time each limb starts with its movement, i.e. lifting the leg. Therefore, a limb-dependent clock is generated from shifting the overall robot clock about pre-defined time spans. Thus, the same leg controller can be implemented to align the leg's swing t_{swing} and stance phase t_{stance}. The current implementation supports timings for two-legged, four-legged, five-legged and six-legged walking gaits but can be extended to any other number of limbs N_{limbs}. A phase shift φ_{limbs} (`phase_shift`) can be set to influence the time between consecutive legs. A full phase shift ($\varphi_{limbs}= 1$) evenly distributes the start of each leg movement, whereas no phase shift ($\varphi_{limbs}= 0$) leads to lifting diagonal legs simultaneously[4]. Figure 3.5 shows example gaits in form of Hildebrand diagrams [Hildebrand, 1965]. Depending on the chosen combination of φ_{limbs} and t_{stance} (also called duty factor), different symmetric gaits are realized: stable but slow gaits (Figure 3.5a and 3.5b), more efficient, classical walking gaits close to biological counterparts as trot (Figure 3.5e) and tripod (Figure 3.5f), but also hybrid forms (Figure 3.5c and 3.5d) that might be optimal for robotic walking machines to suit specific stability characteristics. The parame-

[4]For the implemented five-legged gait, this refers only to the middle and rear legs

ters have to be chosen carefully to create statically or quasi statically stable walking patterns. Dynamic patterns with full flight phases can also be generated but are not suitable for the target systems of this work. Thus, dynamic gaits are not considered any further.



(a) φ_{limbs} = 100%, t_{stance}=75% (single leg walking)

(b) φ_{limbs} = 100%, t_{stance}=83% (wave gait)

(c) φ_{limbs} = 66%, t_{stance}=70%

(d) φ_{limbs} = 50%, t_{stance}=58%

(e) φ_{limbs} = 0%, t_{stance}=53% (walking trot)

(f) φ_{limbs} = 0%, t_{stance}=53% (tripod)

Figure 3.5: Hildebrand-style diagrams for a selection of 4-legged (left) and 6-legged (right) walking gaits, the grey bars indicating the stance phase

3.3.3 Posture Controller

The posture controller consists of a *posture compositor* and a *stability controller*. The *posture compositor* allows to define the basic posture, thus defines the target positions of each end effector without considering movement. To set the distance between the first and last pair of legs, as well as the lateral difference between left and right legs, a basic stance $\boldsymbol{s}_{base}$ (`step_base_x`, `step_base_y`) must be chosen (Figure 3.4). It is also possible to set an individual offset along every axis for each limb with $\boldsymbol{o}_l$. This way, end effector positions can be modulated to meet the range of motion of potentially different arm and leg kinematics or to provide an interface for other behavior modules, e.g. for a foot step planner that adapts the next target spot based on map information to avoid stepping on an slippy edge.

In the following, only the computation of end effector positions is described, since many multi-legged robots feature point contact feet which do not need a distinct orientation. In case rigid multi-contact feet are utilized and sufficient DOF are available, a desired orientation can be provided, e.g. derived from map information. If not, a flat surface is assumed, i.e. surface and $LocoFrame$ orientation are equal.

A desired translation $\boldsymbol{o}_{body}$ (`body_shift_x`, `body_shift_y`, `body_shift_z`) and orientation $\boldsymbol{\varphi}_{body}$ (`body_rot_pitch`, `body_rot_roll`, `body_rot_yaw`) can be set to define the target pose ${}^{Loco}_{COM}\boldsymbol{T}$ of the center of mass (COM). The target pose of the robot root node is then derived with

$$ {}^{Loco}_{Robot}\boldsymbol{T} = {}^{Loco}_{COM}\boldsymbol{T}\,{}^{Robot}_{COM}\boldsymbol{T}^{-1}. \tag{3.4} $$

For some robots the COM can be assumed to be static. Otherwise, the measured COM in the robot frame ${}^{Robot}_{COM}\boldsymbol{T}$ can be computed with

$$ {}^{Robot}_{COM}\boldsymbol{T} = \begin{pmatrix} \boldsymbol{I} & \frac{\sum m_i \boldsymbol{r}_i}{\sum m_i} \\ \boldsymbol{0} & 1 \end{pmatrix} \tag{3.5} $$

by using the current joint angle measurements and the kinematic model that provides the masses m_i and vectors $\boldsymbol{r_i}$ to every modeled body part i.

The chosen body pose and basic stance have an effect on the robot's range of motion and stability. For the target systems for instance, a lower and wider posture improves stability on the costs of energy efficiency due to unfortunate levers that increase the power required to maintain the body height. In contrast, a higher narrow posture usually reduces required joint speeds but decreases the robot's stability.

In general, the overall approach is designed in the way that every parameter is allowed to be changed during runtime. Thus, if any parameter change must be timed, the corresponding behavior module takes care about it itself. Here, any parameter

adaptation that influences the basic stance is only applied when the corresponding leg is in the air to avoid shear forces as well as tensions between the limbs.

To maintain a stable stance for static walking, the COM projected along the gravity vector $\boldsymbol{g}$ must lay within the support polygon of the robot, which is the convex hull of the end effector points that are in contact with the environment. Because this area continuously changes during the step cycle through lifting and setting a leg, the COM needs to be continuously shifted. The implemented *stability controller* generates a periodic COM trajectory that is synchronized with the chosen gait. It connects the current and next center of support polygon with a spline interpolator to a smooth COM trajectory which is used to set offsets for $\boldsymbol{o}_{body}$. Since targeting the center of support polygon is most secure but not always necessary to maintain stability, a scaling factor f_{com_cpg} (`com_cpg_scale`) is introduced to decrease the required body motion. When no movement is desired, the COM reference sticks to the current center of support polygon. When a new movement is initiated, the stability controller starts a quarter step cycle earlier than the limbs to relieve the upcoming leg before lifting it. Controlling the error between actual COM projection and the target point automatically introduces a body shift when walking in inclines that increases stability and avoids tip overs. To incorporate dynamic effects, the projected COM can also be replaced by the ZMP [Vukobratović and Borovac, 2004] or the center of pressure.

3.3.4 Limb Controller

Every limb controller consists of a *state pattern generator* and an *elevation and depression reflex (EDR)* module. The state pattern generator generates a Cartesian end effector trajectory which superposes the posture-derived quasi static target position. It is a cyclic trajectory which is based on the current progress of the current limb-specific step cycle. Every step cycle starts with a swing phase for a given period of time and finishes with the end of the consecutive stance phase. As depicted in Figure 3.6, the swing phase is divided into a lift t_{lift} (`t_lift`), shift t_{shift} (`t_shift`), and down phase t_{down} (`t_down`), each defined by a portion of the overall step cycle

$$1 = t_{stance} + t_{swing} = t_{stance} + t_{lift} + t_{shift} + t_{down}. \tag{3.6}$$

In order to realize a statically stable walking pattern that has no flight phases, the swing time must not exceed the maximum air time $\hat{t}_{swing}$ that is defined as

$$\hat{t}_{swing} = 1 - \check{t}_{stance} \tag{3.7}$$

$$= \frac{1}{2} - \varphi_{limbs} \cdot \left(\frac{1}{2} - \frac{1}{N_{limbs}} \right), \tag{3.8}$$

with $\varphi_{limbs} \in \mathbb{R}[0, 1]$ and $N_{limbs} \in 2\mathbb{N}_{>0}$.

A leg can only enter the lift phase when movement is desired, i.e. if stop is commanded but the period counter still increases to end motions of other limbs, the leg will not be lifted although a new step cycle may have started. During lift phase, the end effector is moved to the target step height h_{step} (`step_length_z`). Then, during shift phase, the end effector is shifted to the front or side according to the chosen step lengths in longitudinal and lateral direction $\boldsymbol{l}_{step}$ (`step_length_x`, `step_length_y`). Afterwards, the leg is lowered to the ground during down phase. During the entire stance phase, the end effector is moved on the ground equally to all other feet with ground contact, which determines the theoretical overall robot speed $\boldsymbol{v}_{theo}$

$$\boldsymbol{v}_{theo} = \frac{\boldsymbol{l}_{step}}{(1 - t_{shift})t_{cycle}}. \tag{3.9}$$

During lift and down phase the leg is moved against the direction of movement to compensate the actual robot speed. Consequently, a vertical lift with respect to the environment is realized, which is beneficial while climbing over obstacles. Since the generated trajectory requires much movement range, one can scale the lift and down phase as well as their target values with f_{ps} (`phase_shortage`). By increasing this shortening, a prolonged shift phase is created, which requires less movement range. In order to maintain the same step height h_{step}, a spline is generated from the end of the lift phase over the peak in the middle of the step to the start of the down phase, which is indicated by the red curve in Figure 3.6. Thus, larger steps are possible or less joint velocities are needed, well suited for walking on plain ground.

Each phase has its target point which is reached within its given target time using either a linear or spline interpolator. The linear interpolator is used for the stance phase as well as for the horizontal components in the lift and down phase to avoid upcoming shear forces due to different speeds. The spline interpolator is used for the vertical motions and within the shift phase to generate smooth, acceleration-limited curves.

Above descriptions hold for longitudinal and lateral movements. To generate turning motions, a rotation angle ψ (`turn_rate`) can be defined that specifies how much the end effector is rotated around the $LocoFrame$ at the end of the lift phase. During shift phase, the end effector is continuously rotated towards the inverted angle. During down, stance, and lift phase, the interpolation towards the target value causes the turning of the robot. Thus, a coherent turning motion is realized by all stance legs which can superpose the translational movement.

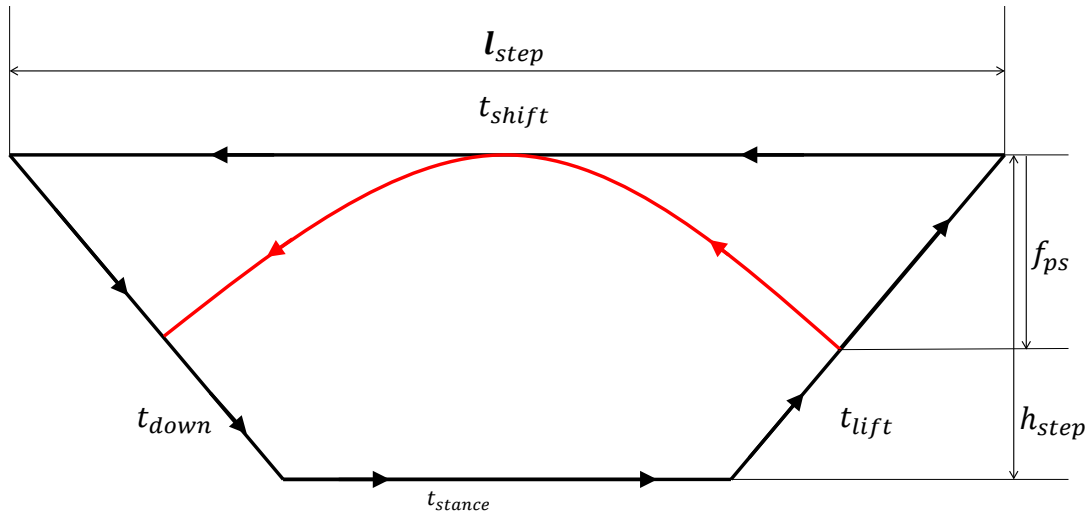


Figure 3.6: Foot trajectory and its configuration parameters. The red curve shows the generated trajectory for $f_{ps}= 0.6$

The state pattern generator creates an open-loop trajectory, i.e. external influences are not considered. When traversing unstructured terrain, hazardous situations may occur, if the robot is not capable of adapting to the environment. One problem originates from obstacles within the path of a leg which most likely happens during down phase. If the corresponding leg was stretched completely as intended by the open-loop trajectory, the entire robot would tilt or even tumble over. Another problem is the absence of a supporting structure during stance phase, e.g. stepping in a hole, which can end in the same result. For both situations, the EDR module adapts the end effector pose, so that the overall robot adapts to the terrain. Therefore, each leg needs to be able to detect ground contact. The selected target systems feature force-torque sensors to provide this information. If the leg detects ground contact in the down phase, the EDR writes inverted vertical offsets with respect to the original trajectory. Thus, the height of the leg will be maintained, i.e. the leg remains crouched on top of the obstacle. If the contact is lost, a control loop stretches the leg until contact is established again. This awareness remains for the entire down and stance phase, thus also compensating for holes as long as no kinematic boundaries are violated. During lift and shift phase, the written offset is faded out to approach the original trajectory again.

There is the possibility, that over time all legs are stretched or crouched which results in a different body height. Thus, an *EDR coordinator* module supervises all written offsets of the stance legs and compensates them if needed to avoid body height fluctuations. In addition, situations may occur when the triggering of a reflex must be inhibited, e.g. when the body is tilted to the front, the front legs will automatically touch the surface earlier whereas the rear legs will for sure not hit the ground in the stance phase. The unsupervised EDR can then even worsen the situation. Thus, the EDR coordinator inhibits crouching and stretching depending on the measured inclination error φ_{err} in longitudinal and lateral direction

$$\varphi_{err} = \varphi_{ref} - \varphi_{robot} = \varphi_{slope} + \varphi_{body} - \varphi_{robot}, \tag{3.10}$$

where φ_{robot} is the current measured inclination of the robot and φ_{ref} the reference inclination that depends on the desired robot pose φ_{body} and the actual slope of the surface φ_{slope}. For instance, if the y-component of φ_{err} is positive and exceeds a threshold, e.g. by 2° that indicates a slight tilt to the back, stretching of the fore limbs is inhibited as well as crouching of the rear limbs. The opposite yields for a negative φ_{err}. Accordingly, the left and right limbs are influenced by the x-component of φ_{err}.

3.3.5 Further Robot-Specific Behavior Modules

The behavior modules above generate walking behaviors to traverse unstructured and inclined terrain. In order to apply the motion control on different robots, only behavior module parameters need to be adjusted to meet the movement capabilities of the corresponding robot. In addition, the required current end effector poses and desired Cartesian set points need to be transformed from or to the joint level by the robot-specific forward and inverse kinematics, respectively.

Nevertheless, the robot-specific features, e.g. Charlie's active spine or ankles or Mantis' additional body joints to support four-legged and six-legged walking, require additional behavior modules that can be integrated in the hierarchical control scheme. Since the corresponding control parameters are not adapted within this work, further details are not part of this thesis. However, further information of the presented modular central pattern generator and its applications can be found in [Dettmann et al., 2020, Kuehn et al., 2020, Dettmann et al., 2018b, Kuehn et al., 2018, Kuehn et al., 2017, Bartsch et al., 2016, Kühn, 2016, Bartsch, 2014].

3.4 Evaluation of Walking Behaviors

The developed modular CPG approach allows to generate diverse walking patterns by tuning specific control parameters. The following general performance features are selected for evaluating the resulting locomotion behaviors.

First, it is important to evaluate how closely the robot executes the desired action. In this application, the planning and navigation layer usually generates a path which is followed by using a trajectory follower. The trajectory follower continuously compares desired path with current robot pose and generates motion commands to compensate the error. These commanded longitudinal, lateral, rotational velocities mainly define the action that has to be realized by the motion control. Thus, the robot has to evaluate its own motion by also measuring the longitudinal velocity (`velocity_x`), lateral velocity (`velocity_y`), as well as angular velocity (`velocity_rot`) around its geometrical center defining the action performance features

$$\boldsymbol{p}_{vel} = \begin{pmatrix} \texttt{velocity\_x} \\ \texttt{velocity\_y} \\ \texttt{velocity\_rot} \end{pmatrix}. \tag{3.11}$$

These features need to be known very precisely for other algorithms as well, e.g, map building. Since many real world applications do not support the usage of accurate external reference systems, the velocity needs to be estimated by the robot itself. The target robots in this thesis use a contact point odometry that is based on [Schwendner and Joyeux, 2011]. In principle, it averages the changes of foot positions during ground contact and fuses them with the measured orientation changes by the IMU to compute the overall robot pose difference.

Besides motion commands, in some applications the operator needs to influence the robot's posture as well, e.g., to fit through door frames or to enter flat confined spaces. Thus, behaviors need to be distinguishable in terms of dimensional aspects, which is done in this work by estimating the robot's lateral size (`body_width`) and its height (`body_height`). The performance feature `body width` is simplified as the largest lateral distance between any end effector of the left and right side because this is usually the case for the selected target systems. The performance feature `body_height` is represented by the inverted average of all vertical end effector positions with respect to the robot's coordinate system. This ignores the robot's rotation and the exact morphology but is sufficiently precise to have an influence on the body height of generated walking behaviors within the autonomous behavior adaptation.

Summarizing, the posture is characterized with

$$\boldsymbol{p}_{posture} = \begin{pmatrix} \texttt{body\_width} \\ \texttt{body\_height} \end{pmatrix} = \begin{pmatrix} max(\boldsymbol{t}_{l,y}) - min(\boldsymbol{t}_{l,y}) \\ -\bar{\boldsymbol{t}}_{l,z} \end{pmatrix}, \tag{3.12}$$

where $\boldsymbol{t}_{l,y}$ and $\boldsymbol{t}_{l,z}$ are the vectors of the respective y- and z-component of the translational part of every ${}^{Robot}_{EE_l}\boldsymbol{T}$ which is the transformation from $RobotFrame$ to corresponding EE_lFrame based on measured joint angles and the forward kinematics.

The above mentioned action performance features will indicate a high behavior accuracy if they are close to the commanded values. Since kinematically complex robots have numerous possibilities to perform the same action in various ways, additional criteria to compare them are required. For these meta performance features, every describing attribute can be chosen, that might be of interest for the target application. In [Cully et al., 2015] for instance, the stance time of each leg is a metric. This is a reasonable choice for the respective application, in which limbs are damaged on purpose and the robot should adapt to a behavior which does not use the broken leg, thus has minimal stance time. In this work, locomotion behaviors are additionally characterized by selected stability and efficiency features.

A stability feature should characterize how far the robot is away from tipping over. For statically stable walking robots, the static stability margin (`ssm`) [McGhee and Iswandhi, 1979] is a common metric. It measures the stability as a minimal distance between the COM projected along the gravity vector $\boldsymbol{g}$ and the edges of the support polygon, which is the convex hull formed by the robot's contact points (Figure 3.7). This metric was chosen as performance feature for its simplicity and low computational costs. In order to account for dynamic effects, the force-angle stability margin [Papadopoulos and Rey, 2000], here referred to as dynamic stability angle (`dsa`), is selected as second stability feature. It defines the stability as minimal angle between the acceleration vector $\boldsymbol{a}_{robot}$ which projects the COM to the ZMP and the vector from the robot's COM to the closest edge of the support polygon. Thus, the body height is taken into consideration as well as the robot's acceleration , which is in principle more precise but also more noise-affected. Summarizing, the stability performance features are

$$\boldsymbol{p}_{stability} = \begin{pmatrix} \texttt{ssm} \\ \texttt{dsa} \end{pmatrix}, \tag{3.13}$$

where higher values indicate higher stability.

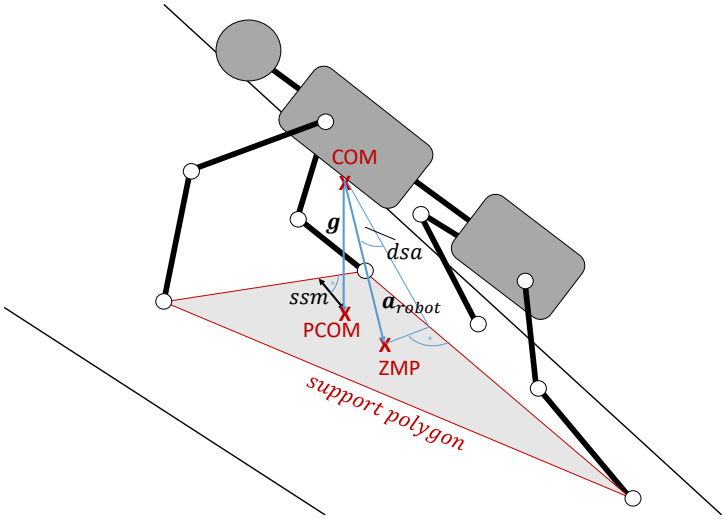


Figure 3.7: Sketch of stability features *ssm* and *dsa*

In addition to the stability features, three performance metrics are selected to characterize the efficiency of a walking behavior

$$\boldsymbol{p}_{efficiency} = \begin{pmatrix} \texttt{power} \\ \texttt{epd} \\ \texttt{vibration} \end{pmatrix}. \tag{3.14}$$

The overall power (`power`) needed by all joints as well as all sensor and processing units is one feature. The metric also indirectly incorporates self-collisions since colliding body parts during actuation drain energy which is not used for locomotion. Because a high energy consumption is not avoidable in some situations, e.g. when walking fast or in steep inclines, the energy per distance (`epd`) is used as second criterion. In order to maintain a continuously measured metric, the current power consumption is multiplied with the processing period t_{period} and divided by the average arc length $\triangle\overline{s}_{arc}$

$$\texttt{epd} = \frac{p_{power} t_{period}}{\triangle\overline{s}_{arc}}. \tag{3.15}$$

The average arc length $\triangle\overline{s}_{arc}$ represents how much every foot in contact with the ground moved with respect to the $RobotFrame$ during one time step to achieve the measured translation and orientation changes. With using this measure as pose change of the robot, rotations on the spot are also considered and do not cause an infinite `epd`. At last, the undesired body vibration (`vibration`) is approximated by the Euclidean distance between sensed and desired rotational velocity

$$\texttt{vibration} = ||\dot{\boldsymbol{\varphi}}_{robot} - \dot{\boldsymbol{\varphi}}_{body}||^2. \tag{3.16}$$

This metric provides information on how smoothly the walking gait traverses the surface. Non-matching parameter combinations would cause early or late touchdowns, thus increasing the undesired `vibration`. Low values for all chosen efficiency features characterize high efficiency.

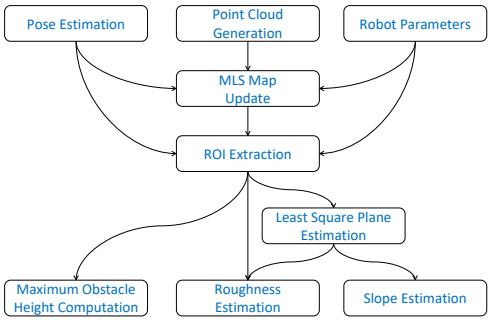
3.5 State Context Estimation

As defined in Section 1.2, the state context z is a vector of features which characterize the environment or the internal state. There is no hard specification which context features to take, since it highly depends on the application as well as the sensor and processing capability to compute them. Nevertheless, each of them should have a measurable influence on the performance of an executed behavior.

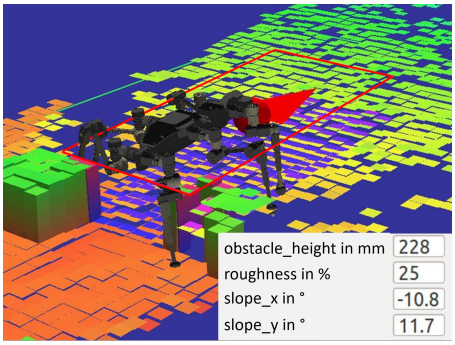
In the use case of a walking robot traversing unstructured terrain, characterizing the type of environment with few features is important. Soil properties can have a large influence but are disregarded within the scope of this thesis because developing an efficient and reliable implementation is still a research topic itself. Consequently, only rigid surface structures are considered. Four state context features are selected that can be derived from map information, the longitudinal and lateral slope (`slope_x` and `slope_y`), the peak obstacle height (`obstacle_height`), and the terrain roughness (`roughness`). Thus, the state context z for the target application is given by

$$z = \begin{pmatrix} \texttt{slope\_x} \\ \texttt{slope\_y} \\ \texttt{obstacle\_height} \\ \texttt{roughness} \end{pmatrix}. \tag{3.17}$$

In order to generate the selected state context features, a 3D map is built from continuous synchronized point cloud data and pose information as visualized in Figure 3.8a. In this work, the environment is represented in form of a multi-level surface map [Triebel et al., 2006], which has the possibility to handle overhanging and vertical objects, e.g., the representation of confined spaces. Therefore, robot-specific configuration parameters are needed: The size of a single patch, that represents the mean of all point cloud data in this area, has to be set to match the foot size of the robot, e.g. 8 cm x 8 cm for the selected target systems Charlie and SpaceClimber. By considering the height of the robot and its maximum step height, the traversable surface is built from the environment map, which excludes all patches unaccessible to the robot.



(a) Flow diagram of the context generation module



(b) Visualization of generated map, region of interest (violett area beneath red rectangle), and estimated state context features

Figure 3.8: Context generation

The relevant part of the environment is extracted, the so-called region of interest (ROI), because not the whole map should have an influence on the current locomotion behavior. Two possible ROI are useful: For linking a context to an evaluation, the area beneath the robot during the evaluation process is the important sector. In order to react on upcoming surface characteristics, the region in direction of movement during the next step cycle should additionally be considered (Figure 3.8b).

Even though the 3D map shows many potential obstacles, only one scalar representing the obstacle with highest influence on the walking pattern is considered. This is reasonable since a reactive control approach is supposed to handle obstacles in principle, but fine tuning for expected differences improves the locomotion performance. Thus, the state context feature `obstacle_height` represents the highest height difference between cells with respect to the robot. Therefore, the ROI is transformed to the robot's coordinate system making the step hazard independent from the ground inclination. Furthermore, the sign of the height difference is valuable information. So, different robot behaviors can be selected for going up or down a step, e.g. a robot may increase its body height for climbing stairs, whereas while going down it may keep his body lower to the ground. Thereby, positive steps should have more influence on the locomotion behavior because they can block the intended motion. So, a positive `obstacle_height` above a minimum threshold min_diff is preferred to any negative `obstacle_height`. Algorithm 1 depicts the corresponding algorithm in pseudo code.

To compute the slope of the terrain, a least-square plane fitting algorithm is applied to the ROI [Gu et al., 2008]. Here, the overall inclination value is not sufficient because the single components with respect to the robot frame require different behavior adaptations, i.e. climbing a hill requires different walking patterns than

Algorithm 1 Pseudo code for computing *obstacle height*

```
1: obstacle_height = 0
2: for each patch in ROI do
3:     n_patch = getNeighbor(patch, motion_direction)
4:     diff = getHeight(n_patch) − getHeight(patch)
5:     if |diff| < min_diff then
6:         diff = 0
7:     end if
8:     if diff > 0 then
9:         obstacle_height = max(diff, obstacle_height)
10:    else
11:        if obstacle_height <= 0 then
12:            obstacle_height = min(diff, obstacle_height)
13:        end if
14:    end if
15: end for
```

traversing it laterally. To determine the inclination of the surface relative to the gravity vector, but in the longitudinal and lateral direction of the robot, only the rotation of the robot around the gravity vector is applied onto the ROI. Then, the angles between x and y component of the plane's normal vector and the gravity vector are calculated to compute the value for the terrain slope `slope_x` and `slope_y`.

The state context feature `roughness` is used to distinguish between surfaces with one peak step from completely irregular ones. It is computed by the standard deviation over all cell patches with respect to the least square plane's normal vector. The value is normalized to the maximum possible step height of the robot to obtain a robot-meaningful coefficient.

3.6 Conclusions

Walking systems show impressive mobility and partially autonomous behavior but are still far away from everyday usage. They still operate mostly under supervision and their application requires careful preparation and human intervention to configure or fine-tune their motion control to the application's context as stated in [Johnson et al., 2015]. Consequently, by increasing the context range in which a robot can autonomously behave through context-sensitive behavior adaptation is one step towards reliable utilization of multi-purpose robots within every day live. In particular, hybrid control approaches are used within the locomotion domain to combine the advantages of fast reactive controls with a deliberative planning layer on top. However, they suffer from inconsistency between both layers caused by different forms of representations and update rates. Regardless of the actual implementation

of the reactive execution layer, the deliberative navigation layer cannot respect every detail of the robot or the configuration space of its reactive motion control. In order to avoid a handcrafted middle layer to connect them through the investment of much expert knowledge, the experiences-based behavior adaptation can be applied.

A modular, biologically inspired, reactive control approach is presented that allows both target systems, SpaceClimber and Charlie, to traverse unstructured terrain. By configuring its parameters, walking behavior are generated for a limited context range, e.g. a hill-climbing behavior can be used for certain incline ranges, for others it might work but not optimal, but surely the same behavior cannot be used to traverse a different category of obstacles like random stepping fields. Consequently, many parameters need to be tuned to optimize the locomotion pattern for the current contexts. Usually, the operator needs to manage the numerous configuration options and even simultaneously tune them online to accommodate for new situations. Thus, an autonomous behavior adaptation can improve the overall system's performance by using the robot's full potential and can reduce the danger of misconfigurations minimizing one reason for potential mission failures.

The utilized motion control has behavior parameters to configure the posture, the walking gait, and the reactive modules. 19 of them are utilized for the thesis' experimental evaluation. In addition, four state context features are proposed to characterize the environmental conditions in terms of longitudinal and lateral slope, maximum obstacle height, and surface roughness. To evaluate the performance of a behavior, five action performance (longitudinal, lateral and rotational velocity, body height, and body width) and five meta performance features (static stability margin, dynamic stability angle, power consumption, energy per distance, and undesired vibration) are used. Consequently, with respect to the definitions of Section 1.2, it follows that $B = 19$, $Z = 4$, and $A = 10$ for this application scenario. A complete list of the application-specific inputs and outputs is provided in the Appendix.

Chapter 4

Experience-Based Behavior Adaptation

This chapter describes the concept of the proposed experience-based behavior adaptation. It is a data-driven approach that needs to acquire, manage, and utilize experiences. Therefore, the knowledge base management for multiple actions is described and possibilities to generate them. Then, further implementation details are given to guide a potential user how to extend an existing control with the additionally needed modules. Finally, a manual and automated procedure to generating experiences is explained and the corresponding acquired knowledge bases are described in detail, which build the base for the experimental evaluation. Their utilization for the actual behavior generation is realized with two different approaches that are described and evaluated in the following chapters. This chapter closes with a conclusion on the overall concept and its methodology.

4.1 Concept

Neither pure reactive nor pure planning-based approaches have the capability to fully control state of the art robots. So, hybrid approaches combining the fast response times of reactive controls with predictive capabilities of deliberative controls are usually utilized in complex robotic applications. Unfortunately, this comes with the drawback of a required middle layer, that needs to combine both worlds. The gap between both layers regarding time scales and forms of representations has to be bridged. This often involves application-specific coding and configurating where much expert knowledge of both worlds is mandatory. The implementation usually takes few possible configurations into consideration leading to sub-optimal solutions. Instead, the experience-based motion control interface is proposed (Figure 4.1). It encapsulates the robot-specific reactive motion control, which directly receives and

processes sensor data as well as generates new motor commands. Simultaneously, the experience-based motion control interface provides an application-specific action interface to the deliberative navigation and planning layer by providing additional information and receiving the desired action a with an application-specific performance prioritization w.

The goal to develop a generic interface that is capable of adapting the behavior of any robot for diverse applications leads to the consequence that the underlying control structure needs to be abstracted and seen as a black box. The assumption is, that through adapting control parameters, the behavior of the robot changes, which can be measured by performance metrics that are relevant for the considered application. Thus, a possibility to derive a mapping from action and state context to behavior parameters can be realized through learning from experience. Learning from experience, or also called learning form demonstration, is a machine learning technique that searches a mapping between world and action by learning from examples [Argall et al., 2009]. Every example consists of inputs and corresponding outputs. Within the scope of the thesis, the inputs correspond to the incoming desired action and the sensed state context. The outputs corresponds to the applied behavior, i.e. a configuration of the motion control that actually influences how the robot moves through the world. One key idea is that the robot applies behaviors and is simultaneously capable of self-evaluating the executed behaviors. Through continuously collecting data about its performance in passed situations, the robot can learn from its interaction. Thus, a mapping from continuous context to behavior can be derived and continuously updated. This approach has serval advantages:

- No expert knowledge is required to design a mapping function by hand.
- Because the robot's motion control is abstracted as a black box, the software implementation is independent from target system or application.
- The learned mapping continuously evolves, thus a changing behavior performance over the life time of a robot, e.g. through wearout, is detected and directly included. A proper reaction to this changes may also help to minimize further wearout.
- Solutions for novel situations are generated automatically, which increases the system's flexibility and autonomy.

The component which realizes the behavior adaptation from Equation (1.1) is called *behavior configurator*. It maps desired action and current state context to the supposedly best behavior. The required state context is generated by the *state context estimation* module on the base of preprocessed senor readings. The required

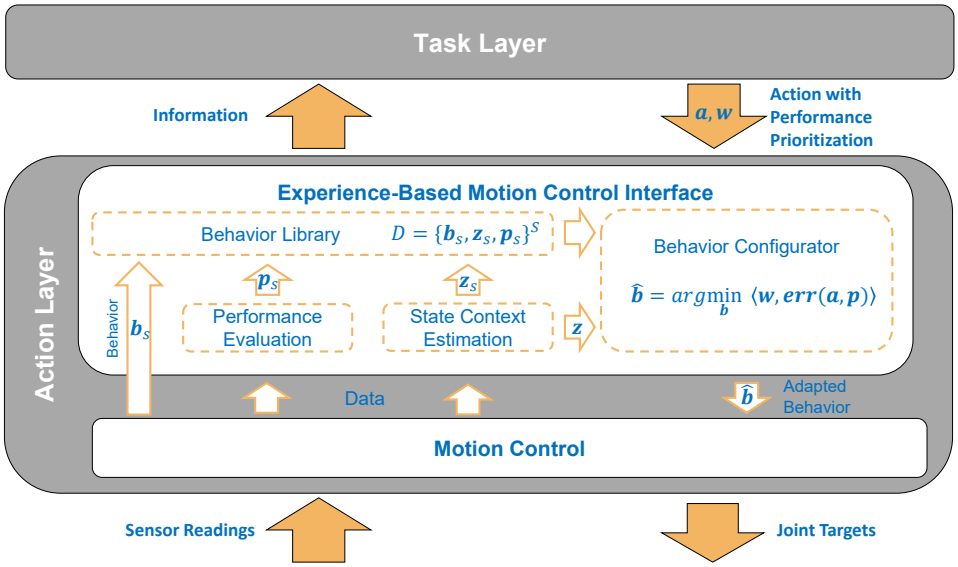


Figure 4.1: Experience-based behavior adaptation - Evaluations of applied behaviors in diverse state contexts are collected and stored in a behavior library. These experiences are used to find the supposedly best behavior for a black-box motion controller with respect to the current context, i.e. desired action, performance prioritization, and state context (state of robot and environment).

experiences are provided by the *behavior library*. It handles knowledge bases for different applications, allows experience queries, and provides utility functions. Every knowledge base is a collection of experiences $D = \{\boldsymbol{b}_s, \boldsymbol{z}_s, \boldsymbol{p}_s\}^S$, whereas each experience represents a sample of the measured performance of a behavior in an applied state context. Thus, it is formed by a tuple of inputs, which are the currently applied behavior parameters ($\boldsymbol{b}_s$) and state context features ($\boldsymbol{z}_s$), paired with performance features ($\boldsymbol{p}_s$) as outputs. The state context estimation and the *performance evaluation* module provide the required samples. Both, state context and performance features are characterizing the whole evaluation period. During this application-specific time span, it is assumed that the behavior stays constant. This does not mean that the robot's movement is constant, instead it refers to the way it behaves or reacts stays the same.

The intention to learn a mapping on the base of features that represent state context and performance is the following. First, by introducing domain-specific knowledge in the feature generation lowers the requirement on the learning side. It is expected that this grey box model approach requires less training data and is more likely to succeed compared to end-to-end learning methods like Deep Neural Networks [Levine et al., 2016, Peng et al., 2017]. Second, the extra effort in deriving the application-specific and interpretable features improves the possibility to analyze and verify the adaptation process.

The methodology to apply the experience-based behavior adaptation is as follows:

1. Define the behavior performance and state context features for the application (Chapter 3).
2. Build up a knowledge base by storing experiences (Section 4.2 and Section 4.4).
3. Choose an appropriate inference type for the behavior configurator and optionally optimize it with respect to the collected data (Chapter 5 and Chapter 6).
4. Apply the experience-based behavior adaptation and keep storing experiences (Chapter 7).

4.2 Management of Experiences

The idea of the behavior library structure is to provide suitable behaviors for every action a task planner in the deliberative layer can possibly chose. Consequently, the behavior library is divided into independent action-specific sub-libraries (Figure 4.2). To account for the assumption of this thesis that the same action is realizable by various approaches, each action has potentially several motion controllers available to chose from. Because experiences of different control approaches cannot be merged, a separate knowledge base is managed by the behavior library for every one of them. Thus, the handling of multiple motion controls during the actual behavior adaptation step is done as follows: First, the supposedly best behavior parameters are derived for every motion control. And second, their respective performance estimation is used to select between them.

In order to compare action and context attributes of different scales and units, it is required to work with normalized values. Thus, action and state context limits are defined that are specific for every action type. In addition, algorithm-specific behavior limits are defined for every action and motion controller combination, which on the one side are needed to define exploration borders for the autonomous behavior evaluation. On the other side, they are needed to distinguish between minor and major parameter changes. All limits define the exploration boundary for the knowledge base. Thus, they are additionally utilized to setup an automated behavior evaluation framework as described in Section 4.4.

Usually, an algorithm has many behavior parameters but only a few have a high impact on the robot behavior and are changed frequently. Thus, a default parameter set is stored for every algorithm and whenever a behavior adaptation is derived, only the difference to the default parameters is applied. This default behavior is usually generated by an expert when he or she manually sets up the standard robot control.

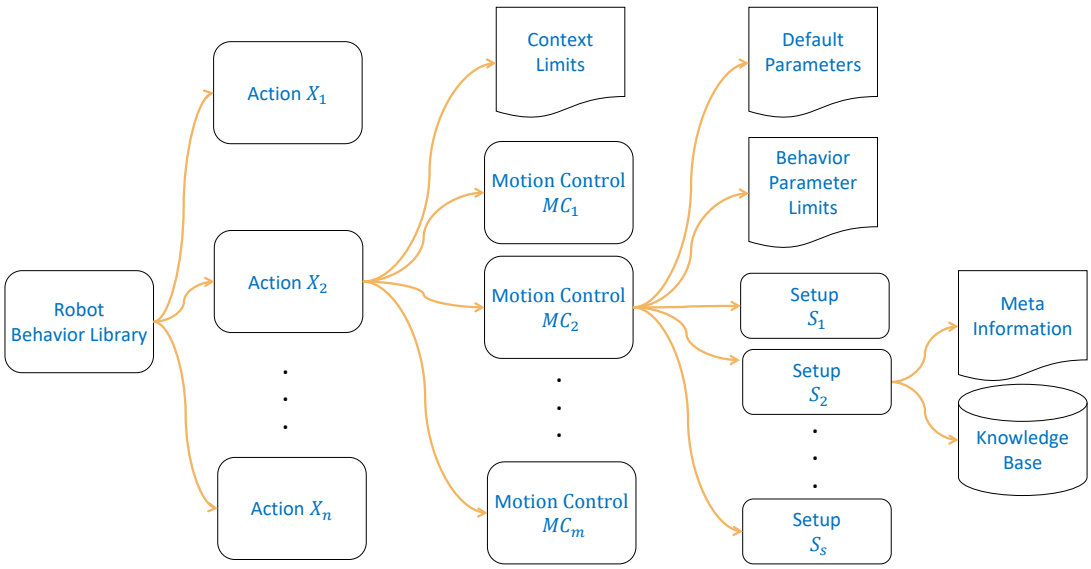


Figure 4.2: Structure of a robot's behavior library consisting of knowledge bases for potential actions, which may be realizable with different motion controllers or corresponding motion controllers.

It is always available and executed whenever no experiences for a specific motion controller are available.

The knowledge base holds all experiences of executed behaviors, where each single experience consist of all inputs, namely behavior parameters and state context features, as well as all corresponding outputs, namely performance evaluation features. In addition, setup information is included that helps to distinguish between different robot states and prevents of mixed incomparable experiences. For instance, a simulated robot will acquire different experiences than the real one. Also different robot modifications can be present for a limited period of time, e.g. different leg designs with or without compliant structures. By setting the setup type, only the corresponding knowledge base is used to store or query experiences, thus, filtering out irrelevant experiences.

4.3 Implementation

All components are implemented in C++ libraries, that are independent from the used robot operating system. This way, only commonly used external libraries like Eigen[1] are the only dependencies. Libraries like the behavior evaluation can be reused in other software without being dependent to the behavior library.

[1] `http://eigen.tuxfamily.org`, accessed on 12/16/2019

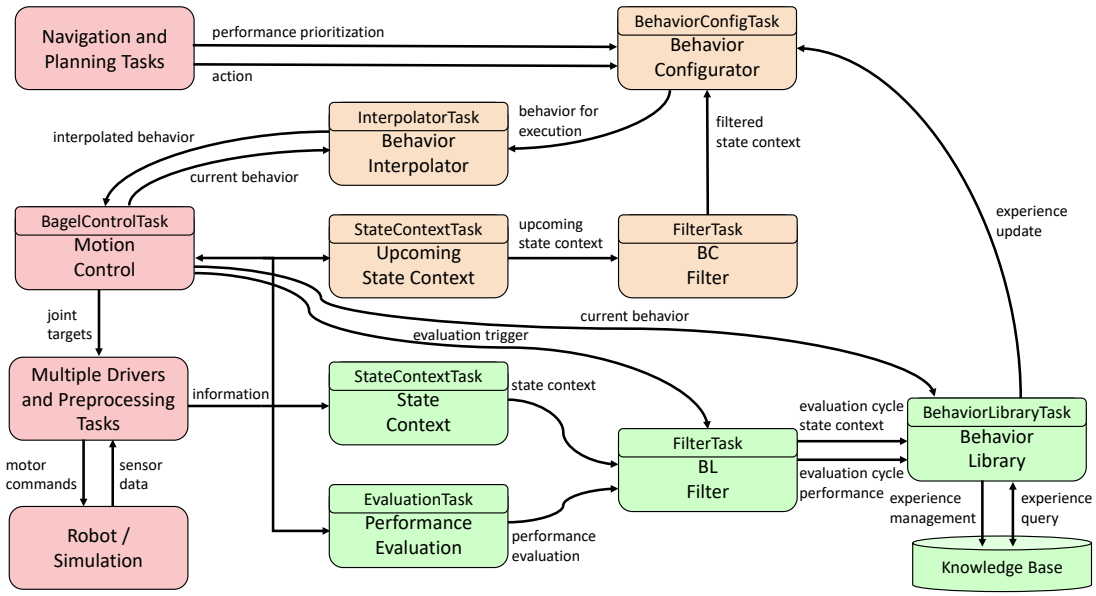


Figure 4.3: Launched ROCK tasks and their interconnections. Tasks of an existing control approach are colored in red. The required tasks to store experiences are colored in green. Tasks that are needed for the autonomous behavior adaptation are colored in orange.

Yet, the modular robot operating system ROCK is used to connect the different libraries. So, for every component, a ROCK wrapper is implemented. In addition, small utility tasks are developed that manage framework-dependent issues. A Ruby script is used to launch and connect all ROCK tasks which are depicted with their interconnections in Figure 4.3. Without the behavior adaptation components, it is assumed that an arbitrary motion control generates joint targets and interfaces the robot (or its simulated counterpart) via multiple drivers or preprocessing components that provide information, e.g. COM computation on the base of joint readings. In addition, the navigation and planning layer produces the desired action and a corresponding performance prioritization.

Whenever experiences should be stored, a task for estimating the state context, a task for evaluating executed behaviors, a filter task, and the behavior library itself need to be started. The filter task continuously collects the generated state context and performance features and derives for every one of them a representative scalar that characterizes the latest evaluation cycle, e.g. the mean is used for the estimated slope and the maximum value for the obstacle height. The evaluation period in the locomotion scenario depends on the periodic walking gait. The trigger to store experiences is sent by the motion control as soon as a complete step cycle in a constant behavior was accomplished. The behavior library then synchronously stores the state

context and performance evaluation features together with the currently applied behavior in the knowledge base. In the current implementation, the behavior library structure as depicted in Figure 4.2 is stored in the filesystem using YAML[2] for config files as well as coma-separated values files to store experiences in a knowledge base.

For the actual autonomous behavior adaptation, the behavior configurator is needed. In the target application, the desired motion command (longitudinal, lateral, and rotational velocity) from the trajectory follower of the navigation layer must be transformed into control-specific behavior parameters, e.g. $\boldsymbol{l}_{step}$, t_{cycle}, and ψ, with respect to the current state context. To be able to react to obstacles which appear in the near future, a second state context estimator is used, which evaluates not only the ROI beneath the robot, but also the area that is traversed within the next step cycle. A second filter task is used to filter out noise and to provide an up-to-date state context estimation for the behavior configurator.

There exist three situations that trigger the behavior configurator to infer a new behavior: (1) the action context changes, i.e. the commanded target values or the performance prioritization weights, (2) the state context changes, or (3) the behavior library notifies the arrival of new experiences. The later triggers an update of the behavior configurator's internal model. Because the model update can lead to a new optimal solution for the current context, the search for the supposedly best behavior needs to be initiated. The behavior configurator has the configuration option to inhibit a behavior adaptation when (1) or (2) occurs. This is used to enforce the execution of a constant behavior for one entire evaluation cycle to guarantee the evaluation and storage of new experiences. In any case, the new behavior is not applied directly. An interpolator interacts between behavior configurator and motion control and fades from old to new behavior. Because of this and the fact, that the motion control allows continuous changes of its input parameters, a smooth transition between walking gaits is realized without interruption.

4.4 Generating Knowledge Bases

A proper knowledge base provides the foundation of the experience-based behavior adaptation. Stored experiences similar to the current context will improve the reliability of the derived behaviors that will be executed. In order to cover a large state context range, behaviors need to be evaluated in as many different state contexts as possible. In order to increase the chance of applying the global best behavior, as many behaviors as possible need to be evaluated in each state context entity. Because every evaluation needs time and due to the curse of dimensionality, only a sparse input space coverage can be achieved, which is especially true at the beginning of the

[2] https://yaml.org/, accessed on 12/16/2019

lifetime of a robot. Thus, a sophisticated behavior configurator is needed that can generate a suitable behavior with knowledge bases of many but also of few experiences.

Because the proposed approach features continuous collection of new experiences, it only needs to be focussed on the generation of a solid initial knowledge base. If a proper behavior adaptation can be realized with an initial knowledge base, further experiences will be integrated over time increasing the knowledge base's competence, in terms of growing evaluated contexts and behaviors, and reliability in terms of solidifying of already included behavior evaluations. In order to store experiences within the robot's behavior library, the motion control and the components to generate and store experiences need to be started, i.e. the state context estimation and behavior evaluation to generate the corresponding features, the behavior library filter to map them to the actual evaluation period, as well as the behavior library task.

Thereby, no difference is made between the physical system or its simulated counterpart except that the setup identifier is set accordingly to avoid an unintended merge between artificial and real experiences, which may be used further post processing steps separately. Experiments with the physical system may include performance degradation through wearout which is an interesting aspect that has to be considered for choosing the right behavior inference approach. However, in most cases, repeatable experiments need to be conducted to compare different control or behavior generation approaches, which motivates the use of a simulator. In the following, the simulator that is used within the thesis including the simulated environments are described. Afterwards, a manual and automatic procedure to generate knowledge bases are described. In addition, the knowledge bases that were generated and used for the experimental evaluation are summarized.

4.4.1 Simulated Test and Evaluation Environments

Generating experiences in simulation has the advantage that various test scenarios can be realized with low effort and repeatable experiments can be conducted. The generation of artificial experiences through simulation will yield meaningful results as long as the simulation reality gap is tolerable. In addition, critical behaviors, which can bring valuable positive and negative experiences, can repeatably be tested in simulation and a thread to the physical system can be avoided. Another advantage is that multiple simulation instances can be started in parallel to speed up the knowledge base generation process.

Within this thesis, the open source simulator MARS[3] is used. It is rigid body simulation that uses ODE[4] as physical core and provides 3D visualization via OSG[5] and QT[6]. By providing a kinematical model with masses and inertias, e.g. in the Unified Robot Description Format, and additional annotations like motor and sensor characteristics, a realistic simulation can be launched. There exist all simulated counterparts for all sensors of the target system. The utilized ROCK wrapper provides the same interfaces as the physical systems. In addition, tools and plugins are provided to load external height maps and to generate own evaluation scenarios.

The simplest scene to test the performance of walking behaviors for legged systems on rigid ground is an empty plane. By moving the robot in different postures and speeds in longitudinal and lateral directions while superposing rotational commands, all performance features (described in Section 3.4) can be evaluated and behaviors can be compared in terms of action performance, stability, and efficiency. Nevertheless, testing in different state contexts allows to gather important experiences about the target system. Thus, obstacle types like steps, random stepping fields, and slopes are used to generate demanding obstacle courses.

In 2013 within the DARPA robotics challenge, a test track[7] was included to test the mobility of state-of-the art legged systems. It is also a good test track to evaluate diverse walking behaviors of the target systems SpaceClimber and Charlie. Thus, a replica in MARS (Figure 4.4) was created following the published dimensions of the real obstacle course. The short test track includes a ramp of 15°, a step of 0.1 m height, and two brick fields consisting of 0.39 m x 0.39 m x 0.1 m steps, where the first has horizontal steps and the second 15°-inclined steps.

The German Research Center for Artificial Intelligence (DFKI) also owns an outdoor test track, located at the Robert-Hooke-Str. 5 in 28359 Bremen, Germany. It contains railways, gravel and scattered stone fields, a bridge, a hill, and stairs. A simulated obstacle course was formed out of these elements, which is further referred to as *outdoor* test track. Its concrete specifications can be seen in Figure 4.5a, which is simultaneously the scene specification file to automatically generate the MARS scene that is depicted in Figure 4.5c.

The DFKI also has manifold possibilities to build up indoor test courses. Besides euro-pallets and adjustable ramps, a modular test course is availbale [Dettmann et al., 2016] that allows an easy and versatile buildup of obstacles. The single elements comply with the American Society of Testing and Materials standards on mobility while using affordable and everywhere available components to

[3] `https://github.com/rock-simulation/mars`, accessed on 12/16/2019
[4] `http://www.ode.org`, accessed on 12/16/2019
[5] `http://www.openscenegraph.org/`, accessed on 12/16/2019
[6] `https://www.qt.io/developers`, accessed on 12/16/2019
[7] `http://archive.darpa.mil/roboticschallengetrialsarchive/`, accessed on 09/12/2014

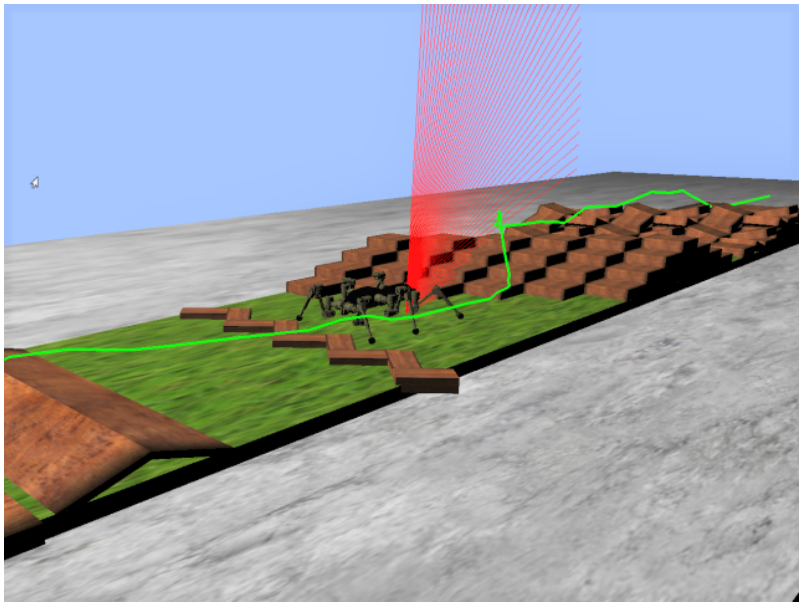


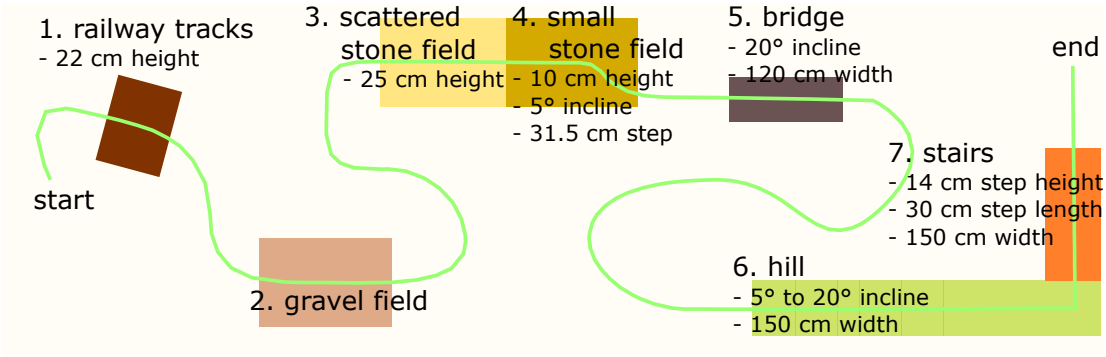
Figure 4.4: Replica of the 2013 Darpa Robotics Challenge test track

make a repeatable recreation possible. Thus, a simulation scene was created to test on similar obstacles within the simulation (Figure 4.5b and Figure 4.5d). The resulting obstacle course is further referred to as *indoor* test track.

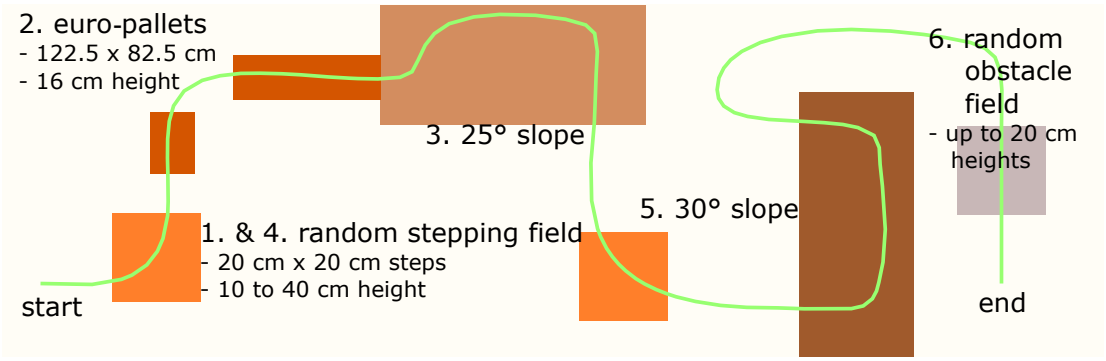
4.4.2 Knowledge Bases

To generate an initial knowledge base, in principle two approaches are possible: (1) manual setting of behavior parameters or (2) automated selection and evaluation of behaviors for available evaluation scenarios. In the following, the utilization of both options is described in detail and an overview is given which knowledge bases were created and used in the experimental evaluation.

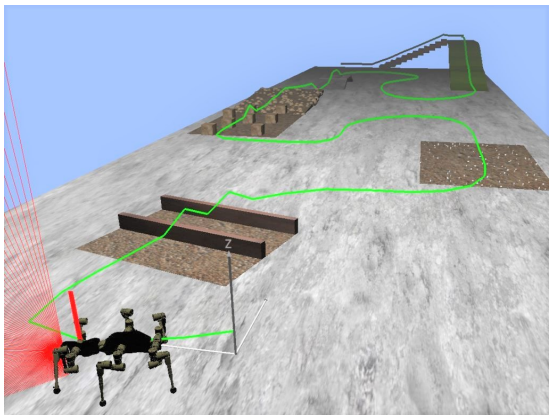
In order to analyze different aspects, several knowledge base were built with the target systems SpaceClimber and Charlie. A short unique identifier abbreviates the robot name (C=Charlie, S=SpaceClimber), the type of experience (S=simulated, R=real), the type of behavior selection (M=manual, A=Automatically), and has a small postfix to indicate an auxiliary description. For both target systems, the behavior parameters listed in the Appendix are principally available for all knowledge base generation runs. A detailed description of every behavior parameter and its influence is described in Section 3.3. However, not all behavior parameters were varied during each knowledge base acquisition. Thus, the number of behavior parameters number of parameters representing a behavior (B) that can be used for the behavior adaptation is depending on the knowledge base. Table 4.1 summarizes all knowledge bases and their key facts.



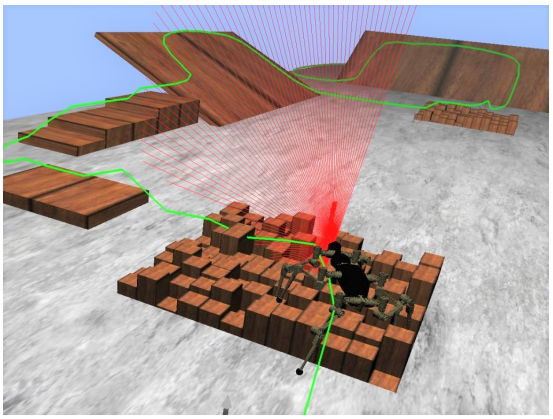
(a) outdoor test track schematic



(b) indoor test track schematic



(c) simulated outdoor test track



(d) simulated indoor test track

Figure 4.5: Simulated test tracks to evaluate walking behaviors

Manual Knowledge Base Generation

As long as the tasks for storing experiences are running (Figure 4.3), experiences are collected automatically. Manual knowledge base generation means that the executed behavior is manually chosen. This has the advantage, that the huge parameter space is searched by common sense or with domain-specific expert knowledge. The drawback comes with the bias towards the individual operator including the avoidance of solutions that are not considered by the operator.

In order to generate knowledge bases that contain plenty well-performing behaviors, the simplest way is to evaluate the behaviors that are chosen by an experienced operator while he or she is controlling a robot. This was done for SpaceClimber, while an expert was controlling SpaceClimber through the simulated outdoor test track (`SSM_Exp`). For Charlie, a manually derived knowledge base was generated within simulation (`CSM_Exp`) and with the physical system (`CRM_Exp`).

Of course, depending on the operator, different behaviors get evaluated, which directly influences the autonomous adaptation process later on. In order to analyze this operator bias, several operator-specific knowledge bases were additionally acquired for SpaceClimber. Therefore, every operator had to manually adapt the walking behavior of SpaceClimber, i.e. select appropriate behavior parameters for its motion control, in order to lead SpaceClimber through a demanding obstacle course. The diverse obstacle types required different behavior types.

In addition, the operators knew beforehand, that they were evaluated by several criteria to compare their skill level, to force the exploration of different behaviors, and to obtain reference values to compare the autonomous control with respect to:

- Position Accuracy: A path was provided to guide the operator through the obstacle course (green line in Figure 4.5). The distance to the path was evaluated to motivate the operators to stay on the path and thus set different turning angles. In addition, the operators were informed that a small corridor of ± 0.1 m resulted in full score to avoid continuous motion corrections, which would inhibit behavior evaluations because a behavior needs to stay constant for a complete walking cycle in order to get evaluated.

- Energy Efficiency: The overall consumed energy divided by the test course distance was used to enforce fast behaviors whenever possible.

- Stability: The average `dsa` was used as metric to enforce the selection of stable behaviors that maintain a possibly low posture.

- Failures: The operators had to avoid SpaceClimber to tip over or to leave the provided area. If such incident occurred, it was recorded and SpaceClimber was set back to the closest path position to proceed the experience generation.

Figure 4.6: Operator control station to operate SpaceClimber through a demanding obstacle course in order to record chosen behaviors, the sensed state context, and the resulting behavior performance metrics.

The operators had possibly best conditions to control the robot due to an experiment setup including a 3D view on the robot in real-size from different perspectives as well as two touch screen monitors to visualize the robot's telemetry data and to conveniently set the control parameters using a graphical user interface (Figure 4.6). Thus, the operator could focus on the behavior adaptation. In order to get familiar with the control, the operators could practice on the DRC-Track before loading the real tracks and actually starting the recording.

Five operators of all skill levels were selected for controlling SpaceClimber through the simulated indoor track, whereas another five operators performed on the simulated outdoor obstacle course, resulting in ten operator-specific knowledge bases (`SSM_InOp1-5` and `SSM_OutOp1-5`). Depending on the operator, the behavior parameters that were adapted vary from person to person. Also the number of experiences shows large variations. On the one side, cautious operators rather chose slow behaviors and closely monitored the executed action before varying parameters. Thus, many complete step cycles could be evaluated. On the other side, other operators experimented a lot with the parameters and continuously fine-tuned their selec-

tion to optimize SpaceClimber's walking behavior. Even though less behaviors could be recorded, their parameterizations had a larger parameter space coverage. This is also reflected by the average evaluations per behavior in Table 4.1. A high value indicates more mature knowledge about every behavior, whereas a low number indicates much anecdotal evidence. In order to form more mature knowledge bases, the single indoor and outdoor knowledge bases were merged to two track-specific knowledge bases (`SSM_In`, `SSM_Out`). Finally, a combined knowledge base was merged to `SSM_Full` containing all experiment experiences.

Automated Behavior Evaluation Framework

The automated behavior evaluation ideally explores the whole input space to ensure that the best behavior is being evaluated to be available for the later behavior adaptation. Unfortunately, a behavior vector of $\mathbb{R}^B$ can create a large configuration space to investigate. Even if this space is roughly covered through pointy behavior evaluations, it has to be repeated within the $\mathbb{R}^Z$ state context space in order to get a general view on the full input space. Assuming that a coverage of each dimension of E evaluations is desired, then, E^{B+Z} evaluations have to be done. Due to the fact that the state context dimensions cannot be commanded because they are sensed by the robot, one can only create distinct different situations (Z_s) which will cause behavior evaluations within specific regions of the state context space. Due to this fact, only $Z_s E^B$ evaluation scenarios need to be launched. Of course, larger values for E, Z_s, and especially B will result in a time consuming effort that cannot be handled by the manual knowledge base generation. For instance, setting E, Z_s, and B to 5 will cause 15625 evaluations which results in 43,4 h of evaluation time if every evaluation will take 10 s on average.

The concept of the developed automated behavior evaluation framework is depicted in Figure 4.7. The configurable *experiment controller* starts and stops the required software modules, manages the data bases, monitors the experiment execution, and handles potential error cases. The evaluation process itself basically consists of two loops. In the outer loop, one behavior after the other is selected from a pool of behaviors for the evaluation process. This pool can be fed by handcrafted behavior parameters or by parameters that resulted from behavior learning. Theoretically, strategies like novelty search [Lehman and Stanley, 2008] could also be used to generate behaviors during the evaluation process to intelligently guide the exploration of the parameter space. In this work, a behavior generator was implemented, that derives all possible behavior combinations by defining the variable parameters, their intervals, the number of values per parameter, and optionally added noise. So, numerous behaviors are quickly generated which can be used to create a knowledge base that covers the broad range of the behavior space.

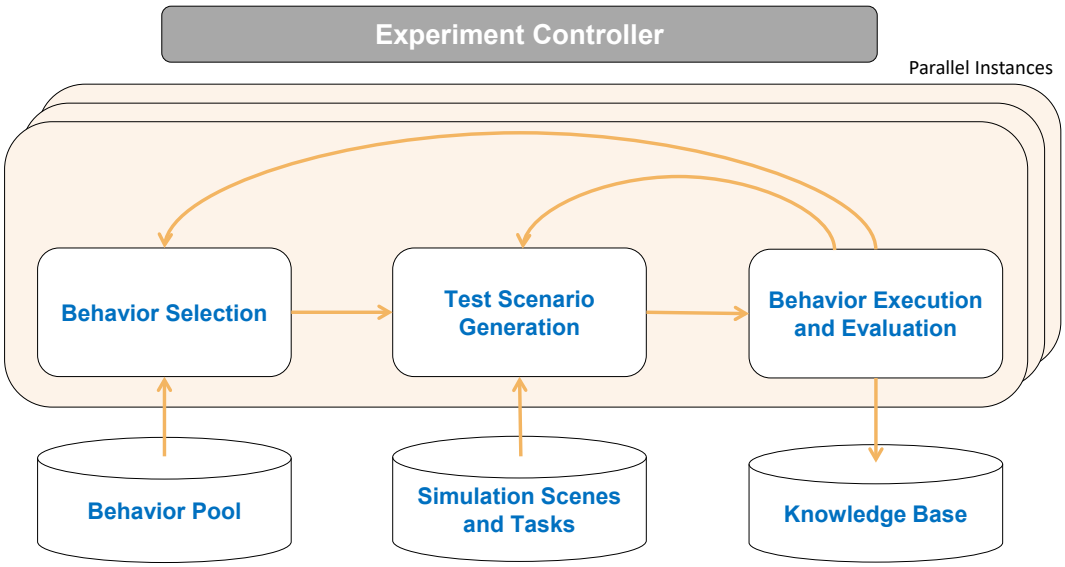


Figure 4.7: Flowchart of the automated behavior evaluation framework

When a behavior is selected for evaluation, an inner loop continuously generates new simulation scenes in order to create manifold state contexts while the behavior is executed. The evaluation time can be defined or a terminate signal, e.g. when the action execution finished, can be used in order to trigger the generation of a new scenario. When a behavior is evaluated within all scenarios, the outer loop is triggered again to select the next behavior. This way, the performance of a behavior for many state contexts is gathered. Especially when having a pool of behaviors that were optimized for one distinct context or context range through machine learning, getting to know the performance in other state contexts delivers valuable information that motivate the autonomous behavior adaptation.

The main advantage of the automated behavior evaluation framework is the possibility to parallelize the evaluation process. This feature requires to launch the involved components of the outer and inner loop several times. In order to avoid cross influences between each launch, name tags are utilized to generate unique task names within ROCK. In addition, each behavior selection instance writes in a specific experiment file which behavior will be evaluated next and thus, blocks it for other behavior selection instances. To lower the computation costs, the whole simulator is then launched with no graphical components. So, multiple behaviors can get evaluated simultaneously on one computer. The possibility to distribute the knowledge base generation automatically on several computers was not realized, as more than ten parallel evaluation instances could be launched on one computer which made most knowledge base generation processes finish within a day.

Table 4.1: Generated knowledge bases - the identifier abbreviates the robot (C=Charlie, S=SpaceClimber), the setup type (R=Real, S=Simulation), the type of generation (A=Auto, M=Manual, and a label for special conditions). The number of average evaluations per behavior indicates whether only anecdotal data or more reliable data based on multiple evaluations has been collected. The experience time represents the time purely used for the evaluation not considering the time for launching the scenarios or situations which have not resulted in storing of experiences.

Identifier	B	Z	A	Nr. of Experiences	Nr. of Behaviors	Avg. Behavior Evaluations	Experience Time in h
SSM_InOp1	11	4	10	223	78	2.9	0.25
SSM_InOp2	9	4	10	86	50	1.7	0.07
SSM_InOp3	13	4	10	54	43	1.3	0.09
SSM_InOp4	9	4	10	64	48	1.3	0.06
SSM_InOp5	10	4	10	64	49	1.3	0.07
SSM_OutOp1	10	4	10	141	75	1.8	0.14
SSM_OutOp2	13	4	10	124	72	1.7	0.10
SSM_OutOp3	8	4	10	111	63	1.8	0.08
SSM_OutOp4	11	4	10	101	47	2.1	0.08
SSM_OutOp5	13	4	10	244	142	1.7	0.24
SSM_In	16	4	10	491	268	1.8	0.55
SSM_Out	19	4	10	721	324	2.2	0.64
SSM_Full	19	4	10	1212	591	2.0	1.19
SSM_Exp	13	4	10	1733	194	8.9	2.27
CSM_Exp	19	0	10	2840	475	6.0	5.32
CRM_Exp	17	0	10	258	103	2.5	0.37
CSA_Walk	8	0	10	21070	6614	3.2	31.2
CSA_Trot	7	0	10	21911	6527	3.4	16.6
CSA_Full	8	0	10	42981	13141	3.3	47.8

The automated behavior generation framework was used to generate knowledge bases for Charlie. First, a knowledge base that only consists of walking behaviors on flat ground was generated (CSA_Walk). It allows the analysis of different posture and walking pattern combinations in terms of stability and efficiency. The same was done for trot behaviors (CSA_Trot). Both knowledge bases were also combined to one knowledge base (CSA_Full) to be able to analyze gait transitions within the autonomous behavior adaptation. It is evident, that the automatically generated knowledge bases have less inputs, but contain much more evaluations. Thus, the input space is more densely covered than the manually recorded knowledge bases. However, with approximately three values for every input parameter, a sparse data space coverage is still the case.

4.5 Conclusions

This chapter presents the concept of experience-based behavior adaptation. It is a data-driven approach that derives the desired mapping from state context, desired action, and performance prioritization to behavior parameters for a motion control by inferring from collected experiences, i.e. evaluated behaviors in various state contexts. Abstracting the motion control to a black box and inferring from past examples avoids hard-coded, robot-specific implementations and provides the possibility to generate new solutions for novel situations and to track long-term changes.

The behavior library is introduced to manage different knowledge bases for multiple actions that a multi-purpose robot can perform. It also handles knowledge bases for incompatible motion controls that can realize the same action but might have their strengths within specific contexts. An implementation example is provided that shows the required modules to store experiences and to use them for context-dependent motion control adaptation.

In addition, the methodology to generate knowledge bases is described. Besides collecting experiences while an operator manually selects behaviors, an automated knowledge base generation framework is presented that exploits the input space and speeds up the time consuming evaluation process. Several generated knowledge bases are described that are used within the experimental evaluation. Even though each knowledge base consist of numerous evaluated behaviors, every one of them only sparsely covers the full input space of the target application. Their different sizes, number of inputs, and type of generation vary to allow the analysis of different aspects. The actual usage of the gathered data to adapt the behavior of a robot during runtime is done by the behavior configurator. Two approaches are proposed, which are described and analyzed in the following chapters.

Chapter 5

Case-Based Behavior Inference

The goal of the behavior configurator within the experience based behavior adaptation is to derive the supposedly best behavior for the current context. This chapter describes the first of two developed approaches. It follows a CBR paradigm that tries to solve problems by reusing past solutions for similar problems. In CBR, a past problem-solution pair is represented by a *case* and is stored and organized in a memory, called *case base* . Through four steps, a solution for the current situation is generated and the case base is updated to increase the reservoir of solutions (Section 2.3). Figure 5.1 depicts the resulting inference scheme and its inputs and outputs. First, cases are retrieved from the case base that are similar to the problem statement, i.e. with respect to the current state context z, the desired action a, and performance prioritization w. Second, the solutions of the respective cases are reused to derive a new solution, which is the supposedly best behavior $\hat{b}$. This solution is then revised to gain new experiences, which correspond to samples of the evaluated behavior b_s in state context z_s and the resulting performance p_s. The samples are finally retained by storing them in the case base.

The following section describes in detail how the concept of CBR is applied for the intended autonomous behavior adaptation. First, the case-based representation of a knowledge base is presented, which has a high influence on the case retrieval and storage. Then, more details on the similarity-based rating for the case retrieval are provided. Afterwards, the application of rated behaviors to generate an executable solution is described. Finally, the whole approach is utilized to adapt the walking behavior of SpaceClimber to overcome demanding obstacle. Multiple experiments are analyzed and a final conclusion is drawn.

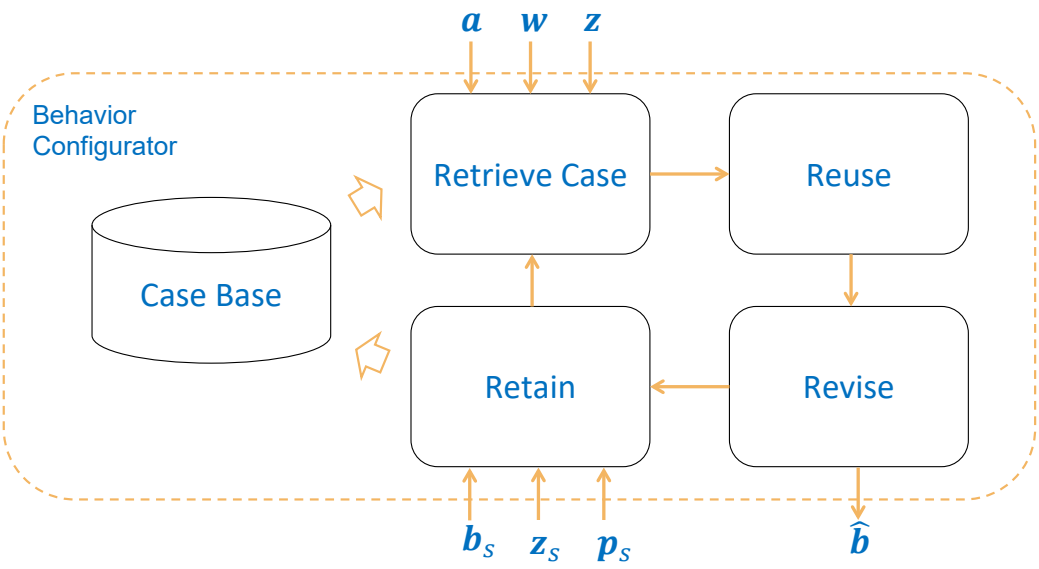


Figure 5.1: Concept of the case-based behavior inference approach. The current context in form of desired action a, performance prioritization w, and detected state context z are used to retrieve the most similar case of the case base. Its solution $\hat{b}$ is reused and applied, which is then revised by evaluating its execution. The final retain step stores the performance p_s of the executed b_s together with the traversed state context z_s in form of cases to enlarge the case base.

5.1 Case-Based Representation of Knowledge Bases

In order to integrate a CBR in the proposed behavior adaptation architecture, first, a case description needs to be specified. As a contextual piece of knowledge representing an experience, a case has to contain:

- a problem description, which describes the state of the world when the case occurred, here the state context
- a problem solution, which is in the context of this work a behavior
- the outcome, which is the result of the applied behavior in this situation, namely the behavior performance
- additional information, that helps to interpret the origin of the data for later post processing or filtering

Considering above aspects, the following case representation was chosen (Figure 5.2). The behavior is described by an unique `algorithm` label and a corresponding `parameter behavior` which is a file holding the configuration parameters. Since a behavior can be evaluated in multiple scenarios, the case contains a

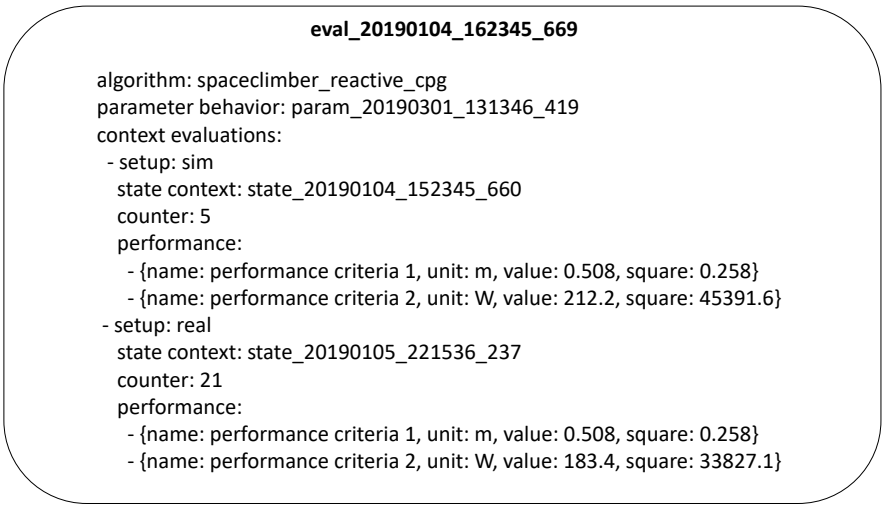


Figure 5.2: Exemplary representation of a case that represents a behavior that was evaluated multiple times in two different contexts.

list of `context evaluations`. Each of them has a `setup` identifier, the `state context` during evaluation, this list of `performance` features, and a `counter`.

The `setup` is providing miscellaneous information about the kind of experience. This can help to filter out data or to avoid mixing up of experiences which do not belong together, e.g. evaluations from a broken robot, from simulation, or from the physical system. Even though physical and artificial experiences usually should be separated, a real robot could use a simulation-trained case base as a starting point. When real data comes in, it can be utilized to replace the initial artificial data. Consequently, both experience types can be valuable until enough experiences are collected with the real system.

Instead of directly describing the `state context` during evaluation, a link to a context file is provided that lists the actual values of all state context features. This has the advantage, that context evaluations of other behaviors in an already evaluated state context can link to the same context file to safe memory.

The `performance` is represented by a list of performance features, each described by `name`, `unit` to avoid later misinterpretations, and statistical measures. The latter is very crucial to minimize the danger of false behavior proposals due to anecdotal evidence. Therefore, `value` is representing the mean value $\overline{p}_i$ of performance feature i over `counter` evaluations. In addition, `square` represents the mean of the squared values $\overline{p^2}_i$. This way, the variance $\sigma^2_{p_i}$ of each performance feature can be computed with

$$\sigma^2_{p_i} = \overline{p^2}_i - \overline{p}_i^2 \tag{5.1}$$

without the need of storing all prior performance values.

The variance of a performance feature can be utilized to evaluate its trustworthiness and can be considered for the case retrieval especially in safety critical situations. The maximum `counter` value can be limited to control the influence of an reevaluated case. In addition, it can be used to filter out experiences with too few significants, e.g. to avoid reasoning on a potentially noise-inflicted single evaluations.

When creating the case base from an existing knowledge base or when new experiences are incoming, multiple storing options need to be distinguished as depicted in Figure 5.3. At the beginning, it is checked whether a case for the current behavior was already created. If not, a new case is stored and a parameter behavior file is generated. If the current behavior was already evaluated, then the new evaluation is added to the existing case. Therefore, it is checked whether the current state context was ever evaluated before. This is very unlikely when a continuous state context representation is used. Consequently, binning of the current state context features is performed in application-specific reasonable discretization steps. Here, a balance in granularity is needed between loosing too much information and the generation of too many classes, which might be irrelevant concerning the noise in the state context estimation as well as the intrinsic adaptation capability of the reactive motion control. In this work, the range of each context feature was discretized to ten equally sized bins to reach this balance. However, if a state context was not evaluated before, a new file containing the discretized state context features is generated, otherwise the existing file is referenced. If the state context is new for the current behavior evaluation, a new context evaluation is added. Otherwise, the counter of the corresponding one is incremented and the values $\overline{p}_i$ and $\overline{p^2}_i$ are calculated.

5.2 Similarity-Based Rating of Cases and Behavior Derivation

Assuming that a knowledge base for the current application is available and a case base was acquired, CBR can be applied to derive the supposedly best behavior $\hat{\boldsymbol{b}}$ for the desired action $\boldsymbol{a}$ with performance prioritization $\boldsymbol{w}$ and the current state context $\boldsymbol{z}$. It starts with case retrieval that is supposed to find cases which are usually most similar to the current situation. Within this work, the fact that many behaviors will solve the current problem but with diverse performance needs to be addressed as well. Thus, cases need to be found which have good performance according the actual performance prioritization and were evaluated in a similar state context. Otherwise, a "bad" behavior which was evaluated in the exact same context or a "good" behavior that was evaluated in a complete different scenario might be selected. Consequently, the similarity s_c of every case in the memory is computed by calculating the action

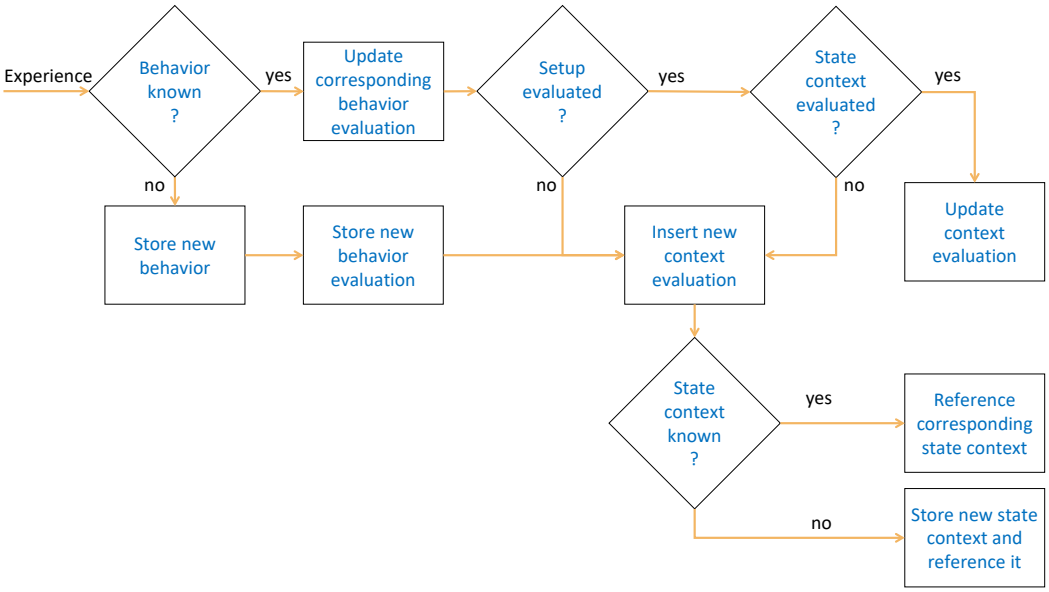


Figure 5.3: Flowchart for storing new experiences in the case base

similarity s_a and weighting it with its state context similarity s_z:

$$s_c = s_a s_z. \tag{5.2}$$

Since a behavior could have been evaluated in several state contexts, a case can contain multiple context evaluations. To obtain the most relevant performance similarity, the context evaluation with the highest state context similarity $\hat{e}$ needs to be retreived with

$$\hat{e} = \underset{e}{\operatorname{argmax}} s_z(e), \tag{5.3}$$

where the state context similarity of any context evaluation $\boldsymbol{e}$ is computed by

$$s_z(e) = 1 - \frac{\boldsymbol{w}_z^T \boldsymbol{err}(\boldsymbol{z}, \boldsymbol{z}_e)}{\sum \boldsymbol{w}_z}, \tag{5.4}$$

with $\boldsymbol{err}()$ being a row-wise error function. Here, the element-wise squared error is used to lower the importance of state contexts that have one feature far apart in comparison to state contexts where all features have a little difference. Thereby, $\boldsymbol{z}_e$ is the state context feature vector of the corresponding context evaluation $\boldsymbol{e}$. The weighting vector $\boldsymbol{w}_z$ allows to scale the influence of state context features to incorporate known uncertainties, e.g. state context features that are computed on the base of visual sensor data in bad lighting conditions. Within this work, $\boldsymbol{w}_z$ is a unit vector and can be neglected. Since context describing features usually have different units and scales, all values are normalized according to the application specific limits, as

stated in Section 4.2 before building the scalar product. The result is normalized by the sum of weights to create a normalized value between zero and one, where zero indicates identical state contexts and one the complete opposite. Thus, it is subtracted from one to obtain a similarity measure.

Having identified the most relevant context evaluation $\hat{e}$, one can set the state context similarity of a case to the one of the most similar context evaluation

$$s_z = s_z(\hat{e}). \tag{5.5}$$

It follows that the action similarity is computed with

$$s_a = s_a(\hat{e}) = 1 - \frac{\boldsymbol{w}^T \boldsymbol{err}(\boldsymbol{a}, \boldsymbol{p}_{\hat{e}})}{\sum \boldsymbol{w}}, \tag{5.6}$$

where $\boldsymbol{p}_{\hat{e}}$ refers to the performance of the context evaluation with the highest state context similarity. As stated in Section1.2, $\boldsymbol{a}$ contains the desired action features and constants for the desired optimal meta performance features. Thus, $\boldsymbol{a}$ and $\boldsymbol{p}_{\hat{e}}$ contain the same features. Consequently, as done for the state context similarity, the action similarity is computed by building an error vector that element-wise calculates the squared error of the normalized values, then building the scalar product with the performance prioritization $\boldsymbol{w}$ as well as scaling it with the sum of weights, and finally subtracting the normalized result from one.

Summarizing, a similarity has to be computed for every case and additionally for every context evaluation that the case contains. Imagine a case base that contains C cases and D discretized state context classes, that are characterized by Z state context features. If every case was evaluated with A performance features in every state context, the computational complexity would be $O(C(A + DZ))$. By changing the bin sizes for the state context discretization, the compromise between accuracy and computation time can be adjusted. The number of state context features Z have a larger influence on the computation time than the number of performance features A. However, it is important to highlight that the case retrieval is completely abstracted from the behavior representation and, thus, the behavior size B has no influence.

After the retrieval, all cases are ranked and sorted by their similarity, i.e. their expected performance in the current context. Stopping at this step would already create a powerful expert system. A user can define its problem based on his or her needs that are demanded by the current situation and will receive a ranking of behaviors which can be used to take further decisions. Thanks to the corresponding case data, further information are available, e.g. which contexts were already evaluated, how often, and how reliable the information is.

However, the goal is to autonomously derive a behavior that can be executed in the current context. The fastest solution is to select the behavior of the case with the highest similarity. This is also the safest way, because only tested behaviors are applied and no new behavior is generated. Due to the separation of the behavior similarity computation from its algorithm and parametrization, one has the advantage that even completely different approaches can be compared and selected whenever they are superior to others, e.g. a fast CPG-based control for flat inclines and deliberative planning approach to climb up steep craters.

When having many cases of only one algorithm, one has the possibility to merge their parameter behaviors to form a new parameter set. So, within this work, a k-nearest neighbor merge is implemented, that averages the parameter sets of the K most similar behaviors

$$\hat{b}_i = \frac{\sum_{k=1}^{K} s_c(k) b_i(k)}{\sum_{k=1}^{K} s_c(k)}, \tag{5.7}$$

with b_i being the i-th behavior parameter and $s_c(k)$ the case similarity of the k-th case used as weighting factor. Consequently, with $K > 1$ it is likely that a new behavior is generated. Even though no behaviors with parameter values larger or smaller than the ones which are used for merging can be formed, depending on the application, good performing solutions may be found in the middle.

However, as soon as a behavior is generated, it will be executed and evaluated. In the final retain step, the evaluated performance information will be stored and the case base will be updated. Thus, an incremental learning process is obtained that increases the reasoners competence and certainty. In addition, the continuous update of previously updated behaviors keeps the case base up-to-date. So, long-term changes like wearout are tracked and incorporated in the behavior adaptation process which may lead to the selection of a different behavior for the same context between start and end of the lifetime of robot.

5.3 Application for Locomotion in Unstructured Terrain

The purpose of this section is to analyze the case-based behavior adaptation approach within a demanding target application. The locomotion of the six-legged walking robot SpaceClimber over unstructured terrain is chosen, as it requires a high-dimensional, non-linear mapping from inputs to outputs as well as hard real-time constraints. The goal of the experience-based behavior adaptation is to derive an appropriate behavior for commanded motion commands while considering changing environmental conditions in order to traverse a demanding obstacle course.

Table 5.1: Normalization limits for SpaceClimber's state context and performance features

Name	Min	Max
obstacle_height	-0.1 m	0.4 m
slope_x	-35 $^\circ$	35 $^\circ$
slope_y	-35 $^\circ$	35 $^\circ$
roughness	0 %	100 %
velocity_x	-0.1 $\frac{m}{s}$	0.4 $\frac{m}{s}$
velocity_y	-0.1 $\frac{m}{s}$	0.1 $\frac{m}{s}$
velocity_rot	-10 $\frac{^\circ}{s}$	10 $\frac{^\circ}{s}$
body_height	0.15 m	0.5 m
body_width	0.5 m	1.3 m
ssm	0.0 m	0.5 m
dsa	0 $^\circ$	65 $^\circ$
power	50 W	450 W
epd	0 $\frac{Wh}{m^2}$	4 $\frac{Wh}{m^2}$
vibration	0 $\frac{^\circ}{s}$	10 $\frac{^\circ}{s}$

In a first experiment, the effect of different knowledge bases on the behavior derivation quality is analyzed. Second, the influence of the behavior merging during the case adaptation step is experimentally evaluated. Third, the effect of varying the performance prioritization is analyzed. Then, a small initially acquired knowledge base with only positive experiences is utilized and the effect of incoming experiences is studied. Finally, experiments with the physical system are conducted to evaluate the transferability of a simulation-trained knowledge base.

All experiments have in common that the action context was described by a desired motion command and a body posture. In addition a given performance prioritization weighted the influence of each command as well as the importance of the meta performance features (Section 3.4). The state context was characterized by the slope in longitudinal and lateral direction, the maximum step height, and the terrain roughness (Section 3.5). In order to discretize the state context, every feature was split into eleven equally-sized bins. The normalization boundaries for the state context and performance features were chosen according to SpaceClimber's mobility limits and are summarized in Table 5.1. The estimated context features passed the filter task before getting to the behavior configurator where a sliding window average filter over all values of the last 1 s was applied. The by the behavior configurator generated behavior parameters were sent to the behavior interpolator to ensure a smooth blend form one behavior to the other. The blend time was set to 2 s. Finally, the blended parameters configured the modular central pattern generator (Section 3.3) which actually derived the joint targets.

Table 5.2: Performance prioritization during autonomous traversal of the artificial outdoor test track

symbol	weighted performance feature(s)	weight
$\boldsymbol{w}_{vel}$	[velocity_x, velocity_y, velocity_rot]	[0.8, 0.2, 1.0]
$\boldsymbol{w}_{posture}$	[body_height, body_width]	[0.0, 0.0]
w_{ssm}	ssm	0.1
w_{dsa}	dsa	0.0
w_{pow}	power	0.0
w_{epd}	epd	0.1
w_{vibe}	vibration	0.1

5.3.1 Knowledge Base Comparison

The objective of this experiment is to analyze the relationship between different knowledge bases and the resulting autonomous control of the simulated SpaceClimber.

Experimental Setup

To evaluate how the autonomous control uses the provided flexibility of the robot's locomotor system, it had to overcome diverse obstacle types that require different walking behaviors. As target obstacle course the artificial outdoor test track as visualized in Figure 4.5c and specified in Figure 4.5a was used. The green line represents the target path from start to finish.

Motion commands defining the desired $\boldsymbol{p}_{vel}$ were derived by a trajectory follower to continuously generate action context inputs. Depending on the orientation error, which was built between actual robot pose and a target point located 0.5 m ahead on the trajectory, it set velocity_x between 0 $\frac{m}{s}$ and 0.15 $\frac{m}{s}$ and velocity_rot between -10 $\frac{^\circ}{s}$ and 10 $\frac{^\circ}{s}$. Consequently, maximum velocity was applied when the target point is directly ahead whereas point turns were commanded for large orientation errors. The desired lateral velocity velocity_y was kept at 0 $\frac{m}{s}$. The performance prioritization was empirically determined and fixed as denoted in Table 5.2. In order to achieve a close trajectory following, highest priority was given to velocity_rot followed by velocity_x and velocity_y. epd, ssm, and vibration were chosen as meta performance features to favor stable and efficient behaviors. Their weight was chosen to be lower in order to distinguish between behaviors of equal action performance and not to degrade the trajectory following character. The weight of a potentially commanded $\boldsymbol{p}_{posture}$ was set to zero because it is not relevant in this application. The corresponding behavior parameters were managed autonomously by the behavior configurator.

The utilized knowledge bases were the ones which were recorded during manual operator control (Section 4.4.1), i.e. `SSM_OutOp1`, `SSM_OutOp2`, `SSM_OutOp3`, `SSM_OutOp4`, `SSM_OutOp5`, `SSM_Out`, `SSM_In`, and `SSM_Full`. Table 4.1 contains all information regarding inputs, outputs, and collected experiences. The storage of new experiences was deactivated to conserve the original knowledge bases.

The behavior configurator utilized the above mentioned knowledge bases to autonomously map the incoming motion commands of the trajectory follower to control parameters of the motion control while considering the estimated context. Every knowledge base was utilized and evaluated five times.

Following metrics were used to evaluate the traversal from start to finish. They were normalized to application-specific limits to allow a comparison of the experimental results:

- position accuracy: The distance to the path was recorded and averaged over the trial, whereas a distance of more than 0.5 m (approx. half of the robot width) resulted in a score of 0% and a distance of less than 0.1 m in 100%.

- efficiency: The efficiency was determined either by the actual power consumption normalized between 150 W (100%) and 300 W (0%) or by the specific power consumption, i.e. accumulating the overall energy consumption and dividing it by the distance of the given path. A value of more than 1.4 $\frac{Wh}{m}$ resulted in a score of 0% indicating inefficient walking with SpaceClimber and a value of less than 0.7 $\frac{Wh}{m}$ resulted in 100% representing efficient walking with respect to SpaceClimber's standard efficiency.

- stability: The measured `dsa` was averaged over the trial and used as stability metric whereas a positive angle scored up to the limit of 30° (100%) and instability (0°) returned zero score.

- completed: In the case of a tip over, the robot was set back on the track in order to gather representative efficiency and accuracy values for the entire course. However, in order to account for tip overs, a successful completion was additionally tracked with this metric.

- overall: The overall criteria represents the equally weighted average of the all performance criteria just for reference.

Observations

Because the operators were evaluated during the knowledge base generation process (Section 4.4.2) in the same way, Figure 5.4 can show the performance of the autonomous traversal on the base of different knowledge bases and of the manually operated trials (operator). The operator results are shown as reference and do not represent the full potential of manual control. They have quite a variance due the fact, that operators of different background and expertise were selected. However, the fact, that one operator caused a tip over due to a bad parametrization during obstacle traversal shows that it is a non-trivial application to test the experience-based behavior adaptation.

The box plot of Figure 5.4 shows that the autonomous control using an individual operator knowledge base performs worst since the obstacle course cannot be passed. This is because every knowledge base has its own weakness, e.g., the knowledge base of `SSM_OutOp1` contains no evaluated behavior which sets the turn rate in slopes resulting in falling from the hill (obstacle 6 on Figure 4.5a), the knowledge base of `SSM_OutOp2` has no adequate leftwards turn behavior resulting in exceeding the map limits (at obstacle 2), and the knowledge bases of `SSM_OutOp3`, `SSM_OutOp4`, and `SSM_OutOp5` have no point turn behavior which leads to falling from the edge between hill and stairs (between obstacle 6 and 7). The lack of behaviors results from the requirement during the behavior generation process, which is that a behavior needs to stay constant for an entire step cycle to get evaluated. Thus, an operator who high-frequently tunes the control parameters allows less behavior evaluations which especially occurs during curves.

Merging all individual outdoor knowledge bases results in a knowledge base of higher maturity (`SSM_Out`) because the knowledge of the single knowledge bases is combined creating superior results with respect to every performance criterium. Energy efficiency is even better compared to manual control because there are less situations where the robot gets stuck due to an unsuitable parametrization. However, in only 60% of the runs, the obstacle course could be completed, because the comparably low path following capability led to falling from the edge on top of the hill. The variances of each run originated from slightly different start poses and different obstacle approaches that caused different behavior selections.

The knowledge base merged from the indoor libraries (`SSM_In`) lead to a control which works in principle even though it was trained on different obstacles but with similar feature characteristic. However, it contains no behavior to cope with the tight point turn on top of the hill (between obstacle 6 and 7) resulting in falling and not completing the course.

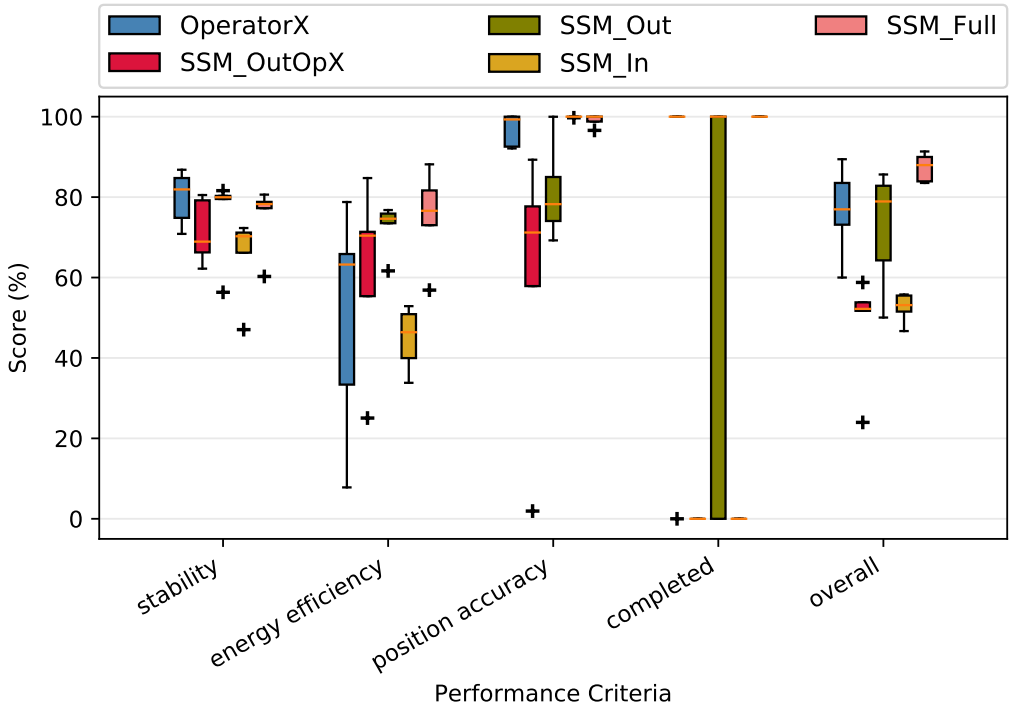


Figure 5.4: Score comparison of manual control (operator) and autonomous control with different knowledge bases - The data for manual control and for the autonomous control using single operator-specific knowledge bases represents every of the five runs, respectively. The data for each merged knowledge base represents five repetitions with the same knowledge base.

The autonomous control using the knowledge from all knowledge bases (`SSM_Full`) performed best. It combines the energy efficiency of the `SSM_Out` with the high position accuracy of the `SSM_In`. The latter in combination with the learned point turn behavior stored in the `SSM_Out` helps to complete the obstacle course reliably.

The scoring details during the traversal of the obstacle course (Figure 5.5) show through performance drops and higher variances that some passages are harder to overcome than others. For example, traversing the tracks, the random stepping field, and the bridge decreases stability and increased power consumption. This is because the selected traversal behaviors set large values for `body_shift_z` and `step_length_z`, i.e. using a high posture and high steps to overcome these obstacles. In addition, the overall movement speed is reduced through high values for `t_cycle` to allow a proper ground adaptation through the EDR. Nevertheless, the position accuracy is also decreased because the blind reactive motion control approach does not incorporate any environmental information based on exteroceptive

sensor data. Thus, hitting and reacting on an obstacle during traversal may lead to undesired orientation changes that are compensated by the trajectory follower within several steps. Climbing the hill and descending the stairs, on the one hand, also lower stability and power scores because the inclination reduces the horizontal projection of the support polygon and requires slower speeds to improve traction. On the other hand, a comparably high position accuracy is maintained due to less disturbances in comparison to rugged terrain. Traversing inclined planes also shows, that small path deviations are not directly compensated by setting appropriate turning angles with `turn_rate`. This is explainable by a lack of turning behaviors that were tested, evaluated, and stored in the underlying knowledge base and thus are not available for selection. The flat passages in between obstacles are traversed with higher position accuracy and stability, though curves are sometimes not followed precisely due to a lack of behaviors for specific turning angles. The straight passages, e.g. the gravel which has too small obstacles to be recognized by the proposed step height estimation, have a higher power consumption and less stability due to utilized tripod gait that is selected which comes with potentially higher speeds on the cost of a smaller support polygon.

The computational power to derive a behavior from the case base is roughly linearly increasing with the number of cases. The 591 evaluated behaviors of the `SSM_Full` knowledge base are analyzed within 55 ms by a 3 GHz CPU core, i.e. on average 0.09 ms per behavior. Loading the case base takes roughly 6 ms and the required disk space is 1.5 MB. Having a target update time of 1 s in mind, also larger knowledge bases can be utilized within real-time critical applications.

Conclusion

This experiment results reveal the importance of the utilized knowledge base. The lack of maneuverability shows, that a knowledge base needs a high variety of evaluated behaviors to cover the possible action space as good as possible. Limited interpolation and no extrapolation capabilities cause the requirement of many evaluated behaviors.

However, the merged knowledge base provides sufficient solutions for every combination of detected state context and commanded action context. The experience-based behavior adaptation was successfully applied to realize a fully autonomous control. The compact representation of the state context is sufficient to detect situations which require a behavior adaptation. Also different motion commands can be realized on flat ground, in stone fields, as well as in slopes. Consequently, an overall accurate, stable, and energy-efficient robot behavior is realized.

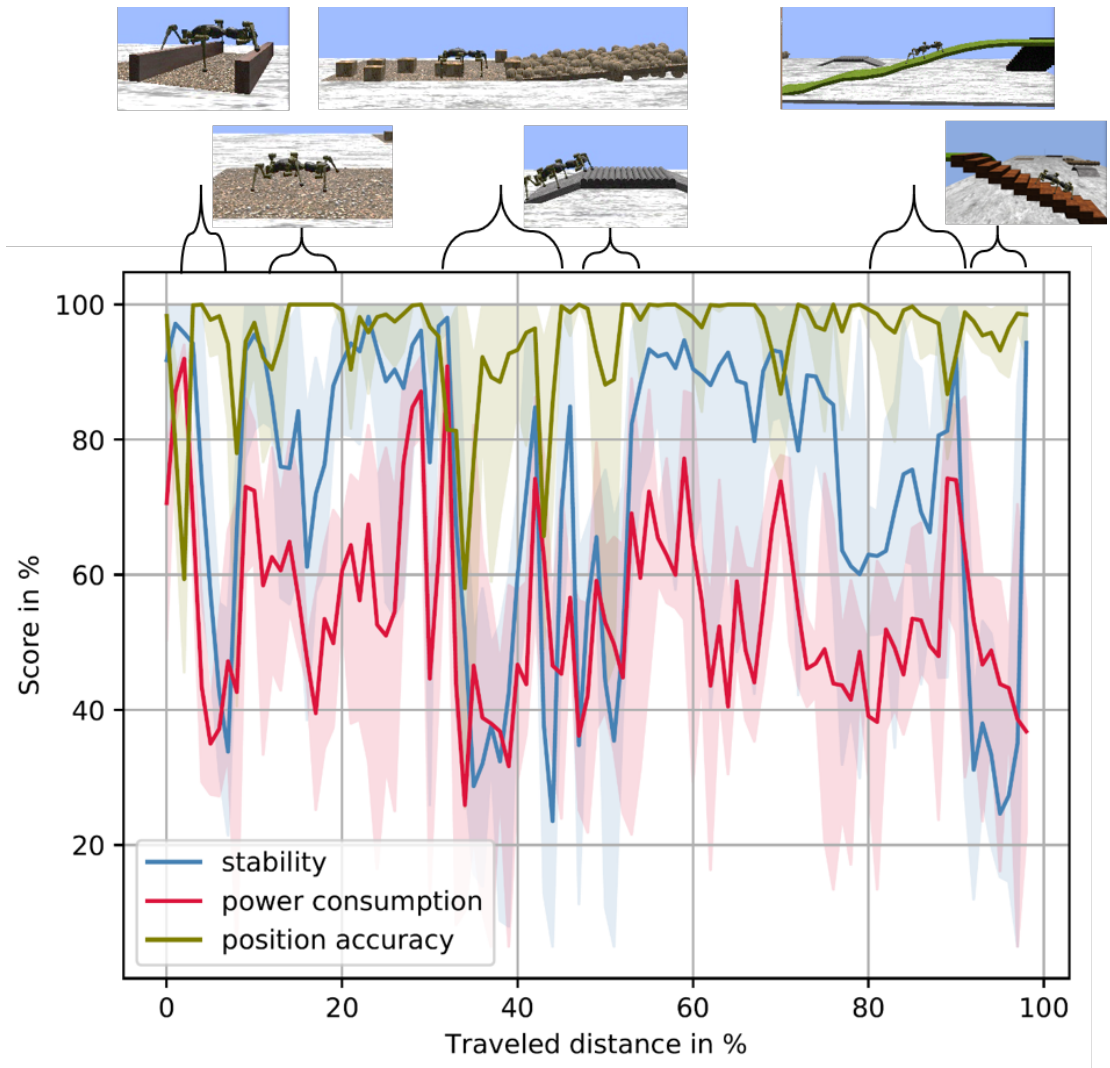


Figure 5.5: Scoring during obstacle course traversal - the line represents the average over all trials, the shaded area depicts the minimum and maximum values.

In the cases where the autonomous control of any partial knowledge base did not complete the course, missing maneuverability was the reason, which can be discovered beforehand by inspecting the behaviors stored in the knowledge base. In addition, as shown through merging of knowledge bases, missing experiences can be added and incorporated later on without extra coding effort. In contrast, the failure reason during manual control was a bad parametrization causing the robot to tip over. The latter reason is serious and may occur more often in situations where the operator's task load is very high.

5.3.2 Influence of Case-Based Behavior Merging

This experiment compares the influence of merging the supposedly best behaviors within the case adaptation step that is defined by Equation (5.7).

Experimental Setup

The setup of the previous experiment was taken, except that only `SSM_Full` was evaluated. The number of behaviors that were used to emerge new behaviors was set to: $K = [1, 3, 5, 7]$. Because less variation in the data was expected for this and the upcoming experiments, the normalization intervals of efficiency was set to [0.45, 0.9] $\frac{Wh}{m}$ and of stability to [22, 11]° indicating a score of 100% and 0%, respectively.

For every K, the track was executed five times and the evaluation criteria for position accuracy, energy efficiency, and stability were continuously recorded.

Observations

The results (Figure 5.6) show, that the simple selection of the supposedly best behavior produces least performance and most variance between the trials. Executing a behavior that is merged from the best behaviors improves stability, energy efficiency, and accuracy. This effect increases until $K = 5$. Larger values especially degrade the position accuracy. This is because especially behaviors at the edge of the action space are effected by behaviors from only one side of the action space, e.g. although a point turn behavior is among the best behaviors when a point turn is desired, additional behaviors that have additional translational components might effect the overall solution negatively.

Conclusion

The experiment shows that merging the three to five best behaviors is advantageous because it smoothes the behavior adaptation output and reduces the influence of noise-inflicted anecdotal evidence in the data set. The merge of more behaviors reduces the adaptation flexibility. The exact number for K is depending on the provided data set. The smaller the data set, the more influence has K which can generate undesired behaviors. In any case, the execution and evaluation of new behaviors that results from a merge lead to knowledge gain that can also positively influence upcoming behavior adaptations.

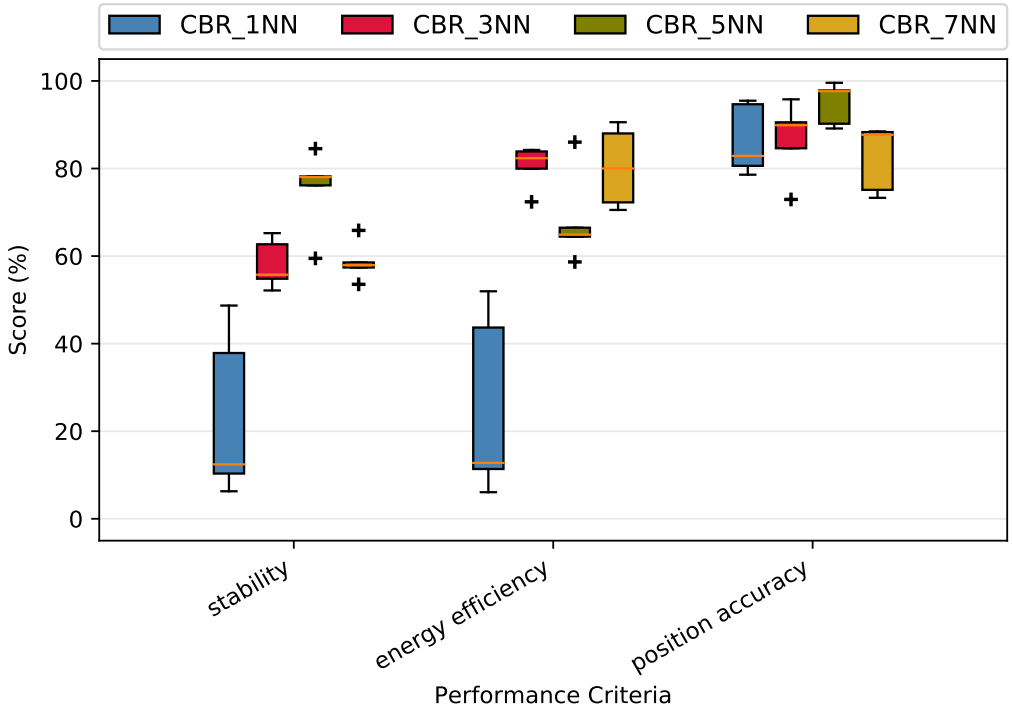
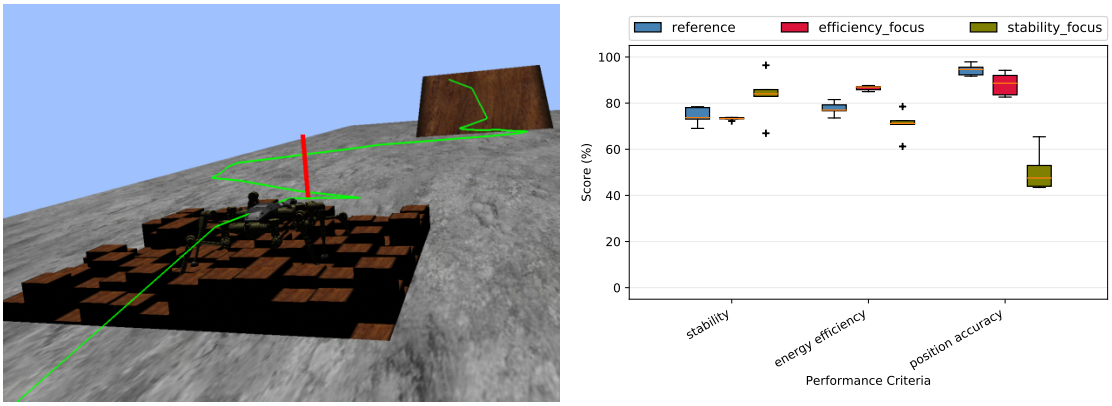


Figure 5.6: Score comparison of with respect to behavior merging



(a) evaluated test track (b) score comparison

Figure 5.7: Evaluation of different performance prioritizations

5.3.3 Influence of Performance Prioritization Changes

Changing the performance prioritization offers the possibility to influence the autonomous behavior selection in order to fulfill current needs that are not modeled within the applications' inputs or outputs. The objective of this experiment is to exemplarily show the influence of changed performance prioritizations.

Experimental Setup

As in previous experiments, SpaceClimber had to follow a given path over obstacles. In order to accelerate the evaluation process, only a part of the artificial indoor obstacle course was utilized (Figure 5.7a), i.e. the random stepping field and the ramp with an inclination of 35°. K was set to five and the reference performance prioritization was set to as stated in Table 5.2. For this experiment, an energy-efficient parameterizations was chosen by setting w_{epd} from 0.1 to 1.0. In addition, based on the reference prioritization, a stability-prioritizing weighting was derived by setting w_{ssm} from 0.1 to 1.0. The prioritization set was applied five times and the autonomous traversal was evaluated in terms of stability, energy-efficiency, and accuracy.

Observations

As depicted in Figure 5.7b, changing the performance prioritization favors one performance attribute on the costs of other. The efficiency-prioritizing weighting improves the energy efficiency score by ca. 10% but degrades the position accuracy by 5%. Increasing the stability prioritization also increases the stability score of around 10%. Unfortunately, favoring more stable behaviors neglects some motion command-related and efficient behaviors resulting in an energy efficiency drop of around 15% while the position accuracy also degrades by 45%.

Conclusion
By changing the performance prioritization, different behaviors are selected that favor specific performance criteria. This way, influence on the autonomous behavior adaptation can be taken in order to configure it for a specific application. The possibility to adapt the prioritization during runtime allows to configure the behavior derivation to changing needs that are not modeled as inputs or outputs.

One can argue that the manual adaptation of behavior parameters is now replaced by adapting the performance prioritization and consequently no benefit is gained. Nevertheless, this replacement has several advantages: (1) the performance prioritization does not have to be varied continuously, (2) it has usually less dimensions, (3) the operator need's to care about application-specific parameters and not about implementation-specific behavior parameters that he or she might not understand due to missing expert knowledge. Consequently, the setting up of performance prioritizations is self-explaining and intuitive solutions can easily be derived for specific use cases. And (4), the danger of catastrophic parameterizations is quite low as prioritizing one performance feature, e.g. energy per distance, simultaneously requires a stable walking behavior even if static or dynamic stability is not prioritized.

5.3.4 Learning Capability

The goal of this experiment is to analyze, how well the case-based knowledge representation is suited when important experiences are missing. In this particular experiment, this is analyzed by providing a knowledge base of only good performing behaviors for traversed state contexts without having any non-working behaviors included.

Experimental Setup
For this experiment, `SSM_Exp` was used as knowledge base. It was generated by recording an experienced operator who manually controlled SpaceClimber over a variety of obstacles. Due to its expertise, only well-working behaviors were executed and evaluated. Thus, the knowledge base contains only positive experience and behaviors that might not work in a certain context are missing. The task was now to autonomously overcome the railway tracks of the artificial outdoor course (which were also traversed during the knowledge base generation but differently approached, thus also being slightly differently represented in the case base) with using the experienced-based behavior adaptation and a trajectory follower to generate proper motion commands. The standard performance prioritization was used and $K = 1$, i.e. the most similar behavior was selected instead of merging the best behaviors.

SpaceClimber had to move from a fixed start to a fixed end point and thereby had to cross the railway tracks. By doing so, the behavior evaluation and recording was activated to extend the initial knowledge base and to update the corresponding case base. After the final target position behind the tracks was reached, the corresponding training session ended triggering the subsequent training session. This was repeated 12 times. The acquired experiences of each training session were directly integrated in the case base to provide further knowledge within each session for all the following. In order avoid continuous high-frequent parameter adaptations that prevent behaviors from being evaluated, a new behavior adaptation was only allowed after a behavior was evaluated (and not immediately when context changes appeared).

Observations

Figure 5.8 shows the first six trials - the time needed to overcome the railway, the number of evaluations, and how many context evaluations were added. The latter only considers evaluations where a behavior was evaluated in a new context excluding behavior performance updates of already evaluated contexts. Thus, a higher number indicates that a behavior was selected for execution despite the fact that it was not evaluated before in the specific context. In addition, the average state context similarity over the finally executed behaviors is plotted. A low value indicates the selection of behaviors that must fit well to the provided action context with respect to the selected performance prioritization but were not evaluated in the current context, yet.

In the first trial, SpaceClimber was initially not able to overcome the tracks. Though obstacle-climbing behaviors were available in the case base, non of them had a large case similarity s_c because they were comparably slow, inefficient, and instable. Instead, fast, efficient, and stable behaviors for plain ground were selected, because their action context similarity s_a was large enough to compensate the lower state context similarity s_z where mainly the maximum obstacle height `obstacle_height` differed to the evaluated conditions. Consequently, SpaceClimber remained in the low, stable, and energy efficient posture and constantly hit the obstacle. Of course, with doing so, the chosen behaviors got an additional context evaluation for the current context. This increased their state context similarity but lowered their action performance similarity dramatically which led to new selected and executed behaviors. 23 new context evaluations were needed in order to realize, that none of the behaviors for flat ground are well suited to overcome high obstacles. Only then, an obstacle behavior was selected which finally enabled SpaceClimber to cross the rails. It took about 220 s to move from the start to the target position.

The second training session started with the knowledge at the end of the previous session. Now, SpaceClimber needed around 95 s less to overcome the rails. Still, some

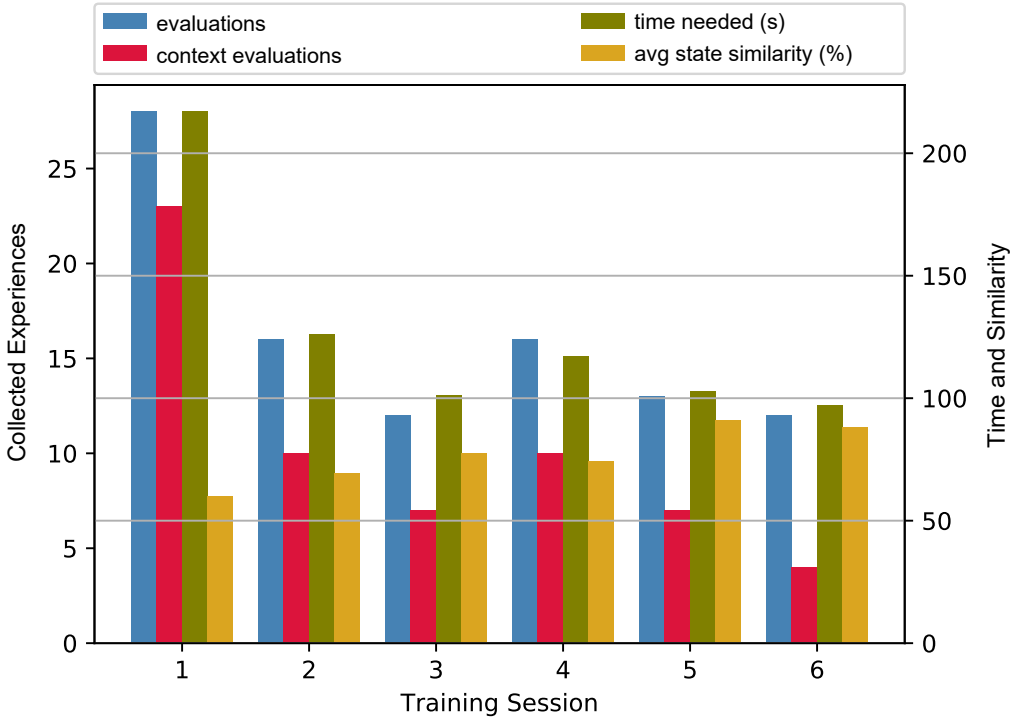


Figure 5.8: Collected experiences per training session, average similarity of detected state context and resulting time to overcome tracks.

behaviors for flat ground were selected first and additional context evaluations were needed to know that these behaviors do not perform well when high obstacles are in the path. They might not have been evaluated in the previous run, because the approaching angle towards the rails always varies causing different motion commands and different views on the obstacle with slightly changed state context estimations. Nevertheless, less behaviors had to be evaluated since the number of new context evaluations decreased from 23 to 10.

The time to cross the obstacle decreases with further runs until a minimum time of ca. 95 s is reached. The number of evaluation converges to twelve evaluations per run, which reflects the minimum amount of evaluated step cycles to reach the target. In the long run, the number of new context evaluations approaches zero. Nevertheless, there are still some trials in between, where the obstacle is approached differently causing different context inputs and potentially again the execution of a flat behavior that was not tested on obstacles before. In such a case, the trial's number of context evaluations increases again, e.g. as also seen in the 4th trial. However, the increasing values of the average state context similarity show that less unexpected situations occur leading to a more similar and reliable behavior adaptations per run.

Conclusion

This experiment shows the adaptation capability of the case base and the change of selected behaviors for a certain context. In addition, it underlines the usage of a mature knowledge base. Despite well-performing behaviors, also "bad" experiences are vital to mark behaviors that do not perform well in certain situations. In order to generate bad experiences, using an accurate simulation to provide evaluations in critical situations is mandatory as long as the physical system is not robust enough to withstand the consequences of failure producing behaviors. The proposed automated knowledge base generation (Section 4.4.2) can help to efficiently evaluate existing or new behaviors in manifold situations.

Even though the case base quickly integrates new experiences which constantly changes the derived behavior, if fundamentally new information are added, the overall case base adaptation process requires many updates because the cases are not interconnected. Especially when multiple similar behaviors are present, it will take time to iteratively reevaluate all of them before diverging to another type of behavior because the update of a behavior has no influence on the behavioral neighborhood. Having a correlation between behaviors would probably require less experiences in order to switch between completely different behavior parameterizations when new divergent situations occur, e.g. state contexts that were not visited before or discrete changes of the robot state like a broken joint.

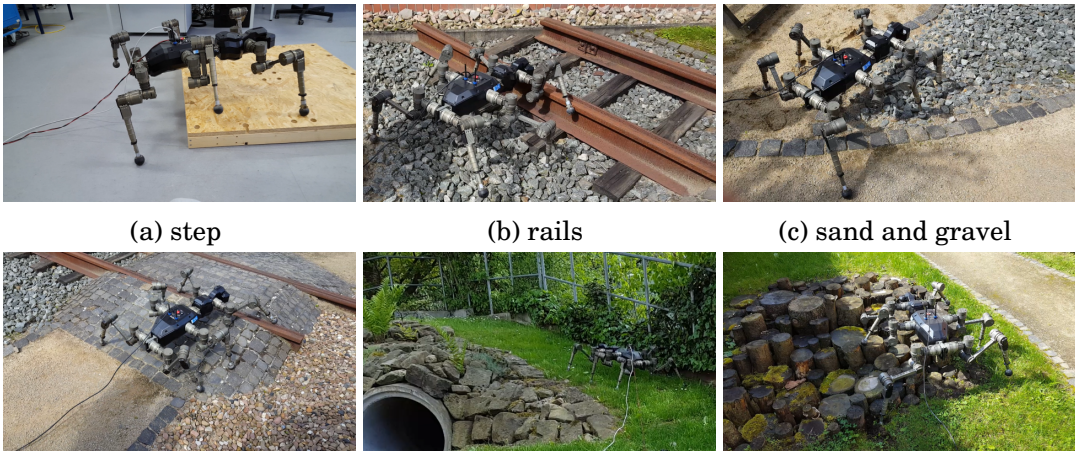
5.3.5 Outdoor Validation with SpaceClimber

The goal of this experiment is to analyze the application of the experience-based behavior adaptation on a physical robot. Both, the transferability of an in simulation generated knowledge base as well as the influence of a real world scenario on the behavior adaptation are investigated.

Experimental Setup

The final knowledge base of the previous experiment was applied on SpaceClimber and tested on different surfaces and obstacles (Figure 5.9): hard plain ground, sand, gras, gravel, steps, rails, and slopes. In order to test the experience-based behavior adaptation on the real system, nothing had to be changed from the perspective of the control architecture, except that the simulation was replaced by the real hardware drivers. The performance prioritization was again chosen as stated in Table 5.2.

A semi-autonomous control was used to operate SpaceClimber, i.e. the motion command was manually set to navigate SpaceClimber over the obstacles whereas the behavior adaptation was performed autonomously based on the commanded action context and estimated state context. The latter was derived from map information,



(a) step (b) rails (c) sand and gravel

(d) slope (e) grass hill (f) stepping field

Figure 5.9: Example obstacles for behavior adaptation experiments

as done in the simulation experiments. No further experiences were collected during the execution in order to provide the same knowledge base for all obstacle types.

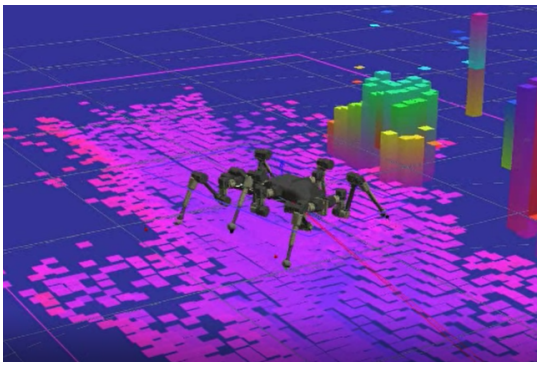
Observations

Even though having crossed several obstacle types, in the following, the control over flat outdoor terrain and high grass bundles is described in more detail whereas the other trials are summarized more briefly afterwards.

Before crossing the grass bundles, SpaceClimber was navigated over comparably flat terrain as shown in Figure 5.10a. The small irregularities in this terrain like small stones, the path boundary, or gras perish in the sensor noise and cannot be recognized in the created map of the environment (Figure 5.10b). Thus, while traversing this flat passage (first 80 s of Figure 5.11a), the detected obstacle height `obstacle_height` has some noise around zero. Since the average value did not exceed the threshold to reach the next discretized state context feature bin for `obstacle_height` (which is 7 cm due to the chosen normalization limits and discretization steps), all state context features including `slope_x`, `slope_y`, and `roughness` are zero. Consequently, stable and efficient walking behaviors for a flat plane are selected: low `body_shift_z` and `step_length_z` among other behavior parameters. As soon as SpaceClimber approaches the grass bundles, larger obstacles enter the ROI and `obstacle_height` is estimated to almost 40 cm. This triggers an immediate behavior adaptation, i.e. that `body_shift_z` and `step_length_z` are increased simultaneously among others. These climbing walking patterns are kept as long as SpaceClimber traverses the gras bundles (between $t = 80\,s$ and $t = 215\,s$ in Figure 5.11, Figure 5.10c and Figure 5.10d). Only few behavior adaptations are done according to different obstacle height measurements. As soon as no major obstacles

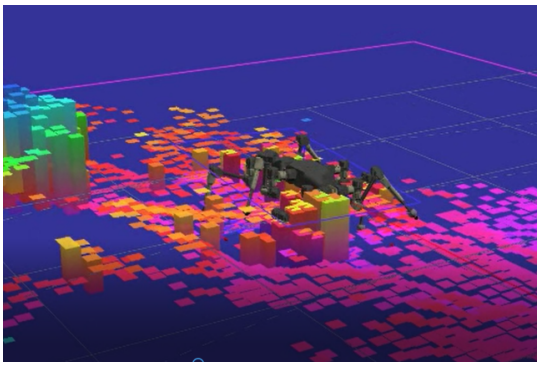
are detected within the ROI, the walking behavior is again strongly adapted to plane walking behavior types (Figure 5.10e and Figure 5.10f).

(a) initial flat terrain



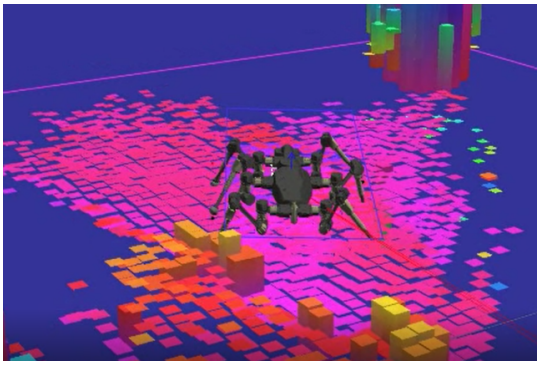
(b) map of initial flat terrain

(c) traversing high grass bundles



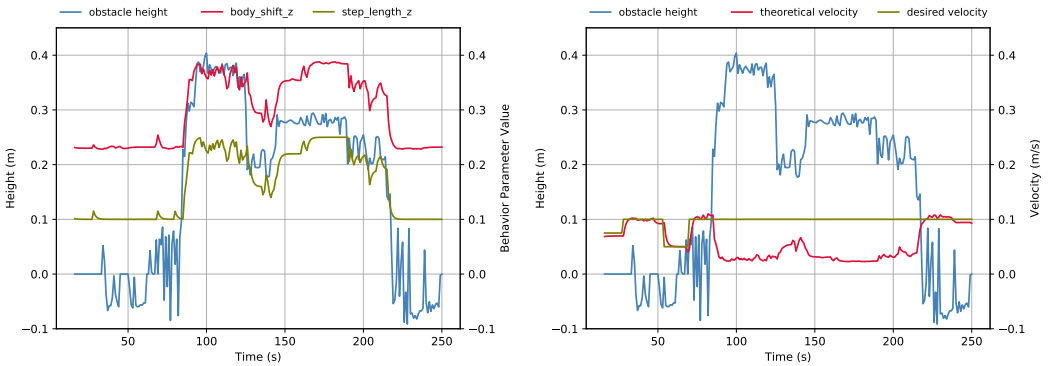
(d) map of high gras bundles

(e) reentering flat terrain



(f) map of after overcoming the gras bundles

Figure 5.10: SpaceClimber traversing the outdoor test course of the DFKI



(a) Selection of adapted posture parameters

(b) Chosen theoretical and commanded velocity

Figure 5.11: Applied parameter changes and detected obstacle height during traversal of the outdoor test course of the DFKI

Besides the visible postural adaptations, the velocity of the robot varied through the adaptation of `step_length_x`, `t_cycle`, `t_shift`, and `turn_rate`. Based on these parameters, the theoretical velocity is computed according to Equation (3.9) and the longitudinal component plotted in Figure 5.11b together with the commanded longitudinal velocity. The plot shows that commanded and theoretical velocity match very good during the flat terrain. Especially up to $t = 75\,s$, one can see the parameter blending between behaviors of few seconds when the desired velocity was decreased and increased stepwise by 50 $\frac{mm}{s}$. However, as soon as higher obstacles are detected, behaviors with less theoretical velocity than the commanded one are chosen. Since, the performance prioritization on longitudinal velocity is relative high ($w_{vel_x} = 0.8$), obviously no behaviors are stored in the knowledge base that can realize the commanded speed over obstacles of this height. As indicated in the phase between $t = 125\,s$ and $t = 130\,s$, a little less detected obstacle height directly causes to speed up a little. After returning to flat terrain, behaviors with larger speed are selected that again meet the commanded velocity.

A similar adaptation strategy can be seen when traversing other step-like obstacles, e.g. the wooden platform of 10 cm (Figure 5.9a), or rails (Figure 5.9b). Unfortunately, the rails could not been traversed due to the fact that only experiences of behaviors with a maximum `step_length_z` of 22 cm were stored in the knowledge base which were sufficient to overcome the simulated rails of 20 cm height. But due to small sinkage into the gravel before the rails and light tilting to the front, these behaviors did not succeed to overcome the real rails. Less demanding, flat gravel and sandy surfaces (Figure 5.9c) are less problematic due to the adaptation capabilities of the motion control itself.

Selected behaviors for slopes usually cause a body shift towards the slope due to improved stability characteristics, which worked well on rigid slopes (Figure 5.9d). In contrast, climbing the grass hill (Figure 5.9e) did not succeed because the noisy surface structure caused the detection of fake obstacles that triggered the selection of behaviors with high `body_shift_z` which finally produced instable and wobbling walking behaviors on the slope. The hard surface on the random stepping field was traversed until SpaceClimber got trapped with the front left leg in one of the holes between the wooden pillars (Figure 5.9f).

Conclusion

The overall experience-based behavior adaptation approach could successfully been applied on SpaceClimber. The walking behavior is plausibly adapted with respect to terrain type and desired motion command. Thus, the operator can stay focussed on the navigational task without taking care of walking pattern or postural constraints. The fact that during flat areas the desired motion command is followed whereas during obstacle traversal slower walking patterns are autonomously selected, shows an additional advantage. The robot decides based on its experiences how to get the maximum performance. Impossible action commands are dismissed and only working solutions are applied. If the operator knows more performing solutions, he or she has still the chance to manually set the behavior parameters and let them being evaluated during execution in order to extend the current knowledge base.

The knowledge base that was generated in simulation is applicable in real world scenarios on moderate terrain because an appropriate behavior in combination with the reactive control approach generate motions that cope with a range of state contexts. In challenging situations, i.e. close to the limits of the robot's mobility, the demand of a perfect parametrization is higher. But additional noise and the simulation reality gap of the knowledge base, especially when leaving hard underground, lead to near optimal solutions that finally prevent a successful traversal. However, further reevaluations of existing behaviors with the real system will iteratively update the utilized case base and increase its competence.

Situations like getting stuck with a foot, e.g. while traversing the stepping field, cannot be solved by the experience-based behavior adaptation because the underlying reactive motion control cannot handle situations like this. A deliberative control approach is needed to plan a trajectory for the trapped leg to release it. However, the large performance discrepancy between the current behavior evaluation with respect to the respective case of the case base, can be one potential signal to trigger the control architecture change.

5.4 Conclusions

This chapter introduces a case-based behavior inference approach as behavior configurator within the experience-based behavior adaptation architecture. For the reasoning process, a case base is generated from the experiences of a knowledge base which is a collection of behaviors, their evaluations in different state contexts, and a list of visited, discretized state context classes. Every behavior evaluation, or case, represents possible solutions, i.e. the performance of a behavior, for specific state contexts.

In order to rate the applicability of behaviors, their similarity to the desired action context and to the current state context is computed. Behaviors that proved to provide the currently desired action in a similar state context are potential candidates for execution. Either the supposedly best behavior is selected or a weighted merge of the corresponding parameters of the best behaviors can be initiated to generate a new behavior that is being applied. In any case, the integration of the evaluation results of executed behaviors continuously updates the knowledge and case base improving its competence.

The experimental evaluation within the locomotion scenario of controlling SpaceClimber shows the applicability and real-time capability of this approach. During the traversal of different obstacles with varying motion commands, SpaceClimber autonomously adapts its walking pattern to maximize its performance. Thus, the chosen state context and performance features for the locomotion use case are useful to characterize the environment as well as the walking pattern. Posture and walking parameters are autonomously and simultaneously adapted which facilitates the manual control of such a complex system and opens the path towards fully-autonomous control.

The experiments with different knowledge bases indicate that more experiences improve the flexibility and performance of the control. The performance prioritization can be changed online to influence the adaptation process in order to focus on mission related needs. The possibility to merge potentially applicable behaviors generates new behaviors that may combine positive aspects and cause less parameter variations.

The case base is capable to adapt to new state contexts and long-term performance degradation through continuously updating executed behaviors. The computational costs nearly linearly increase with the number of cases which scales up to thousands of cases for real-time applications. The independency from the behavior's dimensionality promotes this approach for knowledge bases with many dimensions but where the behavior space is sparsely covered. Despite fast experience integration into the case base, the learning process in terms of updating previous knowledge is relatively

slow since no correlations between cases are considered, i.e. every single case needs to be updated several times. Even though the experiments showed that experiences from simulation are transferable to the real system, the simulation reality gap leads to different performance values which makes a reevaluation of existing cases necessary in order to maintain a stable and efficient locomotion. Thus, further research on intelligent case base update schemes and memory clean up can be done. Using a model-based approach is an alternative that is investigated in the following chapter.

Chapter 6

Model-Based Behavior Inference

A behavior needs to be inferred for the current context ($\boldsymbol{z}$, $\boldsymbol{a}$, $\boldsymbol{w}$) based on a knowledge base, i.e. a collection of past experiences ($D = \{\boldsymbol{b}_s, \boldsymbol{z}_s, \boldsymbol{p}_s\}^S$). A robot that continuously evaluates its behavior throughout its lifetime will inevitably acquire new experiences that are worth storing. Thus, large knowledge bases will be generated. Despite their advantages, lazy learning algorithm like CBR, get inefficient for large knowledge bases due their knowledge interpretation at runtime. In contrast, eager learning algorithms compile their knowledge offline to a representative model that can be used online with rather low computational costs to derive solutions. Also, interpolating and extrapolating capabilities are possible. However, the reliability of model-based solutions is highly depending on the utilized algorithm as well as on the provided data in terms of quality, such as noise level, and quantity in terms of data space coverage. Unfortunately, real robotic applications usually face many input and output variables as well as incomplete and inaccurate information, which is also the case for the autonomous behavior adaptation within this work. Consequently, a sophisticated approach is needed to handle these challenges while utilizing the advantages of eager learning algorithms.

This chapter presents a model-based behavior inference strategy to autonomously adapt the behavior of a robot for changing contexts in real time. First, its concept is described in detail. The approach utilizes a regression model to predict the performance of a behavior for a given state context within an optimization loop. The resulting requirements on the prediction model demand an accurate and fast regression method. Thus, an extension to available GPR implementations was developed, which is presented in Section 6.2. After describing this new regression algorithm, an experimental evaluation on different data sets is provided in Section 6.3. Finally, this chapter closes with a conclusion.

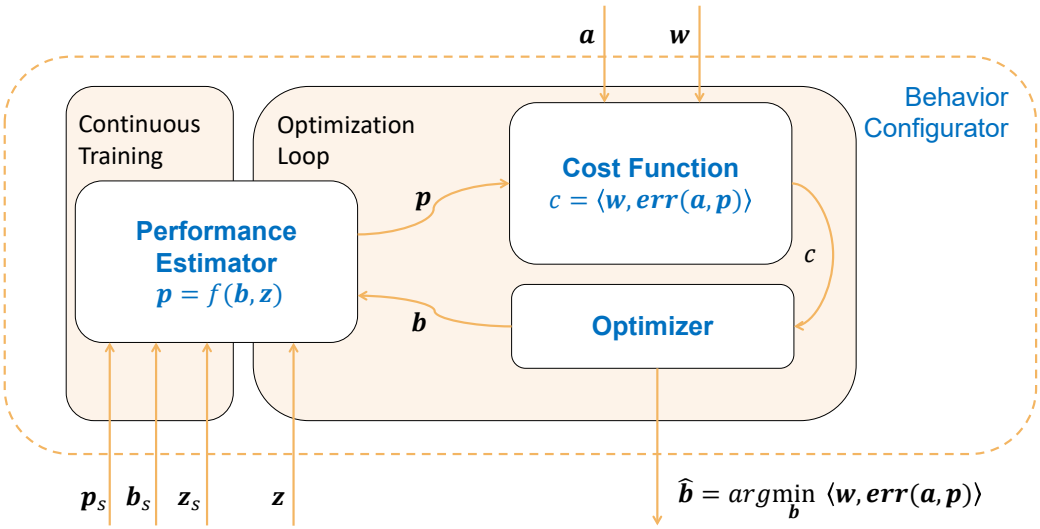


Figure 6.1: Concept of the model-based behavior inference approach

6.1 Optimization Based on Performance Estimation

Various learning from demonstration methods are usually suitable for acquiring a model based on provided state-action pairs (Section 2.2). In the context of this work, using a model to predict the supposedly best behavior $\hat{b}$ for the given inputs (z, a, w) is not sufficient. Besides, data that is available in the knowledge base like state context z and performance features p that correspond to the desired action a, no weighting information w is available. Therefore, in contrast to other application fields, no direct mapping can be generated. Instead, the control scheme that is depicted in Figure 6.1 is proposed.

Based on the experiences stored in the knowledge base in form of behavior b_s, state context z_s and performance samples p_s, a model is trained that predicts the performance p for a behavior b in a given state context z, i.e. it estimates the mapping from Equation (1.2). This performance estimation is used to evaluate if a behavior is suited for the actual action context, i.e. desired action with performance prioritization, according to a cost function c. Based on the costs, an optimizer can then effectively generate new test behaviors whose behavior performance is estimated again. This optimization loop is triggered as soon as desired action, performance prioritization, or state context change. It finishes, when the costs converge to a minimum value indicating that the supposedly best behavior $\hat{b}$ is found. Otherwise, if a maximum optimization time is given and finally expired, the behavior with the minimum costs so far, is selected for execution.

Selecting an optimizer that converges fast and is simultaneously robust to local minima is desired to find the supposedly best behavior as fast as possible. In order to be independent from the performance prediction model, a black-box optimizer is chosen. Within this work, two optimizers were selected and tested:

1. Evolutionary strategies, inspired by natural evolution, is an optimization technique that tests a set of candidate solutions, also called individuals each representing a real-valued parameter vector, and evaluates their fitness according to a cost function. Based on the results of this so-called generation, a new generation of new individuals is generated with the use of mutation, recombination, or selection in order to evolve iteratively better solutions. The popular implementation Covariance Matrix Adaptation using Evolutionary Strategy (CMA-ES) [Hansen and Ostermeier, 2001] adds two features to accelerate the optimization process. First, the covariance matrix is computed and used to guide the mutation direction. Second, a cumulative step width calculation is used to regulate the mutation scale, i.e. adjusting the magnitude of the parameter variations with respect to the magnitude of fitness variations.
2. Particle Swarm Optimization (PSO) [Eberhart and Kennedy, 1995] is an optimization strategy inspired by the behavior of a swarm of birds. Thus, candidate solutions are represented as particles having a position and velocity, representing the adaptation speed and direction. These particles explore the search space according to their velocity. After every evaluation of all particles according to a cost function, the velocity is influenced by a constant damping, the attraction towards the best parameters of a particle, and towards the best parameters of the whole swarm. The result is a guided search that is robust to local optima and suitable for unstructured and time-dependent fitness landscapes.

The utilized performance features have different units and scales that make them incomparable. Thus, they are normalized between [0, 1] according to robot and application-specific limits. The estimated, normalized performance values of the testing behavior are then compared feature-wise to the desired action with an error function $\boldsymbol{err}()$, e.g. absolute error or squared error. Finally, a representative cost value c is calculated by building the scalar product between the resulting error vector and the performance prioritization $\boldsymbol{w}$

$$c = \boldsymbol{w}^T \boldsymbol{err}(\boldsymbol{a}, \boldsymbol{p}), \tag{6.1}$$

which is sent to the parameter optimizer. Depending on the optimizer, the cost value is normalized by the sum of weights and subtracted from one to obtain a normalized fitness value that is to be maximized.

The model of the performance estimator can be seen as an internal simulator that is frequently triggered within an optimization loop to find an optimal solution before its actual application. A quick result is desired in robotic applications, e.g. if the experience-based behavior adaptation is used to select a planner configuration that finds a solution with a minimal planning time but the optimization itself takes already several seconds, its benefit will easily be lost. Consequently, different requirements arise that needs to be considered when selecting a proper approach for the performance estimation.

1. The model inputs consist of the adaptable behavior parameters and the application-specific state context features. The outputs are defined by the action-dependent performance features. Both, inputs and outputs, are continuous real numbers that demand a data-driven regression approach.

2. It is expected that a model prediction is needed many times before the optimization algorithm converges to an optimum. Thus, the model's prediction time should be low to allow the usage of experience-based adaptation in time-critical applications.

3. It is envisaged, that a robot learns over its life time, i.e the training phase of the model is only partially completed before its execution. Consequently, an incremental learning approach is needed, that can deal with an unlimited stream of input data while maintaining an appropriate training time and without rejecting samples after a certain amount of time, e.g. when the model size is limited and full. The model has to reflect the fact, that on the one hand, old experiences might loose in relevance with respect to new ones. On the other hand, old experiences may provide important information that should not been forgotten completely.

4. The additional number of hyperparameters is also a criterium to select a proper regression approach. Only few parameters that are valid for a large range of applications are acceptable to avoid much expert knowledge when setting up the experience-based behavior adaptation. Otherwise, adapting several behavior parameters would be replaced by adapting several model parameters.

As stated in Section 2.2, huge effort were made to develop incremental learning variants of powerful regression techniques. With the help of approximations and intelligent subset selections, online version are available. Nevertheless, non of the mentioned approaches, perfectly match all requirements of a fast and reliable performance prediction for continuously gathered data throughout the life-time of a robot. Thus, a new approach was derived that incorporates and extends ideas of existent approaches as described in detail in Section 6.2.

6.2 Stream Online Gaussian Process Regression

In the proposed model-based behavior adaptation approach, a model is needed that predicts the performance of a behavior in a given state context with the help of collected experiences. Within this thesis, every input vector $\boldsymbol{x}$ consists of behavior parameters $\boldsymbol{b}$ and state context features $\boldsymbol{z}$, thus $\boldsymbol{x} \in \mathbb{R}^{B+Z}$.

Real-time online model learning within the scope of model-based behavior adaptation poses four major challenges:

1. Noisy samples in from of multidimensional input and output feature vectors are expected.
2. The prediction process needs to be sufficiently fast for being called within an optimization loop.
3. The learning system needs to deal with unlimited training data while preserving a limited update time.
4. The model has to continuously adapt to new training data as input-output relations might change over the lifetime of a robot.

To cope with this demanding challenges, an incremental online GPR with limited computation time, called Stream Online Gaussian Process Regression (SOGPR), was developed within this thesis. The underlying principle to maintain a constant training and prediction time is the limitation of the kernel matrix size. In addition, efficient training methods are used to lower the computational complexity. However, the prediction step for a query input including the uncertainty estimation are analog to standard GPR (Section 2.2), except that the utilized RBF kernel already incorporates the expected output noise:

$$k(\boldsymbol{x}_m, \boldsymbol{x}_n) = \sigma_f^2 \exp\left(-\frac{||\boldsymbol{x}_m - \boldsymbol{x}_n||^2}{2l^2}\right) + \delta_{x_m x_n}\sigma_n^2, \tag{6.2}$$

where $\delta_{x_m x_n}$ is the Dirac function of $\boldsymbol{x}_m$, and $\boldsymbol{x}_n$ which means that the noise is only applied if $m = n$. This facilitates the implementation and reduces Equation (2.8) to

$$p_* = y_* = \mu_* = \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{y} \tag{6.3}$$

to estimate the performance of a feature p_* for a specific behavior parameter and state context combination $\boldsymbol{x}_*$. The uncertainty for the performance estimation u_* is based on Equation (2.10) but facilitates to

$$u_* = \sigma_*^2 = k_{**} - \boldsymbol{k}_*^T \boldsymbol{K}^{-1} \boldsymbol{k}_*. \tag{6.4}$$

Using any GPR model as performance estimator within the overall behavior optimization loop comes with the advantage of uncertainty information for every prediction. In order to avoid the execution of curious potentially harmful behaviors, the uncertainty vector $\boldsymbol{u}$ that reflects the uncertainty for every performance feature can be used within the optimization loop to punish behaviors that might perform well but have no evidence in the data that was used for training. Thus, Equation (6.1) is extended to

$$c = \boldsymbol{w}^T \boldsymbol{err}(\boldsymbol{a}, \boldsymbol{p}) + \boldsymbol{u} \tag{6.5}$$

when using SOGPR, where $\boldsymbol{p}$ contains the estimations of all performance features.

The training process of a limited kernel matrix for SOGPR is different to standard GPR in order to obtain a constantly low training time even for unlimited input samples. The maximum kernel matrix size M must be chosen according to the application. A small kernel size limit guarantees fast training and prediction times. But higher kernel size limits might be considered for complex mappings in high-dimensional data spaces. However, the n kernel elements with $n <= M$ and $n \ll S$ need to reflect the entire knowledge base and must be carefully selected. Thus, as depicted in Figure 6.2, the process of training the Gaussian process model involves several steps. First, the importance of the incoming sample ($\boldsymbol{x}_s$, y_s) needs to be evaluated. Second, depending on the sample's importance, two different update schemes are proposed, *sample learning* and *case learning*. In the following, further details on the single steps are provided. Additionally, the utilized hyperparameter strategy is presented to reduce the effort to tune hyperparameters manually.

6.2.1 Sample Importance

When a new sample is incoming, one has to decide whether it is worth storing it in the limited kernel matrix or not. Thus, to estimate the samples importance δ_s, several metrics are possible. Two metrics are implemented to evaluate their pros and cons: *distance-based* and *variance-based*, which can be selected by $type_\delta$. The distance-based metric is the minimum Euclidean distance between $\boldsymbol{x}_s$ and all kernel elements

$$\delta_s = min\left(||\boldsymbol{x}_s - \boldsymbol{x}||^2\right), \forall \boldsymbol{x} \in \boldsymbol{X}. \tag{6.6}$$

The variance-based approach, proposed by [Nguyen-Tuong and Peters, 2011], uses a metric that provides the distance to the current entries of the kernel matrix while also considering its independence, which is in fact the uncertainty according to Equation (6.4), computed for $\boldsymbol{x}_s$ instead of $\boldsymbol{x}_*$.

Regardless of the chosen metric, if this value for the current sample exceeds the threshold δ_{min}, the prediction capability of the model at this point is low and, and

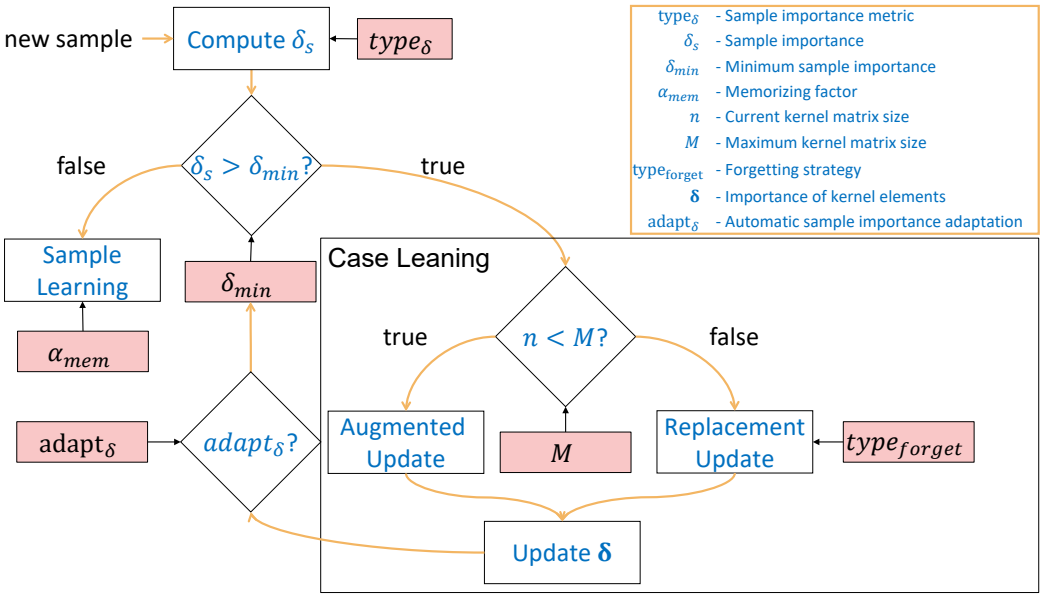


Figure 6.2: Flowchart of the SOGPR training progress. The red boxes indicate hyperparameters to tune the algorithm for different applications.

consequently, it is beneficial to integrate the sample in the current kernel matrix. By adding a new kernel element, the predicting capability of the model in the corresponding state space region is improved. This is called case learning, being similar to inserting a representative case in the case base in CBR. If the importance of the incoming sample is equal or below δ_{min}, sample learning is triggered.

6.2.2 Sample Learning

Sample learning actually deals with the challenges that come with infinite stream data. Two scenarios are possible where sample learning is crucial. First, a robot that acts in a fixed context and constantly evaluates similar behaviors. Under this circumstance, on the one hand, integrating all samples as kernel matrix entries is less beneficial, because more similar samples do not provide more information. On the other hand, multiple evaluations of same inputs are useful and should not be discarded, especially when noisy measurements are expected. Second, considering a long-term autonomous system, the limited kernel matrix will sooner or later be filled and the representative kernel elements will more or less evenly characterize the state space. Thus, most incoming data will be similar to the representative kernel elements resulting in an importance measure δ_s below the threshold δ_{min}. Nevertheless, the recent samples characterize the actual behavior the best and may contain a change between the input-output mapping that occurred over a long period of time.

Consequently, instead of discarding the redundant samples, the following data incorporation strategy is proposed. First, the sample with the most similar inputs $\boldsymbol{x}_{sim}$ from all kernel elements is retrieved:

$$\boldsymbol{x}_{sim} = \underset{\boldsymbol{x} \in \boldsymbol{X}_n}{\operatorname{argmin}} ||\boldsymbol{x} - \boldsymbol{x}_s||^2, \tag{6.7}$$

where $\boldsymbol{X}_n$ are the inputs of n kernel matrix elements. Then, the output y_{sim} that belongs to $\boldsymbol{x}_{sim}$ will be updated with decaying weighted average:

$$y_{sim} = \alpha_{mem} y_{sim} + (1 - \alpha_{mem}) y_s, \tag{6.8}$$

with α_{mem} being the memorizing factor controlling how much memory is re-contained in the learning system while learning a new sample. The value α_{mem} is in range from zero to one, where zero forgets all old knowledge and one corresponds to discarding the actual sample. A lower α_{mem}, on the one side, increases the adaptation capability of the model. On the other side, the chance that noisy samples negatively influence the prediction is increased.

6.2.3 Case Learning

The limitation of standard GPR comes from the computational complexity of $O(n^3)$ when an incoming sample triggers a kernel matrix change that requires the recalculation of the inverse of the kernel matrix. In contrast, the case learning methods that are used within SOGPR are incrementally obtained and bypass costly direct matrix inversions to reduce the training complexity for real time applications.

Whenever an incoming sample's importance is higher than the minimum importance level δ_{min}, a new kernel element is injected. Here, two cases may occur. First, if the maximum kernel matrix size is not reached, giving the possibility to simply augment the current kernel matrix with an extra entry, which is further referred to as *augmented update*. Second, if the maximum kernel size has been reached, an existing element needs to be removed and replaced by the new sample, which is called *replacement update*.

Augmented Update

If the kernel matrix limit is not reached and an incoming sample triggers case learning, it is possible to update the kernel matrix $\boldsymbol{K}_{old}$ by adding an extra row and column. Therefore, the kernel function is computed between current sample and every kernel

element $\boldsymbol{k}_s$ and itself k_{ss} and added to the new kernel matrix

$$\boldsymbol{K}_{new} = \begin{bmatrix} \boldsymbol{K}_{old} & \boldsymbol{k}_s \\ \boldsymbol{k}_s^T & k_{ss} \end{bmatrix}. \tag{6.9}$$

Unfortunately, the inverse of the new kernel matrix $\boldsymbol{K}_{new}^{-1}$ needs to be computed for the prediction step according to Equation (6.3), which is costly and a major drawback of standard GPR. Originally, every incoming sample triggers the recalculation of the inverse of the kernel matrix with its cubic computational complexity. The direct inversion of the kernel matrix is fine for the initialization phase with few samples, e.g. $S = n \leqq 10$. After this initialization, the current kernel matrix $\boldsymbol{K}_{old}$ and its inverse $\boldsymbol{K}_{old}^{-1}$ are known. Thus, it is proposed according to [Van Vaerenbergh et al., 2010] to compute the new inverse kernel matrix without direct inversion:

$$\boldsymbol{K}_{new}^{-1} = \begin{bmatrix} \boldsymbol{K}_{old}^{-1} + g\boldsymbol{e}\boldsymbol{e}^T & -g\boldsymbol{e} \\ -g\boldsymbol{e}^T & g \end{bmatrix}, \tag{6.10}$$

with $\boldsymbol{e} = \boldsymbol{K}_{old}^{-1}\boldsymbol{k}_s$ and $g = (k_{ss} - \boldsymbol{k}_s^T\boldsymbol{e})^{-1}$. This method takes the advantage of using information contained in $\boldsymbol{K}_{old}$ and runs fast without losing any precision, resulting in a computational complexity of $O(n^2)$. As a consequence of the augmented update, the kernel matrix grows by one extra dimension with each update.

Replacement Update

The dimensional augmentation cannot infinitely proceed due to rising computational constraints. To achieve constant computation time, one has to limit the kernel matrix size. Thus, if the size of the kernel matrix reaches the limit after many augmented updates, an inversion strategy needs to be used that deletes an existing kernel element and injects the current sample. Within this thesis, two different incorporation strategies are implemented and analysed, *sliding window* and *importance-based* update, which can be selected with $type_{forget}$.

a) Sliding Window: The first method that is implemented is inspired by a sliding window scheme, i.e. to learn the incoming sample while discarding (forgetting) the oldest sample from the kernel sample stack. Here, the underlying assumption is, that a relatively large kernel size is used so that each element has less importance. If the oldest kernel element is deleted, it will generate a knowledge loss and two situations may occur: (1) the state space of old kernel entries is well represented by the other kernel elements, resulting in a small knowledge loss or (2) an influential kernel element is lost. In this case, a similar kernel will replace this spot as soon as

experiences within this data space region are again incoming.

However, in order to compute the sliding window update, the scheme derived from [Liu et al., 2011] is applied. Therefore, the actual kernel matrix is rewritten as

$$\boldsymbol{K} = \begin{bmatrix} k_{11} & \boldsymbol{k}_1^T \\ \boldsymbol{k}_1 & \boldsymbol{K}_r \end{bmatrix}, \tag{6.11}$$

with $\boldsymbol{K}_r$ being the residual sub matrix of $\boldsymbol{K}$ of size $(M-1,\ M-1)$. To compute the new kernel matrix discarding the first (oldest) sample, on can stack the new kernel elements as done in the augmented update, i.e. using Equation (6.9) with $\boldsymbol{K}_r$ replacing $\boldsymbol{K}_{old}$.

Analog to the kernel matrix, the corresponding inverse kernel matrix can be rewritten as

$$\boldsymbol{K}^{-1} = \begin{bmatrix} e & \boldsymbol{f}^T \\ \boldsymbol{f} & \boldsymbol{K}_{r^{-1}} \end{bmatrix}, \tag{6.12}$$

where $\boldsymbol{K}_{r^{-1}}$ is the residual sub matrix of $\boldsymbol{K}^{-1}$ of size $(M-1,\ M-1)$, e is the first element of the matrix, and $\boldsymbol{f}$ the remaining column vector. Then, the new inverse kernel matrix can be computed with Equation (6.10) replacing $\boldsymbol{K}_{old}^{-1}$ with $\boldsymbol{K}_{r^{-1}}^{-1}$. The required $\boldsymbol{K}_{r^{-1}}^{-1}$ can then efficiently be derived according to [Van Vaerenbergh et al., 2006] from known matrices with

$$\boldsymbol{K}_{r^{-1}}^{-1} = \boldsymbol{K}_{r^{-1}} - \frac{\boldsymbol{f}\boldsymbol{f}^T}{e}. \tag{6.13}$$

Besides the kernel matrix and its inverse, the first row of $\boldsymbol{X}$ and $\boldsymbol{y}$ needs to be removed and the new sample $\boldsymbol{x}_s, y_s$ is appended at the end. Since n is kept constant, a constant update time with an overall time and memory complexity of $O(n^2)$ is maintained.

b) Importance-based: For the importance-based kernel matrix update, a vector of importance values $\boldsymbol{\delta}$ is needed that holds the corresponding importance measure of each sample. The idea is, that the kernel matrix element with the lowest importance is replaced by the current sample to avoid forgetting of an important experience that represents the model for an entire region of the data space.

Let j be the sample with the lowest importance, then the new kernel matrix $\boldsymbol{K}_{new}$ is given by

$$\boldsymbol{K}_{new} = \begin{bmatrix} \boldsymbol{K}_{old(1:j-1,1:j-1)} & \boldsymbol{k}_{s(1:j-1)} & \boldsymbol{K}_{old(1:j-1,j+1:M)} \\ \boldsymbol{k}_{s(1:j-1)}^T & k_{ss} & \boldsymbol{k}_{s(j+1:M)}^T \\ \boldsymbol{K}_{old(j+1:M,1:j-1)} & \boldsymbol{k}_{s(j+1:M)} & \boldsymbol{K}_{old(j+1:M,j+1:M)} \end{bmatrix}, \tag{6.14}$$

where the j-th column of $\boldsymbol{K}_{new}$ corresponds to $\boldsymbol{k}_s$ and the j-th row to $\boldsymbol{k}_s^T$.

Due to the reordering of elements, $\boldsymbol{K}_{new}^{-1}$ cannot be computed with Equation (6.10). Instead, as proposed in [Nguyen-Tuong and Peters, 2010], the Sherman-Morrison formula needs to be applied resulting in:

$$\boldsymbol{K}_{new}^{-1} = \boldsymbol{\Omega} - \frac{\boldsymbol{\Omega}(j,:)^T \boldsymbol{r}^T \boldsymbol{\Omega}}{1 + \boldsymbol{r}^T \boldsymbol{\Omega}(j,:)^T}, \tag{6.15}$$

with:

$$\boldsymbol{\Omega} = \boldsymbol{K}_{old}^{-1} - \frac{\boldsymbol{K}_{old}^{-1} \boldsymbol{r} \boldsymbol{K}_{old}^{-1}(j,:)}{1 + \boldsymbol{r}^T \boldsymbol{K}_{old}^{-1}(j,:)^T}, \tag{6.16}$$

with $\boldsymbol{r} = \boldsymbol{k}_s - \boldsymbol{K}_{old}(j,:)^T$ which can be interpreted as the difference between old and new column j of the kernel matrix. Simultaneously, the j-th row of the input matrix $\boldsymbol{X}$ and the corresponding output vector $\boldsymbol{y}$ need to be updated as well:

$$\boldsymbol{X}(j,:) = \boldsymbol{x}_s^T \tag{6.17}$$

and

$$\boldsymbol{y}(j) = y_s. \tag{6.18}$$

Summarizing, the importance-based update only replaces the least important information on the costs of higher computational effort compared to the sliding window update.

Importance List Update

Regardless of the case learning update method, a new kernel element is added that also influences the importance of others. Thus, every element of the importance vector $\boldsymbol{\delta}$ needs to be updated after the previous case learning step according to Equation (6.4).

To obtain a certain model adaptation capability, one option is to use temporal loss in form of a time-variant forgetting rate that is scaling down the importance of existing kernel elements after each update. This will enforce the replacement of old kernel elements by up-to-date samples. However, this approach is not followed for SOGPR because the proposed sample learning strategy provides already a smooth adaptation capability that continuously integrates new samples.

The importance list also provides information on how the data space is covered. Thus, it can be beneficial to include new kernel matrix elements only when the new sample has higher importance than the least one of $\boldsymbol{\delta}$ reducing the overall amount of

case learning. By setting $adapt_\delta$ to true, δ_{min} is automatically adjusted with

$$\delta_{min} = \begin{cases} \delta_{min}, & n < M \\ min(\boldsymbol{\delta}), & n = M \end{cases} \tag{6.19}$$

The adjustment is performed after the kernel matrix limit is reached to avoid high values for δ_{min} in the initialization phase, which would hinder the insertion of new kernel elements.

6.2.4 Optimization of Hyperparameters

Particular hyperparameters are included in the utilized kernel function as given in Equation (6.2). Changing their values cast significant influence on the regression performance. For SOGPR, the steepest gradient ascent method is employed for optimizing the hyperparameters [Rasmussen and Williams, 2006]. The intention is to maximize the log marginal likelihood as $log\, p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is the collection of hyperparameters. In this case, it is

$$log\, p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^T \boldsymbol{K}^{-1}\boldsymbol{y} - \frac{1}{2}log|\boldsymbol{K}| - \frac{n}{2}log(2\pi), \tag{6.20}$$

where the first term relates how well the data are fitted, the second punished the function complexity, and the third term is a normalization constant.

The h-th hyperparameter is updated by following the largest gradient ascending direction:

$$\theta_h = \theta_h + \alpha_l \frac{\partial}{\partial \theta_h} log\, p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}), \tag{6.21}$$

with:

$$\frac{\partial}{\partial \theta_h} log\, p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{y}^T \boldsymbol{K}^{-1} \frac{\partial \boldsymbol{K}}{\partial \theta_h} \boldsymbol{K}^{-1}\boldsymbol{y} - \frac{1}{2} tr(\boldsymbol{K}^{-1}\frac{\partial \boldsymbol{K}}{\partial \theta_h}), \tag{6.22}$$

where α_l is the learning rate to incrementally optimize the hyperparameters over several iterations.

The partial derivatives $\frac{\partial \boldsymbol{K}}{\partial \sigma_f}$, $\frac{\partial \boldsymbol{K}}{\partial l}$, and $\frac{\partial \boldsymbol{K}}{\partial \sigma_n}$ are computed by filling every matrix element K_{mn} with the partial derivative of the utilized kernel function of Equation (6.2), which are:

$$\frac{\partial K_{mn}}{\partial \sigma_f} = 2\sigma_f \exp\left(-\frac{1}{2l^2}||\boldsymbol{x}_m - \boldsymbol{x}_n||^2\right), \tag{6.23}$$

$$\frac{\partial K_{mn}}{\partial l} = \sigma_f^2 \exp\left(-\frac{1}{2l^2}||\boldsymbol{x}_m - \boldsymbol{x}_n||^2\right) ||\boldsymbol{x}_m - \boldsymbol{x}_n||^2 \frac{1}{l^3}, \tag{6.24}$$

and

$$\frac{\partial K_{mn}}{\partial \sigma_n} = 2\sigma_n \delta_{x_m x_n}. \tag{6.25}$$

Thus, the GPR hyperparameters can be derived from the data and do not need to be tuned manually. SOGPR introduces additional hyperparameters. Their combined influence is experimentally analyzed with using a cross validation on recorded data bases, which is presented in the next section.

6.3 Model-Based Regression Analysis

SOGPR was developed within this thesis to provide a powerful regression approach that is needed for the prediction of a behavior's performance in a given state context within the model-based behavior adaptation. Thus, an accurate and fast prediction for a multi-dimensional, non-linear mapping has to be provided. In addition, the prediction capability is expected to work with few samples but also has to scale up to an almost infinite amount of training data.

Before using the approach in the proposed model-based behavior inference, only the model's prediction capability is analyzed to evaluate the applicability of SOGPR. Therefore, in a first analysis, the influence of the SOGPR hyperparameters is analyzed. In a second experiment, SOGPR is compared to other regression approaches. A third experiment analyzes SOGPR's adaptation capability to long-term changes.

In order to remove any bias towards a special data set, several data sets of different use cases were used.

- The *Barrett* data set published by [Nguyen-Tuong et al., 2009] provides 15000 samples to learn the dynamics of an anthropomorphic robot arm with seven DOF. Thus, the regression task is to map from a 21-dimensional input space (seven joint positions, velocities, and accelerations) to the corresponding seven joint torques as outputs. The inputs and outputs are normalized to the corresponding maximum and minimum values included in the data. The full data space is uniformly covered.

- The *Sarcos* data set, also published by [Nguyen-Tuong et al., 2009], provides 48933 samples for the Sarcos anthropomorphic robot arm. Type and dimension of the input and outputs are equal to the Barrett data set.

- In order to use a recorded data set of the targeted locomotion application, the `SSM_Exp` knowledge base (Table 4.1) was used and further referred to as *Spaceclimber* data set. In contrast to the manipulation data sets, it has less training data with 1733 samples. Instead of using the smallest and largest values in the data, every value was normalized to the robot- and application-specific capabilities in order to allow the integration of future experiences without changing the normalization intervals. Thus, some regions of the data space are covered more

densely than others. In addition, multiple inputs are identical due to several evaluations of the same behavior. Nevertheless, same inputs may yield different outputs due to noisy state context and performance feature estimations.

- Charlie's knowledge base `CSA_Full` (Table 4.1) is used as *Charlie* data set. On the one side, the data set similar to the Spaceclimber data set in terms of outputs and quality of data. On the other side, it was automatically generated and, thus, contains more samples (42981) while covering a smaller input data space with more evaluations.

Summarizing, each data set contains a different amount of samples and their quality also varies. On the one hand, the manipulation data sets evenly cover most of the possible input space. On the other hand, both locomotion data sets have an irregular coverage and provide more or less noise-infected outputs. To evaluate the accuracy of a model, the root mean squared error (RMSE) between prediction and measured output over all samples is used. Either the average RMSE over all output dimensions is used or, to increase the interpretability of an error, the denormalized value per output dimension. In addition, the overall training time over all training samples t_{train}, the average training time $\bar{t}_{train}$, as well as the average prediction time $\bar{t}_{pred}$ are used as evaluation criteria.

6.3.1 Hyperparameter Analysis

This experiments is supposed to analyze the influence of the SOGPR hyperparameters. In addition, it is intended to find a basic setup, that can be used for any upcoming experiments.

Experimental Setup

Table 6.1 shows the tested hyperparameter values that were evaluated on all data sets. In order to get the best hyperparameter combination for every data set, grid search was used to evaluate all possible combinations. A five-fold cross validation was applied to split each randomized data set into five equally sized training (covering 80% of all samples) and corresponding test data sets (covering 20% of all samples) resulting in 25600 trained and evaluated models. This way, a bias to fortunate or unfortunate data constellations is visible in the evaluation plots. A high variance over the results indicates a high dependency on the samples that were used for training and, consequently, signals less robustness.

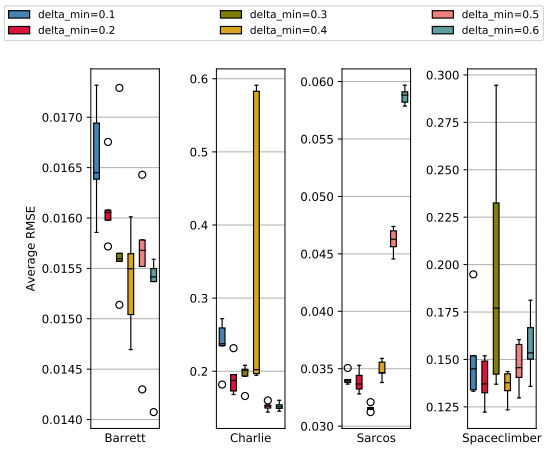
Table 6.1: Tested SOGPR hyperparameters and best combination (bold)

name	description	values
δ_{min}	Initial minimum importance threshold to trigger case learning (depending on $type_\delta$)	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6]$_{DD}$, [0.001, 0.005, 0.01, 0.02]$_{VD}$
M	Maximum size of the kernel matrix	[100, **200**, 300, 400]
α_{mem}	Memorizing factor for sample learning	[0.5, 0.8, **0.9**, 1.0]
$type_{forget}$	Sliding window (SW) or importance-based case learning (IB)	[**SW**, IB]
$type_\delta$	Euclidean distance delta (DD) or variance delta (VD) as sample importance metric	[**DD**, VD]
$adapt_\delta$	Fixed threshold (FT) or automatically adapted threshold (AT)	[**FT**, AT]

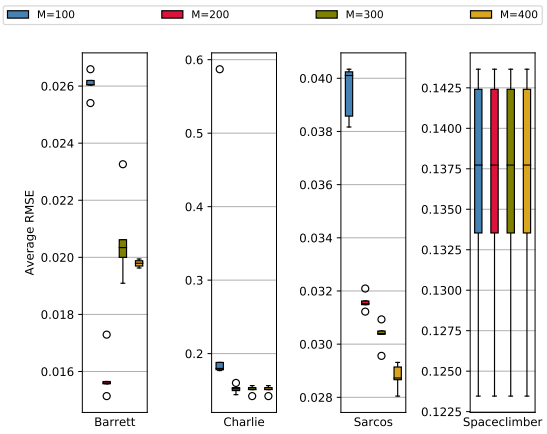
In order to get a scalar for the accuracy of the hyperparameter combination over the k-folds, the average RMSE over all outputs followed by taking the median over the k-folds was used. This value in combination with median training and median prediction time were utilized to select the best hyperparameter configuration for each data set. After the grid search for the best hyperparameter combination, the influence of each hyperparameter was analyzed separately by varying it while leaving the remaining hyperparameters at their best values.

Observations

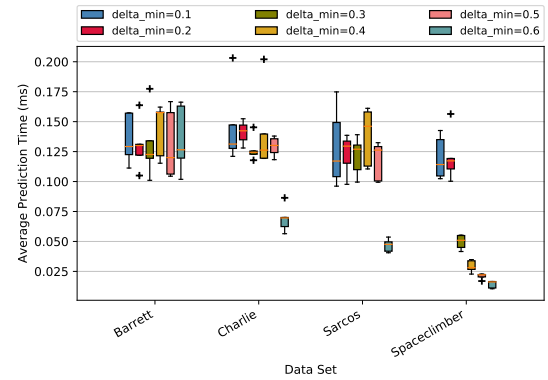
The bold letters in Table 6.1 indicate the best hyperparameter combination for each data set. The threshold to decide whether sample or case learning is initiated δ_{min} is the only hyperparameter that is selected differently for every data set. As depicted in the left plots of Figure 6.3, each data set has its own best δ_{min}. For both manipulation data sets, $\delta_{min}= 0.3$ is optimal. Lower values cause at the beginning of the training process more time-consuming case learning calls that increase the number of kernel elements and, thus, increase $\bar{t}_{train}$ but without noticeable accuracy benefit. Higher values decrease further $\bar{t}_{train}$ due to less kernel elements on the cost of decreased accuracy. A similar observation is made for the locomotion data sets with the difference that the best δ_{min} for Charlie is 0.5 and for SpaceClimber is 0.4. There is no clear tendency for $\bar{t}_{pred}$ except for the small SpaceClimber data set. With increased δ_{min}, $\bar{t}_{pred}$ is decreasing because the maximum kernel matrix size is not reached due to fewer and more similar sample inputs compared to the other data sets. For Charlie and Sarcos, this happens when $\delta_{min}= 0.6$.



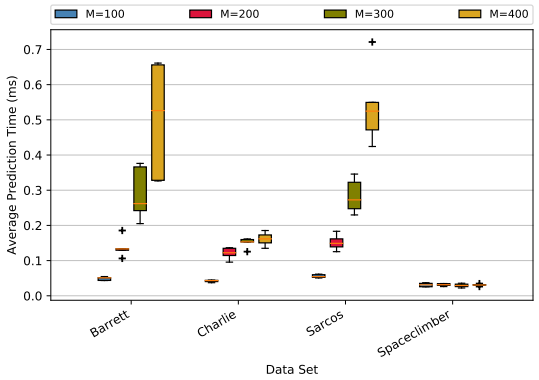
(a) δ_{min} influence on accuracy



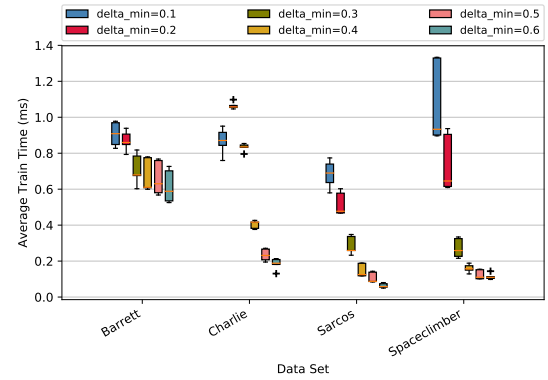
(b) M influence on accuracy



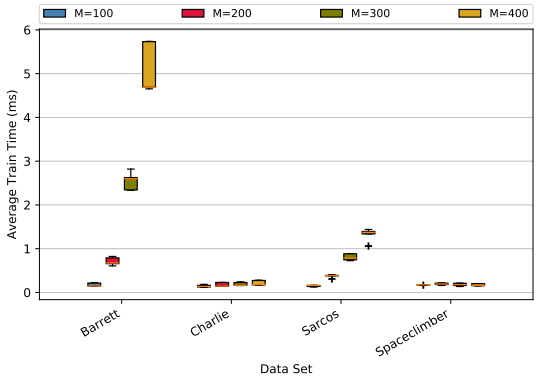
(c) δ_{min} influence on $\bar{t}_{pred}$



(d) M influence on $\bar{t}_{pred}$



(e) δ_{min} influence on $\bar{t}_{train}$



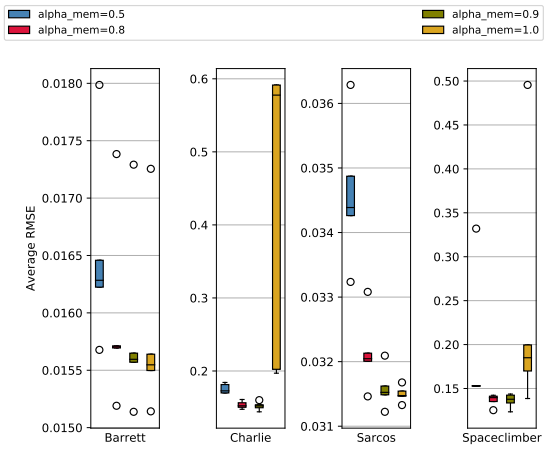
(f) M influence on $\bar{t}_{train}$

Figure 6.3: SOGPR - influence of δ_{min} and M

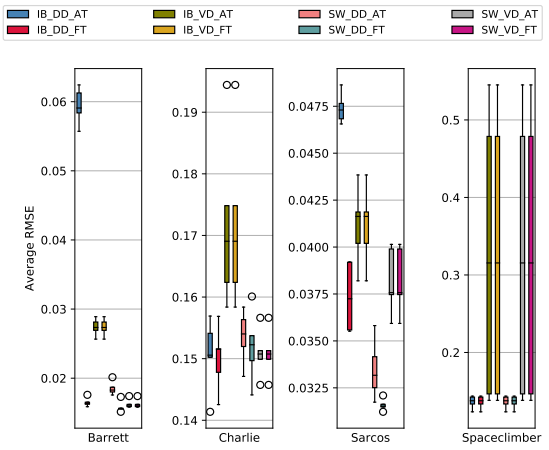
The right plots of Figure 6.3 show that increasing the kernel matrix limit M improves accuracy on the costs of increased $\bar{t}_{train}$ and $\bar{t}_{pred}$. The manipulation data sets show a high correlation between M and $\bar{t}_{train}$ indicating that the heterogeneity of the sample inputs cannot be covered by the tested kernel matrix limits in combination with the chosen sample importance threshold δ_{min}. However, raising the kernel size limit from 100 to 200 is a good trade off between accuracy improvement and increase of computational costs. In contrast, the flattened curve of the Charlie data set indicates that after reaching a size of around 200 less case and more sample learning is triggered. Consequently, a larger kernel size limit does not help to improve the input data space coverage and the prediction accuracy. The Spaceclimber data set is not effected by M because it contains less training data and covers less of the input data space resulting in a kernel matrix size below 65 in any trial.

The sample learning influence, controlled by α_{mem}, has as expected no impact on the computational costs (Figure 6.4c and Figure 6.4e). In contrast, the accuracy (Figure 6.4a) on the noisy locomotion data sets can be improved by $\alpha_{mem} < 1.0$ which will cause an averaging of the kernel elements' corresponding outputs. 10% influence of any new sample, i.e. $\alpha_{mem} = 0.9$, provides best accuracy. The manipulation data set show that discarding new similar samples, i.e. $\alpha_{mem} = 1.0$, has superior accuracy, because an averaging around a respective kernel element reduces the precisely measured sample outputs. Nevertheless, $\alpha_{mem} = 0.9$ has similar accuracy but also improves the models capability to adapt to changing input-output correlations over a long period of time.

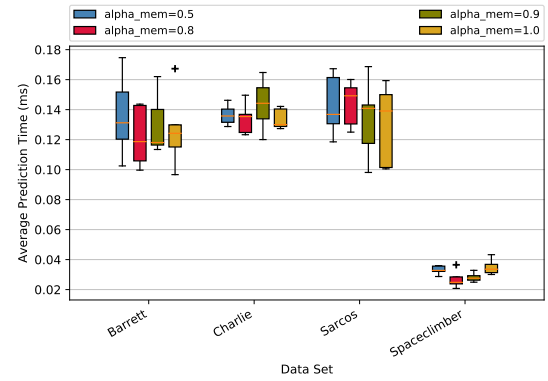
At last, the effects of the remaining boolean configuration options are analyzed, i.e. using the sliding window or importance-based replacement update scheme for case learning (Section 6.2.3), using the distance-based or variance-based importance metric (Section 6.2.1), and adapting δ_{min} during runtime or not (Section 6.2.3). The results of the different combinations are depicted in the right plots of Figure 6.4. The sliding window update scheme reaches slightly higher accuracy than the more sophisticated importance-based update scheme for most data sets. Yet, it can be seen that the sliding window approach has less computational effort during training, that comes from a faster kernel matrix computation which is especially visible in the manipulation data sets were more case learning than sample learning is triggered. This advantage is even enforced when the sliding window update scheme is used without adapting δ_{min} automatically because then there is no need to update the importance list $\boldsymbol{\delta}$. The results of the manipulation data sets also indicate that the automatic adaptation of δ_{min} has the potential to lower the prediction accuracy. This might come from a too fast increase of the threshold for case learning to a very high limit in some cases causing the discard of crucial samples.



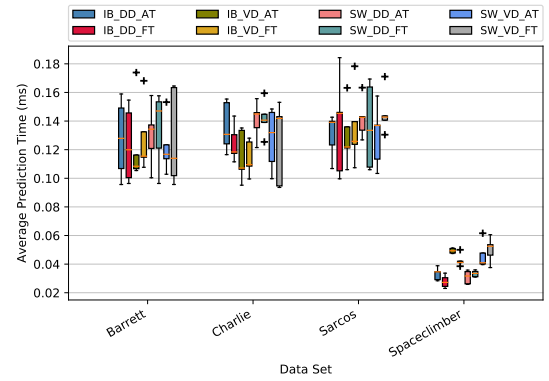
(a) α_{mem} influence on accuracy



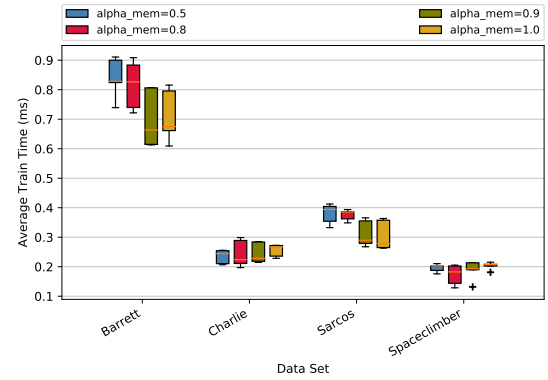
(b) training option influence on accuracy



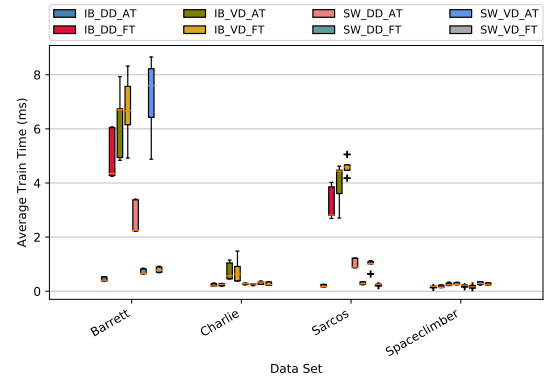
(c) α_{mem} influence on $\bar{t}_{pred}$



(d) training option influence on $\bar{t}_{pred}$



(e) α_{mem} influence on $\bar{t}_{train}$



(f) training option influence on $\bar{t}_{train}$

Figure 6.4: SOGPR - influence of α_{mem} and the boolean training options, i.e. Importance-Based (IB) or Sliding Window (SW) replacement update, sample importance based on distance delta (DD) or on uncertainty-based (variance-based) delta (VD), automatically adapting the importance threshold δ_{min} (AT) or keeping it fixed (FT)

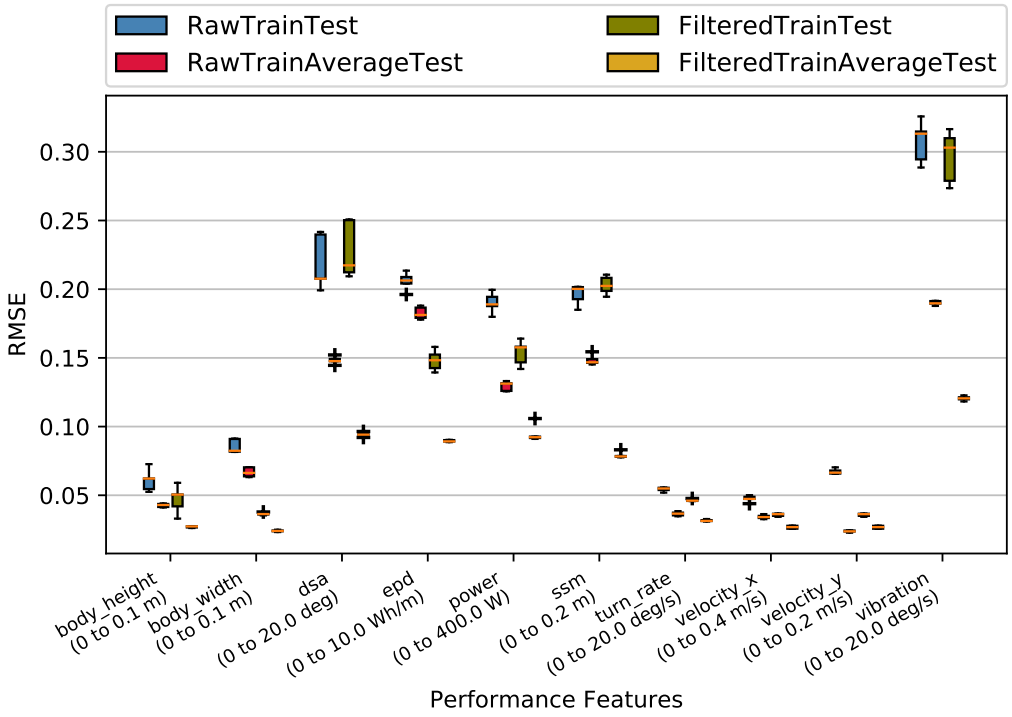


Figure 6.5: Prediction error for Charlie's performance features

In order to maintain comparable results for the variance-based distance as metric for the sample importance, different thresholds for δ_{min} were tested before to find the respective optimum (Table 6.1). Nevertheless, computing the sample importance by using the Euclidean distance, rather than using the uncertainty, seems to be more robust due to less variance in the accuracy evaluation and, in some cases, more accurate predictions.

Comparing the accuracy results between manipulation and locomotion data sets, one notices a larger prediction error for the latter. A closer look on the predicted outputs of the Charlie data set as an example, i.e. the performance features as depicted in Figure 6.5, reveals a clear difference between action and meta performance features when trained and tested with raw data (*RawTrainTest*). The action performance features $\boldsymbol{p}_{vel}$ (`velocity_x`, `velocity_y`, `velocity_rot`) and $\boldsymbol{p}_{posture}$ (`body_height`, `body_width`) can be predicted with around 95% accuracy, whereas the meta performance features `dsa`, `ssm`, `epd`, `power`, and `vibration` can only be predicted with more than 20% RMSE.

This may be caused by output-dependent correlations that cannot be grasped by the chosen hyperparameter combination which is equally utilized for all outputs. Thus, it was additionally tested whether a regression with multiple single models

is more accurate than using one multi-variate model. The results show that only a small accuracy increase of ca. 3% across all data sets can be reached when using multiple models which all have their hyperparameters tuned for every single output. Because training and prediction time for multiple single models almost scale with the number of output features, i.e. by 10, the single multivariate model is still used.

A more influential reason for the low prediction accuracy lies in the data itself. The data is quite noisy since the measured performance characteristics are averaged values over an entire step cycle and already minor state context or execution changes can have large effects on some performance criteria, especially on `dsa` and `vibration`. This fact is underlined by testing the model against averaged prediction values that can be formed from the original test data by averaging all outputs of the same inputs to the new test data set. As indicated in Figure 6.5, especially the prediction accuracy for meta performance features increases. Thus, the average prediction error across all performance features reduces from 14% to 10%.

The remaining prediction error partially comes from outlier of one-time-evaluated inputs that cannot be averaged. The reason for such outliers in the locomotion knowledge bases originate from the inclusion of both, well-working walking behaviors and also critical ones that can lead to instabilities. The later cause large outliers which have an effect on all features, e.g. when the robot tips over just before the evaluation cycle has completed, it can cause a large position change through the falling itself. Consequently, in addition to the resulting low `ssm` and `dsa`, also unusual values will be measured for `velocity_x` and `velocity_y`. Having such experiences in the training data negatively influences the model. In addition, having these none-deterministic experiences in the test data automatically lowers the resulting accuracy. Hence, the prediction accuracy was evaluated again after removing all experiences that included performance values exceeding the application-specific normalization limits, i.e. outliers. As shown in Figure 6.5, the outlier rejection has more influence on rather accurately measured performance features than on noisy data. The overall prediction error slightly decreases from 14% to 12%. Using only plausible data and testing against the average test set provides most accurate results with an overall prediction error of 8%.

Conclusion

A hyperparameter configuration is be found that provides reasonable accuracy paired with fast training and prediction times for all tested data sets leading to the conclusion that the same combination is suitable for other data sets as well. Only the δ_{min} is chosen with respect to the specific data set because it needs to suit the overall coverage of the possible input space within the robot-specific limits. The other parameters stay constant, as indicated in bold in Table 6.1, for further experiments to

represent the standard SOGPR configuration. Even though sample learning is only beneficial for noisy data sets, it also offers the capability to adapt the model to long-term changes, which is analyzed later in more detail. The sliding window update scheme is used as it is faster without accuracy loss. Using the Euclidean distance between samples seems to provide a reliable sample importance metric. The automatic adaptation of δ_{min} is turned off since no advantages are evident.

The selected hyperparameter configuration shows good prediction capabilities paired with acceptable computation costs. Especially the low prediction time of around 0.12 ms (on a 3 GHz core) for a full model allows numerous behavior performance predictions within the optimization loop of the model-based behavior adaptation approach. In order to determine a reasonable cost function, action performance features should be weighted higher, because they are more important for most applications and can be predicted with higher accuracy. Still, even predicted meta performance with lower accuracy can help to avoid instable or inefficient behaviors.

6.3.2 Regression Performance Comparison

In order to compare SOGPR to other regression approaches, the previous data sets were evaluated again in order to compare accuracy and computational costs.

Experimental Setup
Several approaches were implemented and evaluated:

- LGP [Nguyen-Tuong et al., 2009]
- SONIG [Bijl et al., 2017]
- SIGPR [Nguyen-Tuong and Peters, 2011]
- A FNN consisting of an input layer that has $B+Z$ neurons, followed by a configurable number of hidden layers with a configurable number of neurons, and an output layer of A neurons. The network is trained via back propagation.

A coarse evaluation of LGP and SONIG can be found in the appendix (A.1). Due to superior results, SIGPR is selected as online GPR reference and taken for the more detailed analysis. FNN is chosen to serve as reference that is popular in many applications and scientific publications.

In order to obtain a fair comparison, several specific hyperparameters (Table 6.2 and Table 6.3) were evaluated to find an optimal configuration for SIGPR and FNN. For SOGPR, the obtained hyperparameters from the previous experiment were chosen. As for SOGPR, the kernel matrix limit for SIGPR was also set to 200 to generate

Table 6.2: Tested SIGPR hyperparameters and best combination (bold)

name	description	values
δ_{min}	Minimum importance threshold to trigger a kernel matrix update	[0.001, 0.005, 0.01, 0.03, 0.05]
M	Maximum size of the kernel matrix	**200**
f_{loss}	Loss of the kernel element importance per update	**0.999**

Table 6.3: Tested FNN hyperparameters and best combination (bold)

name	description	values
N_{layer}	Number of hidden layers	[1, **3**, 5, 7]
$N_{neurons}$	Number of neurons per layer	[100, **200**, 400, 800
α_{learn}	Learning rate	[0.001, 0.005, 0.01, 0.05, 0.1, 0.2]

similar computational costs. Again, a five-fold cross validation was used and accuracy as well as computational costs were analysed as evaluation criteria.

After finding the best hyperparameter configuration for every data set and approach via grid search, the obtained configurations were used to evaluate and compare each regression approach. In order to analyze the regression performance while a knowledge base grows, selected fractions (0,1%, 0,2%, 0,5%, 1%, 2%, 5%, 10%, 20%, 50%, 100%) of the full data set were drawn and used for training. The models were tested afterwards on 1000 test samples, that were randomly selected from the respective full data set. Each setup was evaluated five times resulting in 150 trained and tested models.

Observations

The respective configurations that combine accuracy with minimal computational costs are highlighted in Table 6.2 and Table 6.3 in bold. For SIGPR, δ_{min} of 0.03 balances high accuracy as well as the time needed for training and predicting. Only the comparably small SpaceClimber data set can be predicted with better accuracy with $\delta_{min}= 0.005$ while maintaining reasonable computational costs. For FNN, three hidden layers with 200 Neurons each has best accuracy paired with low computational costs. Training with a learning rate α_{learn} of 0.1 reaches best results for the manipulation data sets, whereas a lower leaning rate of 0.01 suits better the locomotion data sets.

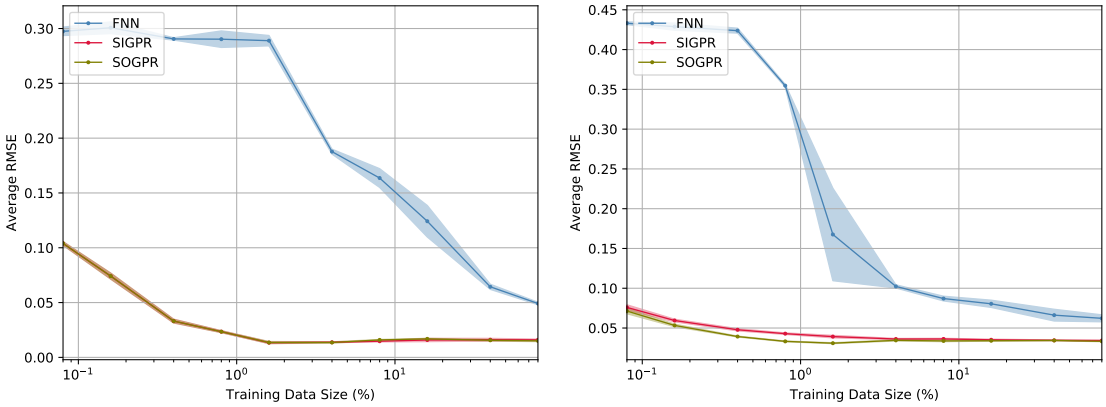
The results of these applied configurations with respect to different data sets and training data sizes is depicted in Figure 6.6 and Figure 6.7. Regarding the manipulation data sets, it is observable that both GPR derivatives predict the test data with higher accuracy compared to FNN (Figure 6.6a, Figure 6.6b). In particular, SOGPR

and SIGPR only require around 2% of the training data, i.e. ca. 250 samples for the Barrett and 800 samples for the Sarcos data set, to reach roughly the final accuracy. When using less than 2% of the data set for training, the kernel size limit is not reached resulting in smaller prediction times (Figure 6.6c, Figure 6.6d). Larger training sets do not increase the time required for prediction since the kernel matrix size stays constant. The average FNN prediction time stays more or less constant around 0.1 ms because the network calculation time is independent of the training data set and progress. The computational effort of SOGPR to train the model is similar to FNN. SIGPR needs more training time for the Barret data set (Figure 6.6e) but less on the Sarcos data set (Figure 6.6f), which is probably caused by the homogeneity difference of the data. When training the model for the Barrett data set, a kernel matrix update is triggered more often because the sample inputs cover a larger input space than the ones from the Sarcos data set.

Looking on the locomotion data sets (Figure 6.7a, Figure 6.7b), SOGPR comparably predicts the test data set with more accuracy for most training data sizes. FNN only outperforms SOGPR in the Charlie data set when more the 20% of the data is provided for training, i.e. more than 10000 samples. SIGPR does not reach the accuracy and has the danger of replacing old kernel elements with noisy samples that are then kept for longer, which may result in accuracy degradation although more samples are used for training, e.g. when using 10% of the SpaceClimber data set. SOGPR is more robust against noisy input data due too its sample learning strategy. Again, FNN needs far more samples to converge and maintains quite constant prediction costs (Figure 6.7c, Figure 6.7d). Both GPR derivatives need less prediction time for the SpaceClimber data set because the utilized kernel matrix size stays below 65. This is different to the Charlie data set, where both GPR derivatives have an increasing prediction time. The kernel matrix size is reached with almost 100% of the training data which finally settles the prediction time. Training SOGPR takes less time than FNN for both data sets (Figure 6.6e, Figure 6.6f). SIGPR is trained as fast as SOGPR for the Charlie data set, whereas for the SpaceClimber data set, a similar training time as FNN is required, because the lower δ_{min} of 0.005 causes the kernel matrix to fill rapidly.

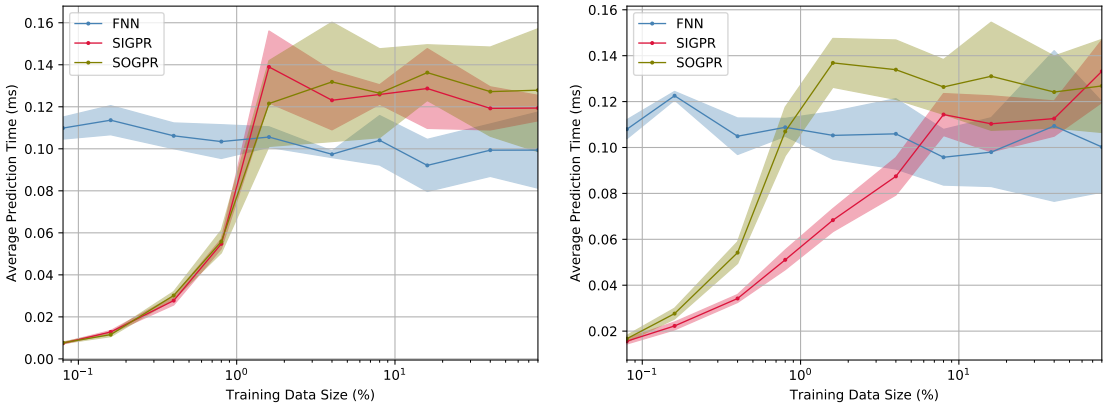
Conclusion

The experimental results show, that SOGPR provides comparably good prediction capabilities, which is especially true for noisy data sets due to the sample learning strategy. Training with few samples delivers reasonable accuracy. Both online GPR approaches prove that they reach high accuracies with few samples, because the influence of every incoming sample during initial model development phase is high. This fact is also visualized on the example of a sine wave in the appendix (A.2).



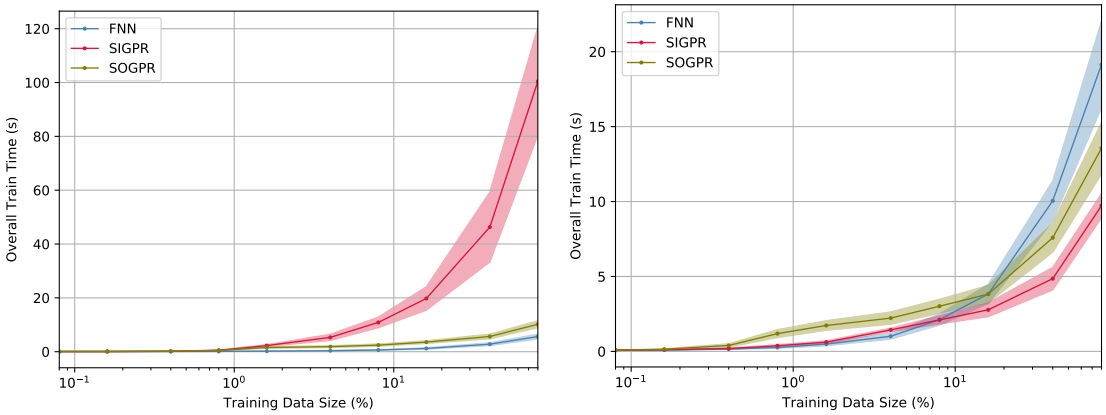
(a) accuracy comparison on Barrett data set

(b) accuracy comparison on Sarcos data set



(c) $\bar{t}_{pred}$ comparison on Barrett data set

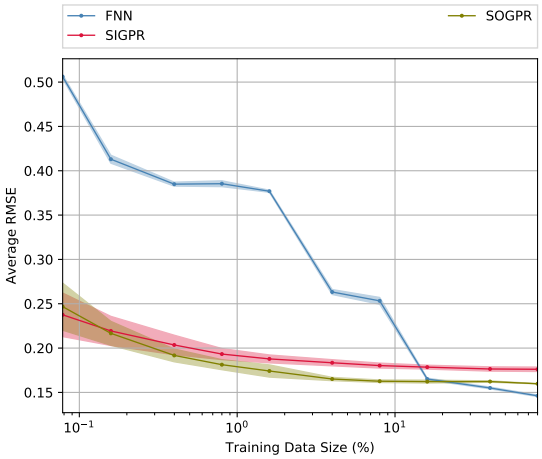
(d) $\bar{t}_{pred}$ comparison on Sarcos data set



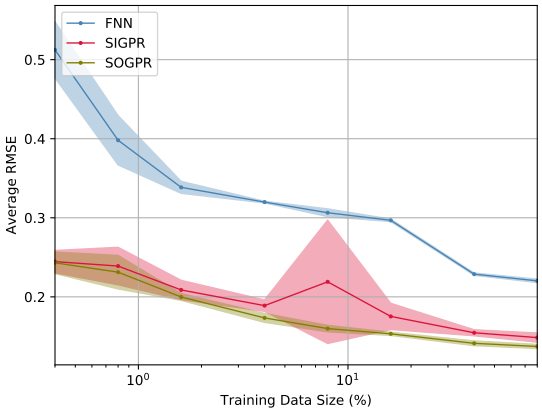
(e) t_{train} comparison on Barrett data set

(f) t_{train} comparison on Sarcos data set

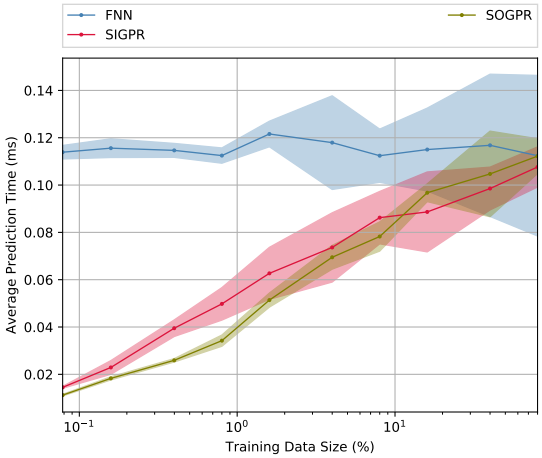
Figure 6.6: Evaluation of regression approaches on the manipulation data sets with respect to different training data sizes. Mean (bold line) and standard deviation (colored range) of the five-fold cross validation results are visualized.



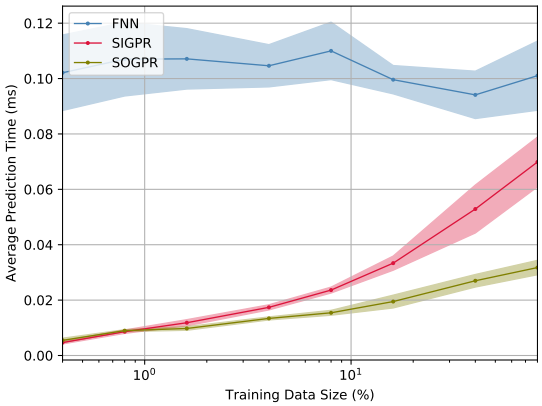
(a) accuracy comparison on Charlie data set



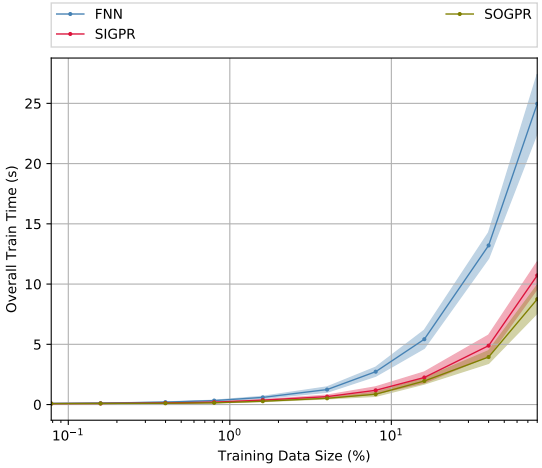
(b) accuracy comparison on Spaceclimber data set



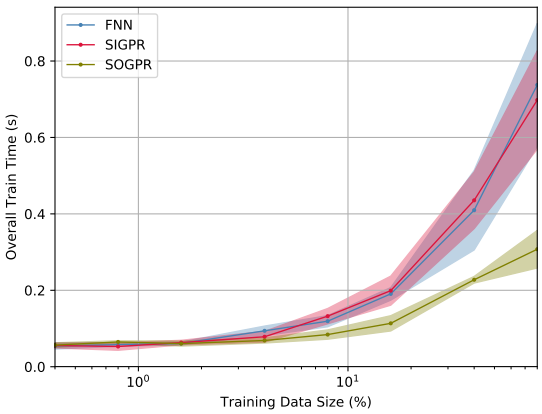
(c) $\bar{t}_{pred}$ comparison on Charlie data set



(d) $\bar{t}_{pred}$ comparison on Spaceclimber data set



(e) t_{train} comparison on Charlie data set



(f) t_{train} comparison on Spaceclimber data set

Figure 6.7: Evaluation of regression approaches on the locomotion data sets with respect to different training data sizes. Mean (bold line) and standard deviation (colored range) of the five-fold cross validation results are visualized.

Training with large data bases further improves the accuracy. Prediction and sample time stay constant as soon as the kernel matrix size is reached, which ensures the application within real-time critical applications. Nevertheless, the limited kernel size limits both, accuracy and computational effort. Thus, applications with numerous input features require larger models, which need to considered when time constraints are present.

6.3.3 Model Adaptation Analysis

A static model for long-term autonomous robots is not desirable since newly detected contexts and a slightly changing performance degradation through wearout may require differently selected behaviors in order to maintain a well-working robot. Thus, the adaptation capability of the model is analyzed in this experiment.

Experimental Setup

Two of the previously used data sets were selected for this experiment, namely Sarcos to represent the manipulation and Charlie to represent the locomotion application. The hyperparameters of each regression approach were taken from the results of the previous experiments.

For this experiment, each model was initialized with four samples. Then, 5000 samples were drawn from the entire data set and sample-wise processed to (1) predict the outputs for the provided sample inputs and compare them with the sample outputs (2) directly utilize each sample's in- and outputs for training the model.

This usual model training, which a robot continuously does during action execution, was then distorted by feeding the model with the same 5000 sample inputs but now with inverted sample outputs. The inversion was done by normalizing every output, subtracting it from 1.0 and denormalizing it again. With this inversion, originally high output values are converted into low ones and vice versa, which simulates a hard performance switch that might be caused by a damaged joint or alike.

Observations

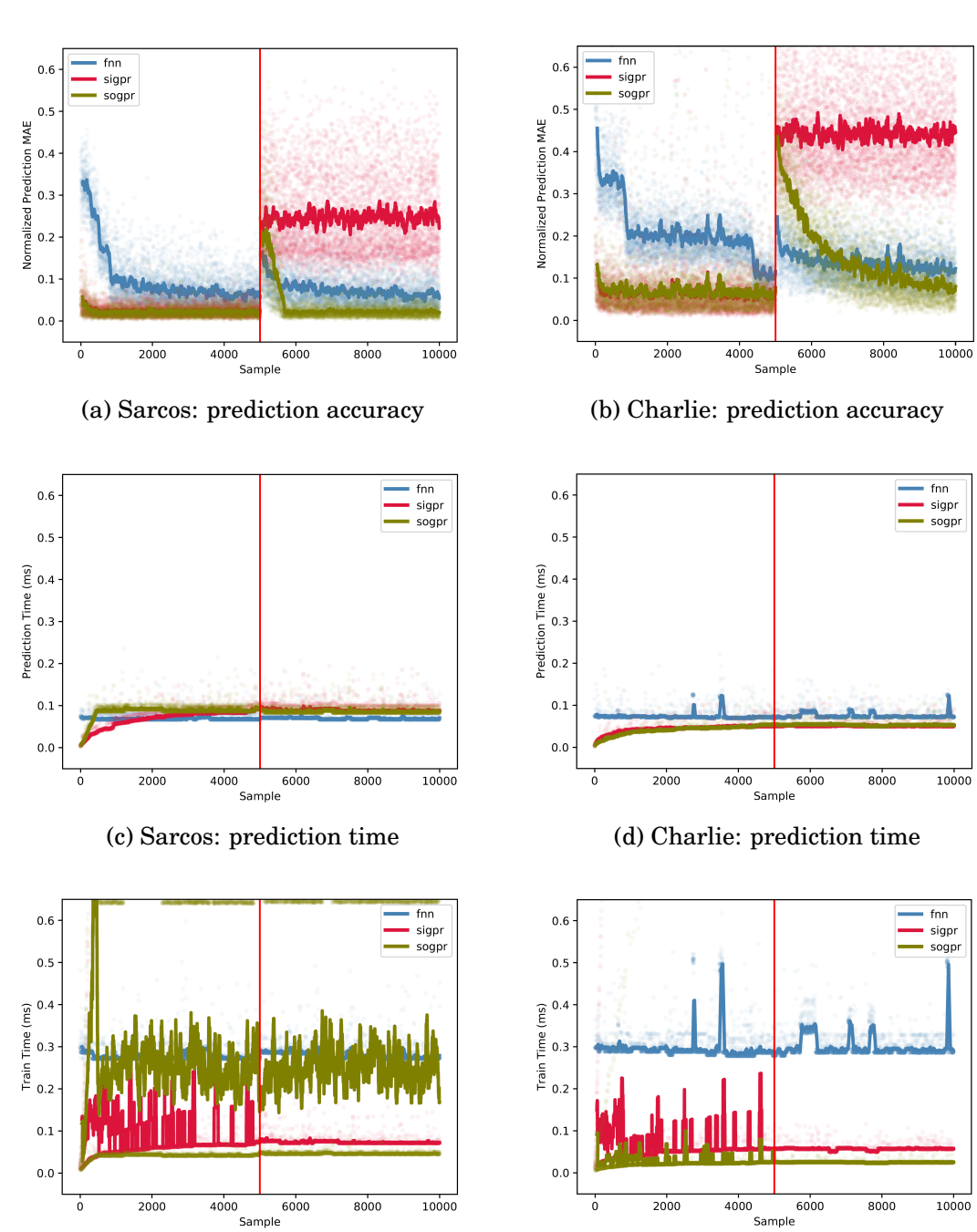
Figure 6.8a and Figure 6.8b show the mean absolute error (MAE) per sample prediction over all outputs. Focussing on the left side of the plots, one can see the normal training process. SOGPR and SIGPR converge to a minimum within the first hundred samples, whereas FNN needs several thousand samples to reach similar accuracy. For the Charlie data set, the steadily increasing prediction time for SOGPR and SIGPR (Figure 6.8d) indicate a growing kernel matrix without reaching the limit of 200 kernel elements. In contrast, the Sarcos data set which uniformly covers the whole input data space, causes a fast grow in the kernel matrix size, especially for

the selected SOGPR parametrization (Figure 6.8c). The full kernel size is also reflected by the training time in Figure 6.8e. Within the first approx. 250 samples, the training time grows rapidly until the kernel matrix limit is reached. Afterwards, on the one side, case learning in form of replacement update is triggered with quite high but constant train times of ca. 0.65 ms. On the other side, it alternates with frequent sample learning since much of the input space is covered. Due to the sample learning's less computational costs, an averages train time of 0.3 ms is reached, similar to FNN.

The inversion of the original testing outputs after 5000 samples is indicated by the red line in the middle of each plot. This switch causes initially less prediction accuracy, since the trained model is not valid any longer. Nevertheless, the combination of sample learning, i.e. continuously merging actual outputs to past outdated outputs of existing kernel elements, as well as case leaning, i.e. throwing out old kernel elements and replacing them with new divergent ones, enables SOGPR to adapt rapidly to new data. Approximately 500 samples are needed for the Sarcos data set to converge again to a minimum prediction error due to many replacement updates. In contrast, it takes serval thousand samples for the Charlie data to converge. As can be seen in Figure 6.8f, only sample learning is triggered because the existing kernel elements cover the entire input space of the test samples. Due to a memorizing factor α_{mem} of 90%, the model adaptation process needs time. However, SOGPR adapts faster than FNN. An adaptation of SIGPR is not visible because it is intended to discard samples of nearby kernel elements. Only a temporal importance loss of 0.1% per incoming sample is foreseen to enforce new replacement updates. That is obviously too small to achieve a noticeable effect within the 5000 samples. Prediction times are not effected in any case since model sizes remain constant after the output inversion.

Conclusion

This experiment shows that SOGPR is capable of adapting to changes with comparably few samples. The very good adaptation to hard changes, as done in the experiment, also implies that slight changes over a long period of time are tracked by the model. Thus, SOGPR is well-suited to be used for model building in long-term robotic applications. Additionally, pre-training on one system and further training on the target system seems to be feasible, e.g. to cross the simulation reality gap. In this case, the collected data from simulation could be reused to reduce the amount of data that needs to be collected on the real hardware.



(a) Sarcos: prediction accuracy

(b) Charlie: prediction accuracy

(c) Sarcos: prediction time

(d) Charlie: prediction time

(e) Sarcos: training time

(f) Charlie: training time

Figure 6.8: Adaptation test on Sarcos and Charlie data set: sample-wise testing and training with 5000 samples followed by sample-wise testing and training with the same samples but inverted outputs. The red line indicates the output switch. The dots represent single samples and the solid line the rolling mean of 50 samples.

6.4 Conclusions

The model-based behavior inference is introduced as one approach for the behavior configurator within the experience-based behavior adaptation architecture. The key idea is to implement an optimization loop that uses a performance estimation model to internally simulate the outcome of numerous behaviors within the estimated state context. A cost function transforms the expected behavior performance and the given action context into a scalar that is used by a black box optimizer to intelligently search for an optimal behavior. Whenever the optimization converges or a given time span exceeds, the currently best behavior is applied on the robot.

Because the performance estimator is used within an optimization loop that needs to converge quickly, a regression technique is needed that can predict a behavior's performance in few milliseconds. Also, the requirement to integrate unlimited training data for long-term autonomous robots given. Therefore, SOGPR was introduced, a new incremental online GPR that uses a limited kernel size and efficient model updates. It incorporates two learning strategies: (1) sample learning for averaging the outputs of similar existing kernel elements and (2) case learning for adding or replacing existing kernel elements. In addition, efficient update strategies are used to lower the high computational costs of the original GPR.

SOGPR is validated and compared to other regression techniques in multiple experiments. SOGPR provides some configuration options concerning the importance evaluation of incoming samples and kernel update strategies. A well-performing hyperparameter configuration was found through numerous experimental evaluations on different data sets, which seems to suit many different applications. In comparison to other regression techniques, SOGPR reaches high accuracy with minimal training effort. Especially, low prediction costs of ca. 0.12 ms that do not increase despite potentially infinite input samples, allow the utilization within the model-based behavior inference. Furthermore, uncertainty information are provided for every model prediction that can be incorporated within the cost function of the optimization loop. Increasing the costs by adding the model's uncertainty leads to more reliable solutions whereas subtracting the uncertainty pushes the exploration of new behaviors. The overall behavior adaptation using the model-based inference is evaluated and analysed in Chapter 7.

Chapter 7

Experimental Evaluation of Case-Based and Model-Based Walking Gait Adaptation

The objective of this chapter is to analyze the experience-based behavior adaptation on the example of autonomous walking pattern selection for the four-legged walking robot Charlie. Charlie's motion controller is based on the modular central pattern approach (Section 3.3). It allows manifold parameter adaptations to change posture, gait, shape of the foot trajectories, and influences of reactive control loops. The correct parametrization of the motion control to create stable and efficient walking patterns usually requires much expert knowledge. In this chapter, the autonomous parametrization based on the case-based and model-based inference for varying action contexts is analyzed, i.e. for changing desired velocities and performance prioritizations. To focus on one state context that can be evaluated in simulation and with the real system, all experiments are conducted on plain flat ground.

The experimental evaluation of this target application consists of a series of experiments. First, the case- and model-based approach are evaluated and compared. Second, the influence of varying performance prioritizations is analyzed for both approaches. Finally, the experience-based behavior adaptation is executed on the physical robot to investigate the transferability from simulation to reality.

7.1 Case-Based vs. Model-Based Walking Pattern Derivation

In this experiment, the proposed case-based and model-based behavior inference strategies are analyzed and compared.

Table 7.1: Varied parameters during knowledge base generation and their ranges (± added white noise of 10% of the parameter range)

Parameter	Min	Max	Default
step_length_z	0.1	0.4	0.2
turn_rate	-0.17	0.17	0.0
t_cycle	2.0	8.0	4.0
phase_shift	0.2	1.0	1.0
step_base_x	0.5	0.6	0.55
body_shift_x	-0.06	0.0	-0.03
body_shift_z	0.4	0.44	0.42
com_cpg_scale	0.0	0.5	0.5

Experimental Setup

The experiment was conducted in simulation. Only Charlie and rigid plain ground were modeled. The automatically generated knowledge base `CSA_Full` was utilized for this experiment because it contains numerous evaluated walking and trot behaviors of different speeds and postures (Table 4.1). To reach this diversity of walking patterns, several behavior parameters were varied during the knowledge base generation process and, thus, are now autonomously adapted. Table 7.1 lists them with their variation ranges during knowledge base generation. The provided default parameters produce a good performing walking pattern and serve as reference.

Based on the results of the case-based behavior inference experiments with SpaceClimber (Section 5.3), a behavior merge of the best three behaviors was applied to generate a new behavior. The SOGPR hyperparameters for the model-based inference were chosen according to the outcome of the hyperparameter experiments (Table 6.1). Having the goal to follow given motion commands, the same performance prioritization was chosen as in the SpaceClimber experiments, which is summarized in Table 5.2, i.e. high priority on commanded `velocity_rot`, and `velocity_x`, a little less on `velocity_y`, and minor priority on `ssm`, `epd`, and `vibration`.

The experiment was split into three parts: (1) The initial knowledge base was utilized by the case- and model-based approach to derive behaviors for the changing motion commands. This was repeated five times to catch the variation between potentially different solutions through slightly different starting conditions and also to discover different optimization results when using the model-based approach. (2) The storing of experiences was activated and further 30 runs were executed with each approach to proceed training the case base or model, respectively. (3) The experience storage was finally deactivated again and the trained case base or model were tested again for five repetitions to compare the results to the utilization of the initial knowledge base.

Every single run was divided into two phases. First, the initialization phase was executed, which starts the simulation with all sensor and actuator drivers as well as Charlie's motion control components. Within this phase, Charlie adopted from its initial model pose where all joints are set to zero to its standard walking posture, which caused time-variant interactions with the ground that led to slightly different starting poses (rotation of few degrees and translation of few centimeters). Afterwards, the setup of the additional components that are needed for the autonomous behavior adaptation (Figure 4.3) followed. This included the creation of the case base or training the model in the first run. In all other runs, the previously generated respective case base or model was loaded directly.

After the initialization phase, a script was executed that sends new target velocities in distinct time intervals. A $0.025\,\frac{m}{s}$ stepwise increase up to $0.15\,\frac{m}{s}$ was commanded to analyze the adaptation process for changing speeds. A stepwise decrease in reverse order followed to check if the same behavior is utilized when the same velocity command is sent the second time. The commands changed every 10 s to provide sufficient time for the adaptation and for the evaluation of the emerged behavior. Since only longitudinal velocities were commanded, Charlie was supposed to move directly along the x-axis of the world coordinate system. To correct the initial pose difference and further path deviations, a desired angular velocity was computed based on the lateral distance to the x-axis of the world frame (p_y) and the current heading of Charlie (ψ)

$$v_{rot} = -\frac{(atan2(p_y, c_x) + \psi)}{t_{cmd}}, \tag{7.1}$$

with c_x being the correction distance that was set to 10 m and t_{cmd} being the selected time interval for every applied motion command.

In order to quantitatively compare the different approaches, following performance scores were used:

- accuracy: average Euclidean distance between theoretical and actually reached position within every command interval,
- efficiency: average `epd` over the entire run, and
- vibration: average `vibration` over the entire run.

Observations

Comparison of Required Computational Resources: Using the case-based inference approach initially requires the generation of the case base from the knowledge base before using it to derive a behavior for the incoming context. Because the utilized knowledge base contains thousands of evaluated behaviors, numerous cases

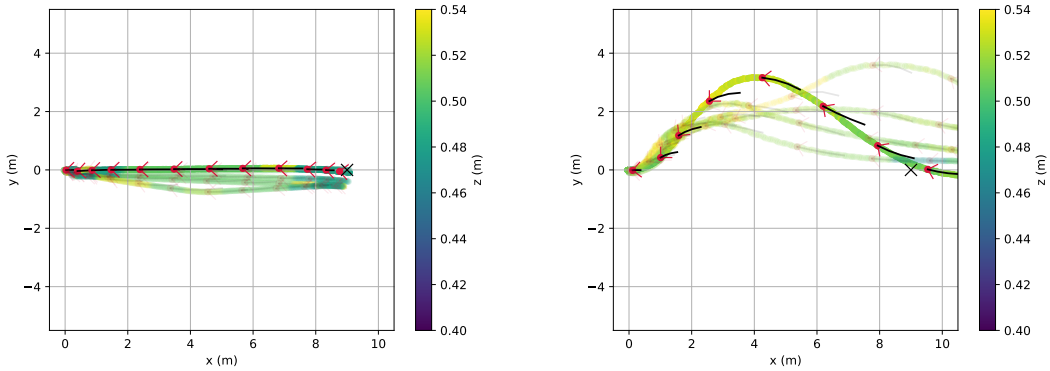
Table 7.2: Summary of disc space and computational costs for using the automatically generated Charlie knowledge base (`CSA_Full`)

Inference Approach	disc space (MB)	t_{train} (s)	t_{load} (s)	$\bar{t}_{solve}$ (s)
case-based	66.1	771.53 ± 32	21.24 ± 4.36	0.84 ± 0.05
model-based	2.8	12.14 ± 1.1	0.033 ± 0.001	0.85 ± 0.05

need to be created and stored. The utilized file-based storage requires 66 MB of disc space and several minutes for the storing process (Table 7.2). However, loading the previously generated case base requires around 21 s. In contrast, the model-based approach requires less than 3 MB for storing the limited kernel matrix, its kernel inputs and outputs, as well as further miscellaneous model information. The whole training process is with ca. 12 s much faster than generating the case base. Loading the model, which is the normal use case during start up, takes only few milliseconds. However, most crucial for real time applications is $\bar{t}_{solve}$ the time needed to generate a behavior when new context information arrives. Here, both approaches show their real-time applicability with average solution times of less than 0.9 s, although not every optimization converged within the given time limit if 1 s.

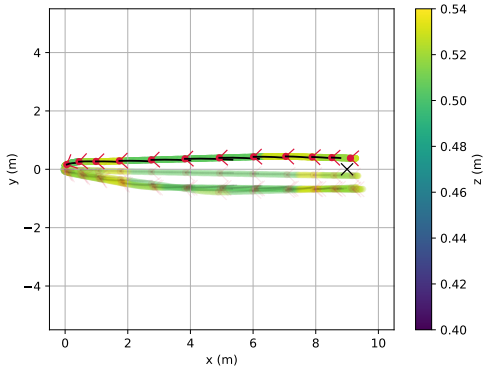
Performance of Utilizing Initial and Trained Knowledge Bases: The model-based approach was tested in different configurations. Without integrating the model uncertainty in the cost function for the optimizer by using Equation (6.1), very extreme behaviors are generated, e.g. very long steps that cause collisions between front and rear legs or very short step cycle times that require joint velocities which the joints cannot realize. Thus, the resulting walking behaviors have a poor ground interaction that cause high vibrations up to tip overs. In contrast, utilizing the model uncertainty in the cost function of Equation (6.5) prevents a good fitness value for these extreme behaviors because no data supports the extrapolated predictions. Consequently, moderate behaviors are executed which lie in the vicinity of previously evaluated behaviors. Due to improved stability, only the setup which uses the uncertainty information of the model prediction is considered for further evaluations. The choice of the parameter optimizer does not noticeably influence the behavior generation. CMA-ES is selected for further applications.

When the desired motion commands would be perfectly executed with infinite acceleration, Charlie should walk on a straight line and end up at 9 m distance, i.e. at $[0.0, 9.0]$ as depicted with a black cross in the scatter plots of Figure 7.1. However, the traversed trajectories of Charlie show, that the model-based behavior adaptation with using the initial knowledge base generates behaviors that differ a lot from the commanded action, which is indicated with a black line at every newly generated command (Figure 7.1b). This is most prominent at slow velocities of $0.025\,\frac{m}{s}$. In

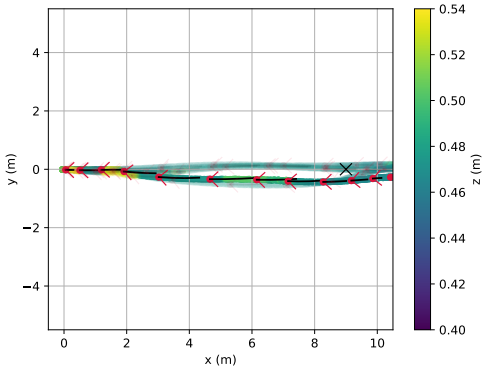


(a) initial: case-based inference

(b) initial: model-based inference



(c) trained: case-based inference



(d) trained: model-based inference

Figure 7.1: Traversed paths (from five repetitions) - the red dot and lines represent charlies pose at every new action command while the black line depicts the path for the upcoming interval if the desired `velocity_x`, and `velocity_rot` would be perfectly executed. For reference, the black cross depicts the position which would be reached if all commands would be perfectly executed.

contrast, the case-based approach generates with the same knowledge base behaviors that produce the commanded longitudinal and angular velocity with high precision and, thus, Charlie almost reaches the theoretical end point within every command interval and finally finishes close to the theoretical end point (Figure 7.1a).

The execution of another 30 runs in between while storing of new experiences was activated, lead to additional experiences to be integrated in the case base (411) and to be used for further training the SOGPR model (395). Using the resulting, so called trained knowledge base, improves the model-based behavior adaptation a lot (Figure 7.1d), whereas the case-based inference has slightly less accuracy (Figure 7.1c). When taking a look at the other performance metrics (Figure 7.2), it is evident that the small drop in accuracy is compensated by less `epd` and lead to less vibrations dur-

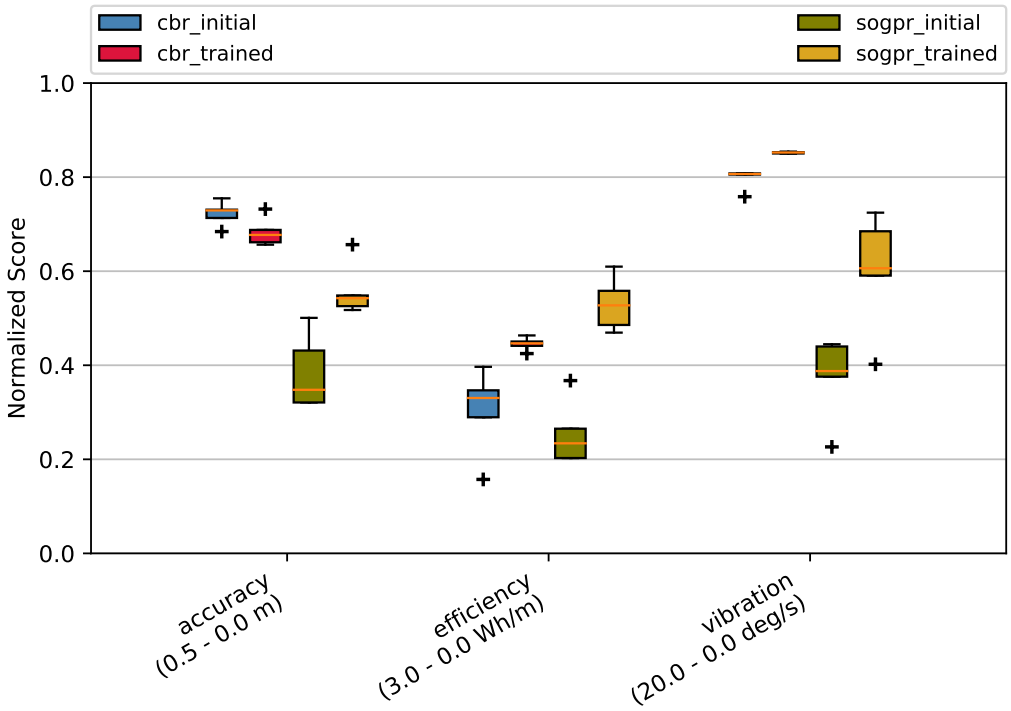


Figure 7.2: Comparison of case- and model-based walking gait adaptation with initial and trained knowledge base

ing locomotion. Figure 7.2 also shows that using the model-based approach with the initial knowledge base has less performance than the case-based approach in terms of vibration, because many parameter combinations are chosen that have a lack of fine tuning and produce bumpy locomotion. When the integration of new experiences is activated, exactly these behaviors are evaluated which efficiently improve the model. Consequently, the model-based approach improves heavily with further training data not only in terms of accuracy but also in terms of `epd` and `vibration`.

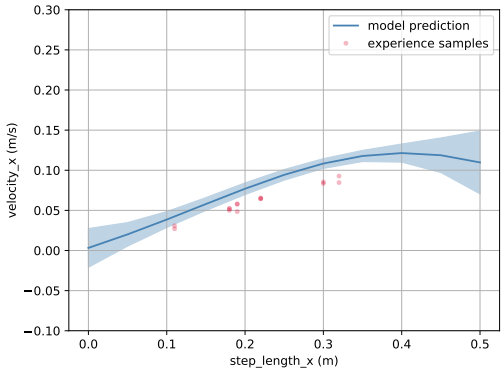
Investigation of the Model-Based Inference: While the proposed behaviors of the case-based approach are reasonable and produce reliable working solutions, the model-based approach needs to be investigated in more detail. Therefore, the plots of Figure 7.3 show the resulting model predictions with uncertainty for varying a single input parameter while leaving the others constant. Also near by samples are visualized which differ more or less from the predicted performance values. This can be the case due to noise or because the model prediction is dependent on many more samples, where most of them are not that similar enough to be plotted but still have an influence. However, all plots show a reasonable discrepancy between theory with-

out considering physical constraints and practice. Even though `velocity_x` should theoretically linearly increase with `step_length_x` according to Equation (3.9), the model shows that this does not match reality. Besides increasing the chance of collisions between front and rear legs, the larger the steps, the faster they have to be moved reaching practical acceleration limits and increasing the interaction forces on the ground, thus decreasing the traction. Also, the in theory hyperbolic relationship between `t_cycle` and `velocity_x` is not visible in the data and in the model prediction because too quick step cycles cause trajectories that cannot be followed by the joints.

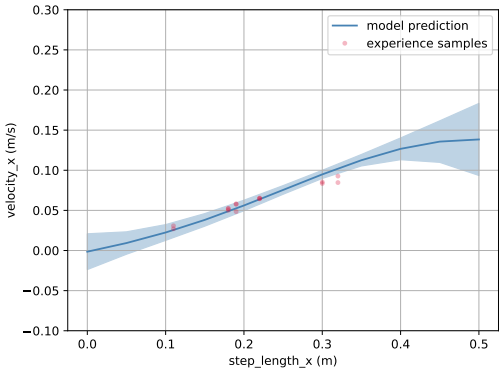
The poor path following capability can be explained by taking a closer look on the resulting model predictions that are based on the initial knowledge base. The model predicts a velocity of ca. 0.025 $\frac{m}{s}$ when `step_length_x` is zero (Figure 7.3a), which clearly is an error in the model. In addition, `velocity_rot` should only depend on `turn_rate` for constant step cycle times, but Figure 7.3e shows still a slight turning motion even for `turn_rate`=0. This can reflect some imbalances of Charlie's kinematically model or its motion control or is an error in the model. If the latter is true, this is another reason for errors during path following. However, after incorporating the additional experiences that were gained through the additional learning phase, the model adapted and now matches better already existing samples and the expectations. Setting `step_length_x` to zero results in no `velocity_x` (Figure 7.3b). Also, the predicted `velocity_rot` matches better the expectation when `turn_rate` is set to zero (Figure 7.3f). There are still theoretical errors left like negative `velocity_x` for `t_cycle`$>$ 7 s as depicted in Figure 7.3d. But also the uncertainty is very high in this region due to the absence of data. Consequently, behaviors in this region of the solution space will be avoided.

Comparison of Case- and Model-Based Inferred Behavior Parameters: Taking a closer look at the generated behavior parameters, one can see rather hard parameter changes when the case-based adaptation is used. Trot is not used (`phase_shift`=1) and also no cyclic shift of the COM (`com_cpg_scale`=0). Figure 7.4e reveals that the maximum required velocity of 0.15 $\frac{m}{s}$ is not reached. Behaviors with higher velocities than 0.1 $\frac{m}{s}$ seem to decrease the other performance metrics a lot and thus, are not executed. The same yields for the slowest desired velocity of 0.025 $\frac{m}{s}$.

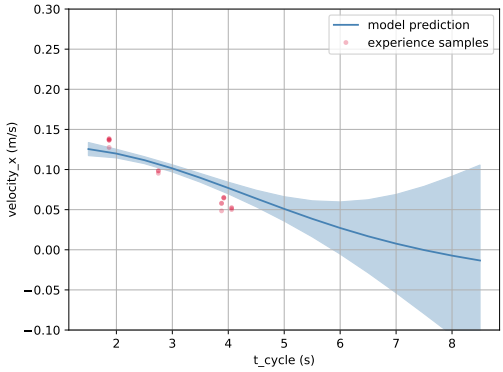
In contrast, the model-based approach generates fine-grained adaptations (Figure 7.4b and Figure 7.4d), e.g. `body_shift_x` correlates with the required speed and is carefully adapted. A slight increase of `com_cpg_scale` is visible for faster walking gaits. The trend to use a higher posture when walking faster as indicated in Figure 7.4a, is not followed by the model-based approach. It is evident that the



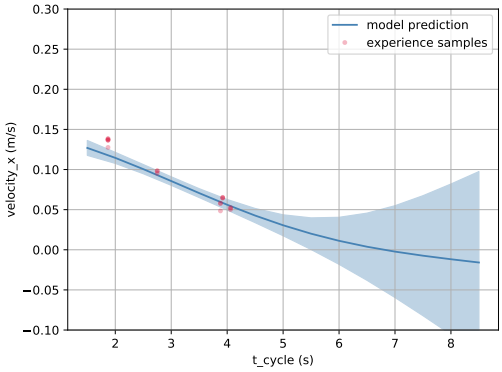
(a) Initial: `velocity_x` over `step_length_x`



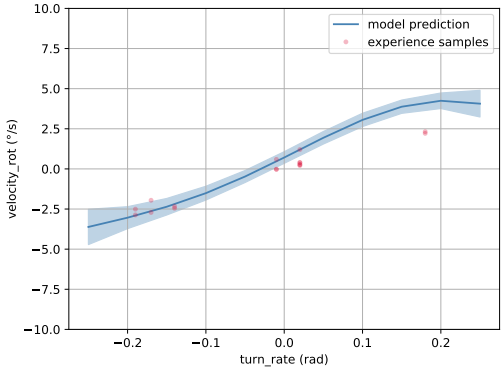
(b) Trained: `velocity_x` over `step_length_x`



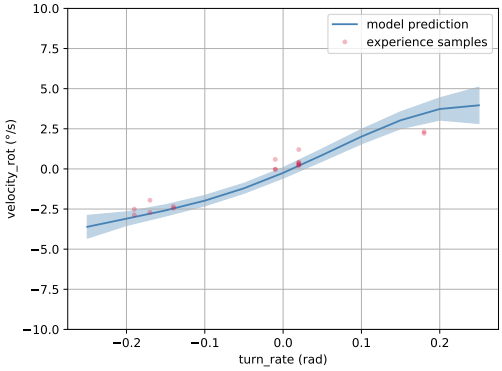
(c) Initial: `velocity_x` over `t_cycle`



(d) Trained: `velocity_x` over `t_cycle`



(e) Initial: `velocity_rot` over `turn_rate`



(f) Trained: `velocity_rot` over `turn_rate`

Figure 7.3: Slices of the multidimensional behavior performance model with model uncertainty. The left side shows predictions for trot behaviors (`phase_shift`=0.2) based on the initial knowledge base whereas the right side depicts the same predictions based on the trained model. All other parameters are kept on their default values (Table 7.1). The red dots show samples from the training data, which inputs have less than 5% deviation from the investigated parameters.

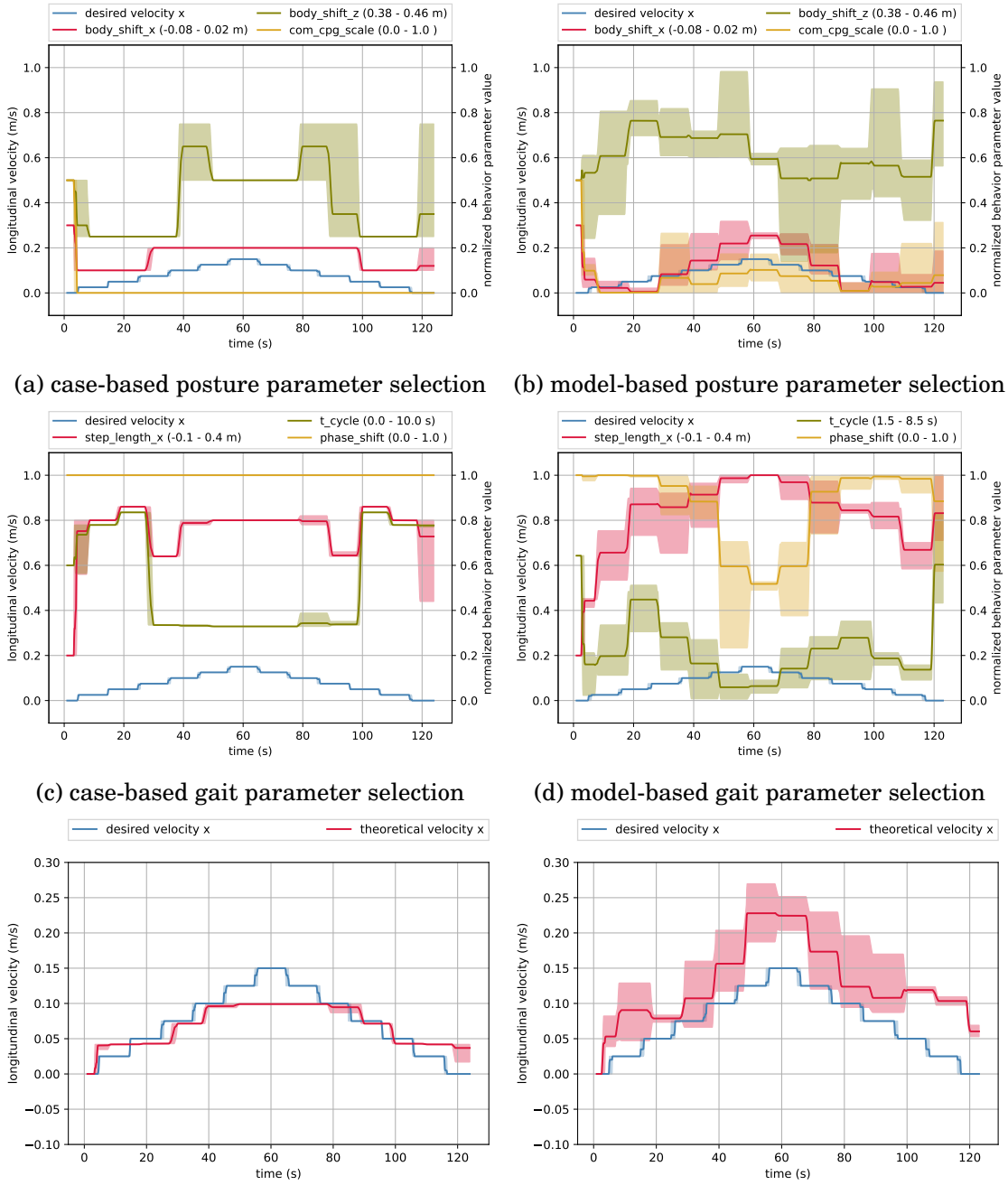
`phase_shift` is reduced for higher speeds. This has a logical reason. The amount of overlapping swing phases of the diagonal legs increase according to Equation (3.8) the maximum allowed swing time $\hat{t}_{swing}$ or the time when all legs are in stance phase. Thus, a smaller cycle time `t_cycle` can be chosen and the maximum velocity of 0.15 $\frac{m}{s}$ is reached (Figure 7.4f). Figure 7.4f reveals that the chosen parameters generate a theoretical velocity according to Equation (3.9) that is higher than the commanded one. This might be needed if the actual velocity is lower than the theoretical one due to influences that are not considered. But Figure 7.1d shows that Charlie overshoots the target point indicating that this is not the case here.

All plots of Figure 7.4 show that the case-based solutions show a symmetry and less variations. This indicates, that reliably the same behavior is generated for the same context. The larger deviations of the model-based approach indicate that the search for the optimal behavior can converge to local minima. Especially, in the situations where the high deviation in the behavior derivation is visible, several optimizations did not converge within the target time of 1 s.

Comparison of Single Leg Walking and Trot: The knowledge base contains evaluations of walking and trot behaviors. An interesting aspect is that when using the model-based approach Charlie intrinsically changes its gait from single leg walking to trot as soon as a certain velocity is demanded. Investigating the multidimensional behavior performance model helps to understand the conducted parameter changes. One reason for trot is that higher speeds with the same parametrization can be reached (Figure 7.5a and Figure 7.5b) while requiring less `epd` (Figure 7.5c and Figure 7.5d), e.g. for the tuple {`step_length_x`, `t_cycle`} = {0.4, 2.0}. Taking a look at the undesired body vibration during locomotion, one can see different input-output correlations between both walking gaits. Taking a constant velocity as an example, which is generated for instance by the tuples {`step_length_x`, `t_cycle`} = [{0.1, 2.0}, {0.2, 4.0}, {0.4, 8.0}], one can see that moderate steps ({0.2, 4}) produce least vibrations for single leg walking, whereas long steps ({0.4, 8.0}) minimize them for trot. In general, single leg walking seems to generate less disturbances than trot when making small and moderate steps. This is one major reason for the selection of single leg walking for slow and medium velocities with respect to the chosen performance prioritization.

Conclusion

Summarizing, the model-based inference is superior to the case-based inference for very large knowledge bases in terms of computational resources. The model-based adaptation process is on the one hand more fine-grained through good interpolation



(a) case-based posture parameter selection

(b) model-based posture parameter selection

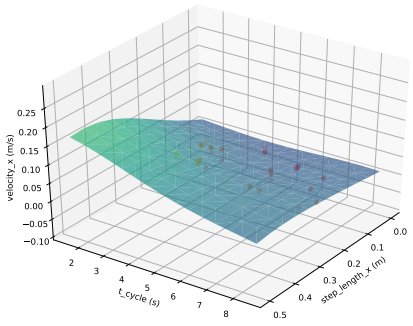
(c) case-based gait parameter selection

(d) model-based gait parameter selection

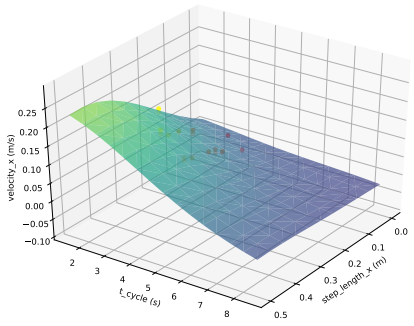
(e) case-based longitudinal velocity

(f) model-based longitudinal velocity

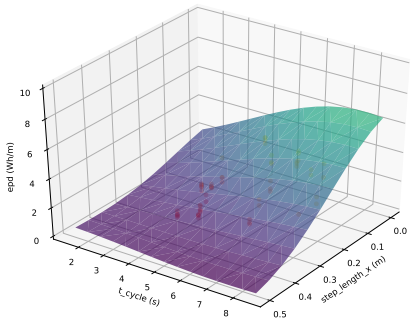
Figure 7.4: Adapted parameters and resulting motion based on `CSA_Full` - the line represents the median of all repetitions whereas the shaded regions bound the maximum and minimum values.



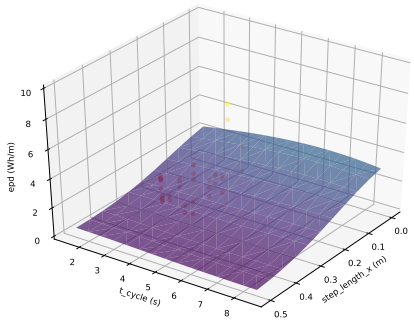
(a) Walk: view on `velocity_x`



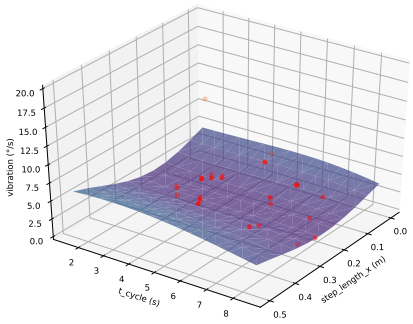
(b) Trot: view on `velocity_x`



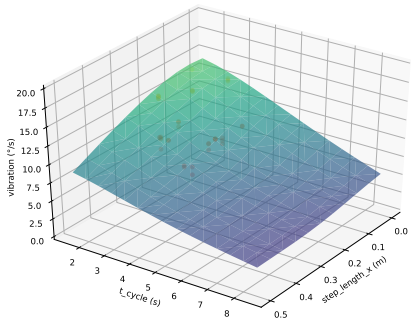
(c) Walk: view on `epd`



(d) Trot: view on `epd`



(e) Walk: view on `vibration`



(f) Trot: view on `vibration`

Figure 7.5: Selected 3D views on the multidimensional behavior performance model. The left side represents walking behaviors (`phase_shift`=1) whereas the right side depicts trot behaviors (`phase_shift`=0.2). The parameters which are not varied are summarized in Table 7.1. The red dots show experiences from the training data, which have less than 5% deviation from the regarded parameters.

capabilities. On the other hand, both, the interpolating and extrapolating characteristic can create non-optimal solutions for desired actions that are not included during training. Incorporating the provided prediction uncertainty in the cost function of the optimization loop is mandatory to degrade unrealistic model predictions at parameter boundaries to finally produce working solutions. The case-based approach delivers more reliable behaviors because the derived parameters stay very close to already evaluated behavior parameters providing non-optimal but working solutions. In both approaches, training data is desired that covers the whole state space and especially the points with changing input-output relations. Especially the model-based approach profits from training samples for input combinations that were not analyzed before.

In this experiment, the model-based inference strategy leads to an autonomous transition of gaits from single leg walking to trot for higher target velocities. According to the behavior performance model, trot can reach higher velocities with higher energy efficiency. In contrast, single leg walking has better performance when walking slowly and produces less undesired vibrations. The behavior performance model catches important aspects of the complex relation between mutual-dependent behavior parameters and the corresponding performance metrics. Nevertheless, little model errors still exist which lead to non-optimal behavior adaptations.

7.2 Influence of Performance Prioritizations

The plots from Figure 7.5 show that the performance features are highly depending on the behavior parameters. Vice versa, changing the priorities of performance features requires to adapt the behavior parameters to optimize for the new circumstances. In this experiment, the performance prioritization is varied to analyze the impact on the generated behaviors.

Experimental Setup

The setup was identical to the previous one, except that the trained knowledge bases of the previous experiment were utilized. Charlie received the same velocity commands. Before execution, the performance prioritization was changed to following configurations:

- `ref_prio`: All weights set to the path following prioritization (Table 5.2).
- `acc_prio`: as `ref_prio` but w_{ssm}, w_{epd}, and w_{vibe} set to 0.0.
- `eff_prio`: as `ref_prio` but w_{epd} increased from 0.1 to 1.0.
- `vib_prio`: as `ref_prio` but $w_{vibration}$ increased from 0.1 to 1.0.

Each performance setup was tested five times and evaluated as in the previous experiment.

Observations

Figure 7.6 summarizes the outcome of the tested performance prioritizations by depicting a score for every performance value, where a high score represents less position error, few `epd`, and less vibrations, respectively. In general, the case-based behavior adaptation achieves the best results in every category. Besides best performance values, less deviations between every trial is noticeable speaking for more reliable behavior adaptations. The model-based approach is more sensitive to performance prioritization changes. Setting a weight to zero discards the corresponding performance value from the optimization process which changes the fitness landscape and, thus, enforces the generation of completely different behaviors. Nevertheless, an acceptable score for the removed performance feature still remains because all performance features are interconnected with each other, e.g. a very accurate behavior still obtains a reasonable energy efficiency and vibration score because it requires a walking behaviors with proper ground interaction to follow the requested motion commands.

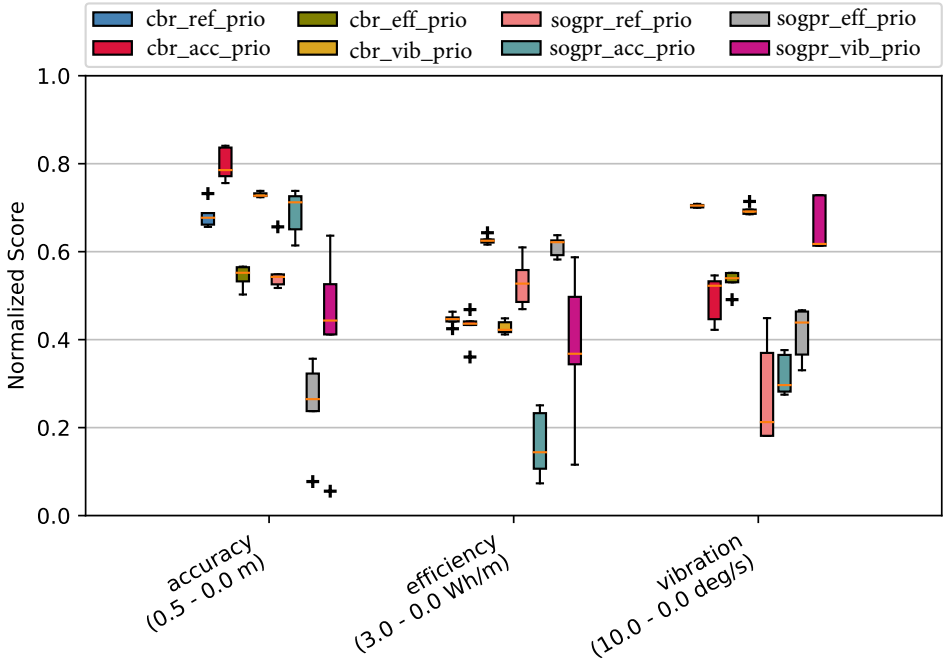


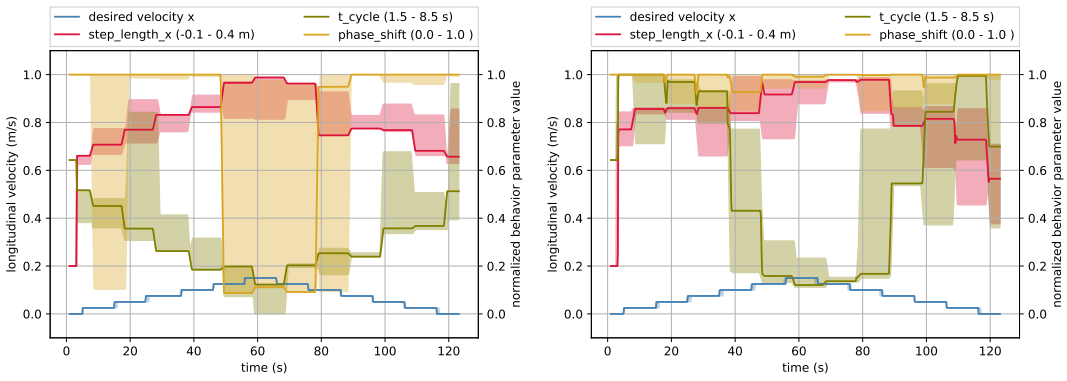
Figure 7.6: Performance comparison of different performance prioritizations

Regardless of the behavior inference approach, the results show that favoring a specific performance feature increases the corresponding performance value on the cost of others. Focussing accuracy decreases energy efficiency and increases undesired vibrations. Increasing the focus on energy efficiency reduced the required `epd` but simultaneously lowers the precision in following commands. For example, very slow walking patterns are avoided because they are inefficient due to relatively high power consumption just for maintaining the body posture and powering the electronics. When prioritizing less vibrations, no noticeable effect on the behavior adaptation can be seen for the case-based inference. In contrast, the prioritization change has a large influence when using the model-based inference. Compared to the reference prioritization, a higher vibration score is reached while maintaining similar accuracy and efficiency scores. This is true for the median, but higher variances between every repetition indicate less reliability off the generated behaviors, which also reflects that `vibration` can be predicted worse due to its high noise level as presented in Section 6.3.

The prioritization influence on the derived behavior parameters is exemplarily shown in Figure 7.7. When focussing on energy efficiency, trot is selected for higher velocities by reducing the `phase_shift`, which reflects the correlation that is shown in Figure 7.5c and Figure 7.5d. In contrast, the `vib_prio` prioritization causes the avoidance of trot behaviors because energy efficiency plays no role and single leg walking behaviors cause less vibrations, as depicted in Figure 7.5e and Figure 7.5f. It is also evident, that for `eff_prio` prioritization `step_length_x` and `t_cycle` are continuously adapted whereas for `vib_prio` the speed is mainly influenced by `t_cycle`. `step_length_x` stays quite constant at 0.3 m for varying speeds. This can be explained by the valley on Figure 7.5e. It suggests that the undesired vibration is minimal around 0.2 m to 0.3 m while `t_cycle` has almost no influence.

Conclusion

This experiment shows that the autonomous behavior adaptation can be tuned via changing the performance prioritization. This can be done for every applications scenario or, if necessary, during runtime. If representative performance measures are used, the weight tuning is intuitive and can be handled by the applicant without being an expert of the underlying motion control. Having at least a minor weight on every performance feature should produce relatively stable results. In any case, to verify whether a performance value is suitable to tune the adaptation process, an analysis as done in Section 6.3 can be conducted to test if the model can learn a proper mapping.



(a) `eff_prio` (b) `vib_prio`

Figure 7.7: Comparison of selected gait parameters for different performance prioritizations

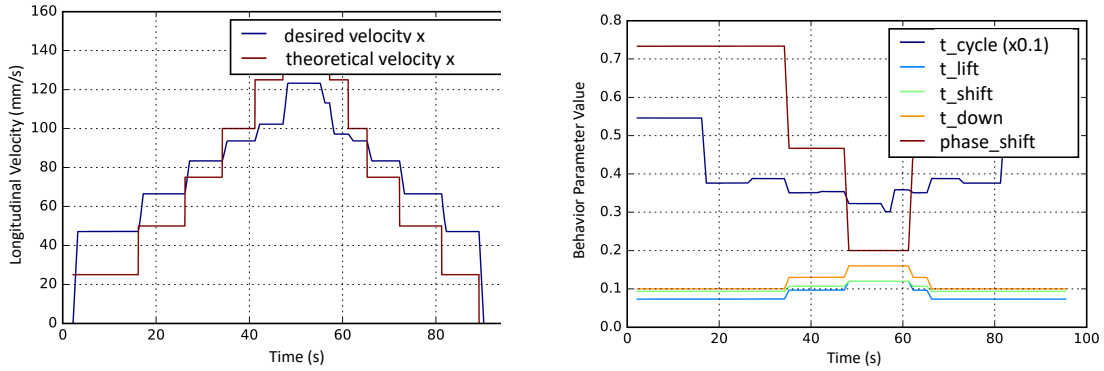
7.3 Transferability to the Physical Robot

In this experiment, the behavior-based adaptation is applied on the physical system to analyze its transferability. Additionally, the model prediction based on the collected real experiences is analyzed.

Experimental Setup

In order to traverse a speed profile similar to the previous experiments, Charlie was placed on a treadmill of ca. 7 m x 3 m length and width that allows long traverses without having the need to turn around or to stop the system. The treadmill is equipped with two ultra sound sensors that measure the position of an object on the running surface. These measurements are used to control the speed of the treadmill to maintain a centralized position of an object on the running surface. Due to this setup, a robot automatically stays in the middle of the treadmill in longitudinal direction regardless of its walking speed. Any lateral drifts needs to be compensated by additional turning commands to avoid falling from the treadmill.

The large knowledge base of the previous experiments contains manifold behaviors that cover large intervals of selected behavior parameters including unfortunate combinations that can create instable walking behaviors. For this experiment, a knowledge base was recorded (`CSM_Exp`) with less behavior diversity and reduced parameter ranges , e.g. avoiding too long step lengths and to short step cycle times. Its characteristics are summarized in Table 4.1. During the generation process, also `t_lift`, `t_shift`, and `t_down` were increased for trot behaviors to use the potential of their prolonged swing phases. Due to its superior path following capability and more reliable parameter generation, the case-based behavior inference was used.



(a) Comparison of desired and theoretical velocity

(b) Derived gait parameters

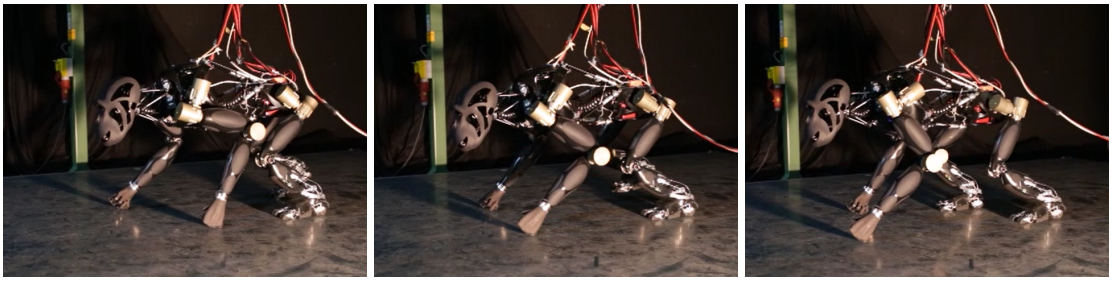
Figure 7.8: Experience-based adaptation applied on Charlie

Before the experiment started, Charlie was placed on top of the treadmill and all software components were started as in the previous experiments, except that the simulation was replaced by the corresponding hardware drivers. After the initialization, the desired longitudinal velocity was increased in 0.025 $\frac{m}{s}$ steps up to 0.15 $\frac{m}{s}$. This time, the path-correcting rotational velocities and time span between commands were controlled manually. During the experiment, the behaviors were evaluated and stored in a new knowledge base (`CRM_Exp`) to be able to analyze the difference between simulated and real experiences.

Observations

Figure 7.8 shows how Charlie adapts its walking behavior to the changing action context. Even though the theoretical velocity that results from the selected behavior parameters is not ideally matching the desired velocity (Figure 7.8a), an overall correlation is clearly visible. In order to reach high velocities, a gait transition is initiated from single leg walking to trot for speeds above 0.1 $\frac{m}{s}$. The phase shift between the diagonal legs is reduced, increasing the possible air time for each leg. The prolonged swing time is efficiently utilized by increasing `t_lift`, `t_shift`, and `t_down` to lower the required joint speeds during swing phase (Figure 7.8b).

The applied behaviors on the real Charlie look quite similar to their execution on the simulated counterpart. This speaks for a quite low simulation reality gap and good transferability. The main difference can be seen during trot (Figure 7.9) especially in the swing phase. The physical system falls to the front causing an early touchdown of the corresponding front foot. Because the surface is smooth, the corresponding front foot slides over the surface for the remaining shift phase and no major differences in the performance features are noticeable.



(a) Before liftoff (b) During swing phase (c) After touchdown

Figure 7.9: Charlie on a treadmill during trot

Through the evaluation of the executed behaviors during the experiment, new experiences were collected. They surely differ to the simulated experiences due the simulation reality gap. This can be quantified by comparing the prediction error of the SOGPR model that is trained with simulated experiences and testet on simulated data with the error that evolves when using the same model to predict the real data. Therefore, a five-fold cross validation was used. The red and blue boxes from Figure 7.10 show that the prediction error of every single performance feature is increased. `velocity_x` and `vibration` are less effected, but an average increase from originally 6,2% of prediction error between training and testing on simulated data to 16,2% when testing on real data can be observed. Fortunately, newly incoming experiences can directly be integrated to improve the performance prediction model online. Using this hybrid model, i.e. initially trained with simulated data and further on trained with real data, reduces the average prediction error to 9,2%, which is indicated by the green boxes in Figure 7.10. This error is similar to the prediction error of a model that is trained and tested entirely on the real data (9,6%).

The worst prediction error of roughly 35% can be observed for `epd`. Figure 7.11a shows that the model that is trained in simulation has a slight offset and has less amplitude in situations where the real data contains large `epd` spikes. Both, the hybrid model and the one which is only based on the real data, catch better the spikes on the costs of a larger offset for moderate values. That the model is capable of learning offsets, shows the prediction errors for the performance feature `power` (Figure 7.11b). The prediction of the simulated model has compared to the real data a large offset. The model is also not capable to model expected power spikes. In contrast, the models that utilize the real data, accurately predict `power`. No offset is left and the spikes are caught with more accuracy. The prediction error is even smaller than using the simulated model to predict simulated data, which might be an indication that the real power measurement is more coherent than its simulated, modelled counterpart. Figure 7.11c shows that a quite reasonable prediction error for `vibration` of the simulated model that cannot model the amplitude of every spike,

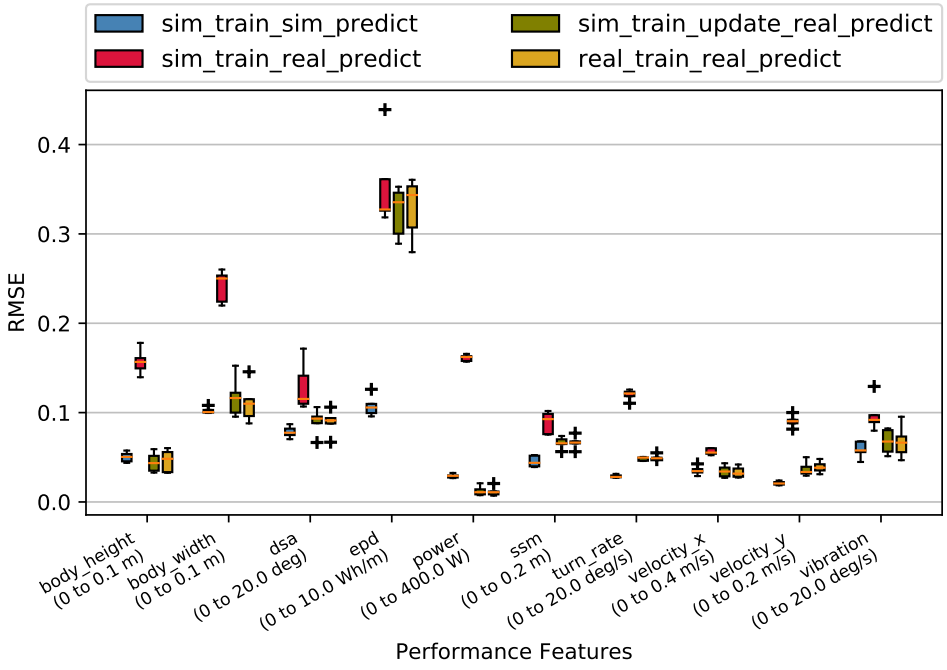
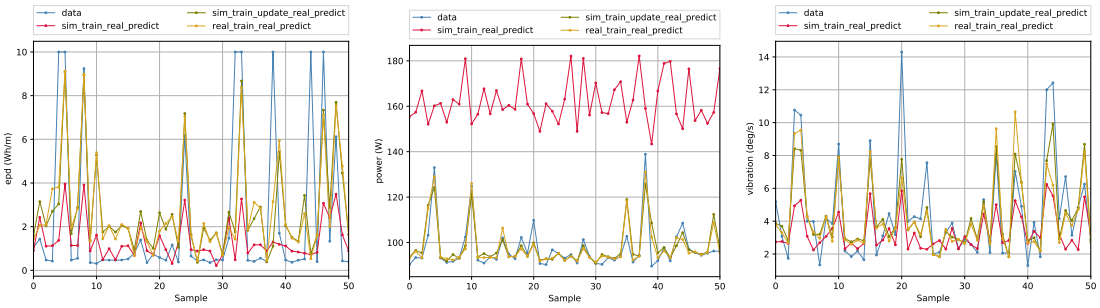


Figure 7.10: Comparison of model predictions based on different training and test data. `sim_train_sim_predict` refers to a model that is trained and tested on the simulated data set `CSM_Exp`. `sim_train_real_predict` shows the prediction error when these models are use to predict the test data of `CRM_Exp`. `sim_train_update_real_predict` shows the prediction error when the simulation-based models are retrained with the train data of `CRM_Exp` to predict the test data of `CRM_Exp`. `real_train_real_predict` simply uses `CRM_Exp` for training and testing. All results are generated by using a five-fold cross validation to separate training from test data and are plotted as normalized errors according to the displayed normalization intervals that represent valid robot-specific performance ranges.



(a) `epd` (b) `power` (c) `vibration`

Figure 7.11: Comparison of real test data and model predictions based on different knowledge bases for selected performance features (random samples from the test set are drawn, i.e. no time series is plotted, the lines are drawn to improve visibility.)

can be improved without inducing an extra offset. Both models that include real data model the spikes more accurately without loosing the precision for moderate vibration values.

Conclusion

In this experiment, the experience-based adaptation is successfully applied on Charlie. Due to the realistic simulation, a solid knowledge base can be generated in advance and directly applied on the physical system. The resulting walking behaviors show minor performance differences. The commanded motion commands are executed while utilizing different walking gaits. Even the transition from walk to trot and vice versa can be realized without instability due to the underlying modular central pattern control approach. However, the results are specific to the underlying surface. A transferability to other substrates is unlikely to perform similar. Further training and testing in similar simulation and real scenarios would be required to analyze it as well as introducing a corresponding state context feature.

Even though the executed behaviors are not optimal when using the experiences that were generated in simulation, it is shown, that the prediction error of the performance model decreases with every incoming experience that is made with the physical system. The model adaptation leads then to different behaviors that will be executed and evaluated again. Thus, the resulting hybrid model that consists of simulated and real experiences quickly emerges to a sophisticated performance estimator leading to continuously improving solutions. The question whether to create a new model and archive the old model depends on the probability to reuse the old model again. For example, if a joint breaks that causes a similar performance drop as in the case of the observed simulation reality gap, it will be beneficial to keep the old model. Therefore, the behavior library can be used to manage different knowledge bases and models and to select the right one for the current situation, e.g. when the joint is repaired, the old model is more accurate and retraining the adapted model is not needed. A large and persisting performance drop can be used as an indicator to back up an existing model. Usually, creeping performance changes like wearout will constantly be tracked within the same model. Comparing the initial and final model is here also a possibility to quantify the wearout which will play an important role in the future as long-term systems will be utilized more often.

7.4 Conclusions

The experimental results prove the general applicability of the experience-based behavior adaptation on the example of adapting the behavior parameters for the walking robot Charlie. In this experiment series, context changes in terms of changing desired velocities trigger the autonomous behavior adaptation that finally creates stable walking behaviors. The adaptation considers, besides obvious behavior parameters like the length and time for a step, also postural parameters and the gait type, which all have mutual dependencies and need a harmonious combined configuration. The fact, that an automatically generated knowledge base can be utilized for the behavior derivation shows that the use of expert knowledge can be omitted except for defining application-specific boundaries for behavior parameters as well as for state context and performance features.

The comparison between case-based and model-based inference approach reveals that the latter scales better for a large amount of experiences in terms of memory and computational costs. In addition, the model-based behavior adaptation has a finer granularity through better interpolation and extrapolation capabilities. The results also show that it is very important to suppresses the generation of behaviors for a behavior space that is not substantiated by gathered experiences. Otherwise, well performing behaviors, according to the model, are derived and executed, which at the end show worse performance. The uncertainty estimation of SOGPR turns out to be a vital property to punish behaviors with almost no supporting data. Of course, the continuous integration of new experiences improves the model and finally avoids making the same mistake again. Nevertheless, in high-dimensional input spaces, this trial and error paradigm is not efficient. Adding to that, the limited kernel size automatically causes a loss of information when new divergent experiences are incoming. Consequently, the higher the dimension of the input space and the more local minima on the real (unknown) behavior performance map exist, the more kernels are needed. The resulting increase in time can easily contradict with application-specific real time constraints. In this application for example, where the maximum time to get a solution is limited to 1 s, the search for the optimal behavior not always converge resulting in the execution of non-optimal solutions, which shows that the limit in complexity is reached.

The case-based inference approach has its strengths here. First, the derived behaviors are very closely orientated on well-performing, already evaluated behaviors. Second, the dimension of the behavior space has no influence on the case retrieval, which makes it very efficient for high-dimensional behavior spaces. The results from the experience series prove, that a reliable behavior adaptation is realized. Here, best accuracy in following motion commands is achieved, except for comparably high

speeds. For the sake of other performance features, like stability and avoiding vibrations, the maximum desired speed is avoided, which can be regarded as intrinsic safety feature.

Even though the model-based approach is not achieving the best results in this particular application, the underlying behavior-performance model is providing interesting insides to explain why certain behaviors are applied for certain contexts. The comparison between single leg walking and trot shows that on the one side, single leg walking is preferred for low and moderate speeds because it causes less vibrations. On the other side, trot can reach higher velocities and maintains a better energy efficiency.

In this particular application, the overall data-driven approach shows its advantage by automatically switching between gaits. The switch is intrinsically triggered by the action context change and depends on the gathered experiences and thus, no hard coded thresholds nor expert knowledge is required. Also, decisions like making rather long and slow steps or small ones with high frequency, which both result in the same speed, can directly be decided by the model. However, only few aspects are analyzed within this experiment. The fact that they provide reasonable results that are conform with available expert knowledge, suggests that further aspects and problems from other domains where expert knowledge is missing can be analyzed in the same way.

Changing the performance prioritization turns out to be an efficient instrument to manipulate the adaptation process. It allows to favor specific performance criteria without generating non-working solutions, which could be the case if behavior parameters were manually tuned. Even though they can be adapted online, usually one initial setup for a specific use case should be sufficient. Due to the application-specific representation, tuning them without being an expert on the details of the motion control is possible.

The direct application of simulation-trained knowledge bases on the physical system is also possible. Even though differences between both worlds are noticeable in this application, stable and efficient walking behaviors can be executed due to a comparable small simulation reality gap. Thus, depending on the simulation reality gap, it opens the possibility to pre-train a model or case base in simulation with few effort and then apply it on the physical system and adapt it in later on. By analyzing the prediction accuracy of real samples, one has the possibility to quantify the simulation reality gap. Due to the quick adaptation capability of SOGPR, it is possible to acquire a representable model for the physical system with a few hundred samples. Thus, experience-based behavior adaptation is an efficient method to cross existing simulation reality gaps. In addition, its good transferability opens up the possibility to manage individual characteristics of single systems within a swarm of robots.

Chapter 8

Conclusion and Outlook

This chapter summarizes the results of the thesis and presents the lessons learned. It closes with discussing the limits of this work and pointing to future directions.

8.1 Discussion and Analysis

The goal of this thesis is to increase the autonomy of robots by adapting their behavior with respect to the current context, i.e. the desired action context (target values with performance prioritization) as well as the state context (state of the environment and the robot itself). The proposed approach fulfills this goal by collecting experiences in form of evaluated behavior parameters in detected state contexts throughout the lifetime of the robot and using them to derive situation-specific behavior parameters for the action-executing motion control with the help of AI techniques. In this thesis, it is shown that the developed approach has several advantages, including:

- The actual motion control is handled as a black box achieving a high transferability across different robots and control approaches.
- The required experiences are generated by the robot itself through self-evaluating executed actions of its parameterized motion control according to application-specific performance criteria. Consequently, the robot continuously learns through its interaction with the world and simultaneously improves its behavior adaptation.
- The adaptation reacts on continuous and discrete context changes while also tracking long-term behavior performance changes.
- The autonomous behavior adaptation can be fine-tuned to current needs because multiple performance criteria are considered and the behavior performance prioritization can be changed during runtime.

The walking pattern adaptation of the kinematically complex robots SpaceClimber and Charlie is selected as exemplary target application, which demonstrates the autonomous behavior adaptation on complete systems using the full sensor and actuator suit. A reactive motion control is introduced that generates stable walking trajectories for various robotic systems and environments by configuring its parameters accordingly, which define posture, walking gait, and influence of reflexes. Thereby, each parametrization is applicable for a limited context range in terms of traversable slopes and obstacles. The simultaneous adaptation of behavior parameters leads to different postures and walking gaits allowing the handling of larger context ranges. By evaluating the traversed maximum obstacle height, terrain roughness, and slope as state context features as well as velocity and posture characteristics in combination with stability and efficiency metrics as performance features, experiences are generated for executed walking behaviors. The experience-based behavior adaptation uses them to derive stable and energy efficient walking behaviors for varying terrain types. Thus, the complex and expert knowledge requiring task of continuously tuning control-specific behavior parameters is taken from the operator. Regardless of the terrain, he or she can focus on setting desired motion commands or being out of the control loop by using a path planner and trajectory follower resulting in a fully autonomous control.

Two inference strategies that implement the actual behavior adaptation are developed and analyzed within this thesis. The case-based behavior derivation is a lazy learning strategy that incorporates new experiences directly in an efficient way in a case base, which is used for reasoning during runtime. The extensive analysis through testing traversals of demanding obstacles courses with SpaceClimber in simulation and in the real world revealed several pros and cons. The implemented case-based selection process has the advantage that the behavior selection is decoupled from the actual behavior representation, i.e. different parameterizations or even motion control implementations can be compared and selected. Furthermore, impossible commands are not tried to be executed, instead, the behavior that comes closest while maintaining stability and efficiency constraints is selected. This is an intrinsic security feature that solves problems such as an operator not knowing exactly how fast the robot can run over various obstacles. However, simply collecting as many experiences as possible is not sufficient for the case-based approach to support long-term autonomous systems. On the one side, the memory and computational costs grow linearly with newly evaluated behaviors. And on the other side, because there are no dependencies between behaviors, the adaptation capability of the case base decreases with increasing number of cases. If events happen that cause a severe correlation change, e.g. a broken leg, every single behavior needs to be reevaluated to regain an update-to-date case base.

To address the aforementioned drawbacks in an alternative way, a second behavior inference approach is implemented and analyzed, the model-based behavior derivation. In this approach, an optimization loop is implemented which uses a regression model to predict the performance of behaviors with respect to the current state context. A parameter optimizer generates new test behaviors and uses their predicted performance for a guided search through the behavior space towards an optimal behavior for the current context. The action context is considered in the cost function to generate a scalar fitness value from the predicted behavior performance features. To model the performance estimator, SOGPR is developed within this thesis, an incremental GPR featuring a limited model size and intelligent update schemes to provide quick and reliable predictions for few and unlimited noisy input data. The detailed experimental evaluation shows that SOGPR provides accurate and fast predictions for small and large data sets compared to other regression techniques, especially when noisy data is used for training. The intelligent update scheme avoids major accuracy drops despite the limited model size and the need of forgetting experiences. Furthermore, the trained model is capable to adapt to slight and severe input-output correlations, which is crucial to be applicable within future long-term autonomous systems.

The overall model-based approach is evaluated in the example of adapting the walking gait of Charlie for varying velocity commands and performance prioritizations. The results show that applicable behavior parameters are generated. However, the chosen parameterizations reveal that the extrapolating capability tends to generate behaviors that exploit regions of the input space that suggest good performance but are not grounded by input data. Thus, a very important characteristic of SOGPR is vital, the capability to provide an uncertainty estimation for every performance prediction. Incorporating the uncertainty in the cost function damps the behavior parameter exploration and guides the search to already evaluated behaviors, generating more reliable solutions. The obtained prediction model can additionally provide vital information to explain the autonomous behavior adaptation by showing influences of specific behavior parameters and state context attributes with respect to selected performance criteria. The provided uncertainty estimation helps to interpret the reliability of the model's performance estimation.

Comparing the case- and model-based approach for the gait adaptation of Charlie, training and loading the model is much faster than generating or loading the case base, while the execution time is similar for around 50000 experiences. Nevertheless, the case-based inference generates more reliable walking behaviors giving a hint that the utilized motion control requires precisely coordinated parameter combinations which are not perfectly represented by the chosen model size even though the adaptation result is more fine grained. In summary, both developed approaches, the

case-based and the model-based, have their pros and cons. The case-based behavior adaptation has its strengths in applications with a small amount of experiences and a large behavior space or in applications that need to choose between different control approaches. In contrast, the model-based approach has its strengths where many experiences for few behavior parameters are available but fine-grained adaptations are required.

Regardless of the specific inference strategy, the operator has the possibility to fine-tune the autonomous behavior adaptation for changing needs or different applications by changing the performance prioritization of the utilized performance features. The experimental results show that prioritizing one or multiple performance features improves the overall behavior in this perspective on the costs of low priority features. This fine-tuning is intuitive for the operator because it is represented by application-specific metrics. The conducted experiments show that providing only good experiences can lead to wrong adaptations because it is not only important to know that a behaviour works well in a specific context. It is just as important to know that it works badly in other contexts. Knowledge bases that uniformly cover the action and state context space lead to more robust and reasonable behavior adaptations than knowledge bases which have a lack of data in the state context and behavior parameter space. Consequently, the continuous integration of new experiences is crucial to obtain and maintain a knowledge base that allows near optimal behavior adaptations for various contexts.

8.2 Outlook

The experience-based behavior adaptation shows promising results in the selected target application to improve the execution of one type of action. In future, further work on the overall integration of this concept has to be made. First, managing several knowledge bases for different action types is theoretically applicable but needs to be proven in practice. It has to be decided for every action type whether a default behavior is sufficient or a mapping from context to an optimal behavior justifies the extra effort of defining and implementing state context and performance criteria and to test and evaluate behaviors for collecting experiences. Second, the gathered experiences consider only a selection of behavior parameters but surely many more exist within the different levels of the whole robot control. For instance, if the control parameters of the underlying joint controllers are adapted, which has a strong influence on the dynamic properties of the joints, the performance of every behavior will dramatically change although the considered behavior parameters stay constant. Consequently, many more meta information need to be stored with each knowledge base to judge whether the contained experiences are still valid or obsolete. This will

also require a more sophisticated handling of multiple knowledge bases including a versioning system. Alternatively, the proposed approach allows to retrain existing knowledge bases after each intervention in the robot's software and hardware components. The automated behavior evaluation framework can facilitate this process, if artificial experiences need to be updated. Yet, the update process of physical experiences can be very costly in terms of time and utilization of hardware resources. Consequently, using the experience-based adaptation concept plays out its strengths on fully developed systems, where soft- and hardware adaptations are seldom.

Further research can also be conducted on the application-specific selection of state context and performance features. In this work, metrics are selected that process the multi-modal sensor inputs and their post-processed information to provide cost-efficient numeric estimates, which roughly characterize the situation and the outcome of a behavior. More precise computations might improve the performance of the generated behaviors on the costs of higher computational effort. However, the reliability of the selected metrics can be derived after the knowledge base generation by evaluating the correlation between state context and performance features and by examining the prediction accuracy of specific performance features. Using autoencoders can be a promising way to automatically generate features, which on the one hand would reduce the manual implementation effort. On the other hand, this approach carries the risk, that a human-meaningful representation of state context and performance features is lost, which hampers the understanding and verification of the autonomous behavior adaptation.

In any case, continuously evaluating its own executed behavior is vital to constantly learn and improve its actions. Countless applications in times of Big Data show that data is the basis for tackle current and future challenges. Only a solid data base allows the application of AI in domains where analytical and engineering approaches reach their limits. The recorded knowledge bases of this work contribute to this development. While the gathered data is used here for autonomous reconfiguration, other utilizations are conceivable, such as using the model-based performance prediction for a more accurate cost estimation within mission and path planning or to determine the wearout by comparing the performance of the initially acquired and actual experiences. Due to the data-driven knowledge of the nominal performance in various contexts, performance drops could be detected by the robot itself and handled by higher layers or an operator enabling an appropriate reaction at early stages. The exciting question for the future is how this data-driven approach can be used in the various levels of robot control in an automated way. This is not only a challenge from the software engineering perspective, but also the triggering of "play time" to efficiently explore and evaluate the robot capabilities while not risking any mission success raises further research questions.

Appendix A

Additional Experimental Data

A.1 Regression Comparison between SOGPR, SONIG, and LGP

The experimental results below show the regression accuracy and computational effort as evaluated in Section 6.3.2 for SONIG and LGP. The corresponding hyperparameters were chosen according to their documentation. All hyperparameters stayed constant for all investigated data sets. Here, the prediction time corresponds to the time that was needed to predict outputs for the entire data set. As summarized in Table A.1, Table A.2, Table A.3, and Table A.4, SOGPR has superior performance. LGP is only more accurate on the SARCOS data set, but need much more time for the prediction. SONIG has in general the fastest prediction time but less prediction accuracy. In addition, the training process takes much longer.

Table A.1: Regression comparison on BARRETT data set

Algorithm	SOGPR	SONIG	LGP
MSE ($\cdot 10^{-2}$)	0.48	4.44	6.69
Training Time (s)	27.85	1926.68	74.24
Prediction Time (s)	1.30	0.41	167.63

Table A.2: Regression comparison on SARCOS data set

Algorithm	SOGPR	SONIG	LGP
MSE ($\cdot 10^{-2}$)	1.31	8.65	0.87
Training Time (s)	84.90	2747.03	53.98
Prediction Time (s)	2.28	0.94	368.25

Table A.3: Regression comparison on Charlie data set

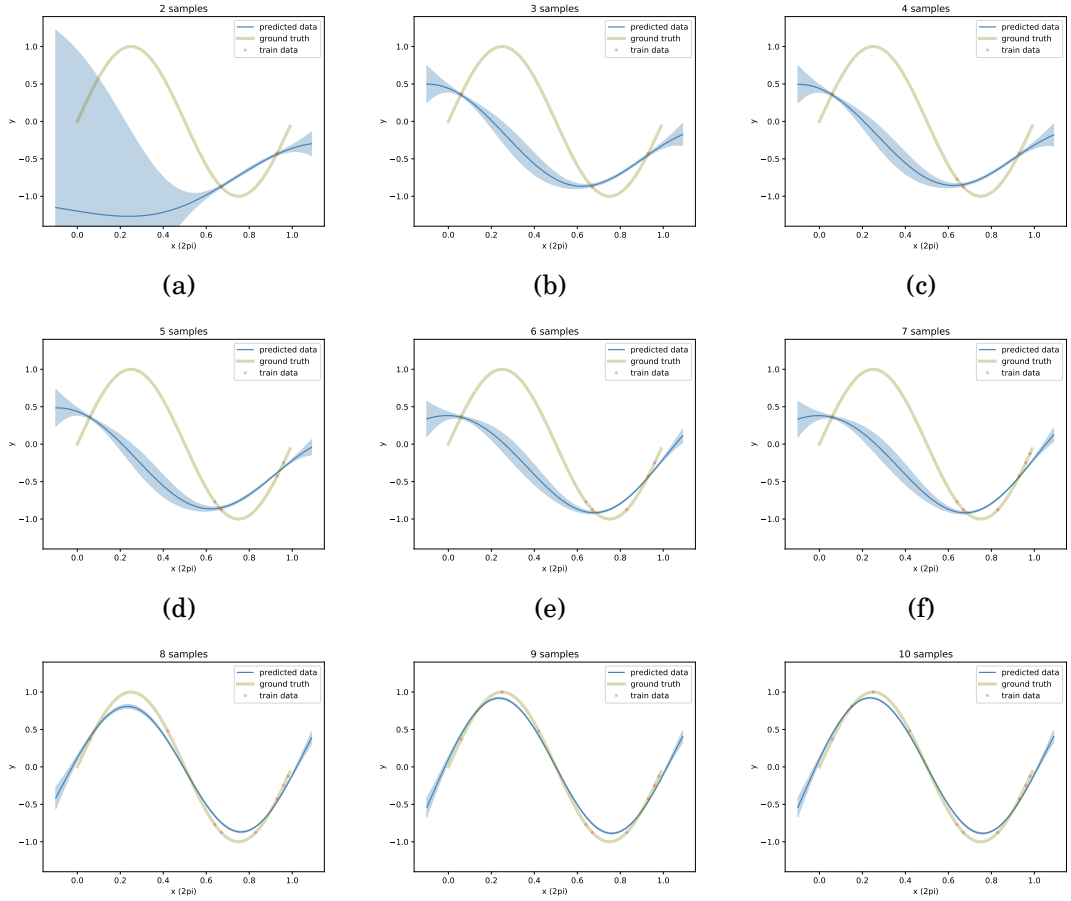
Algorithm	SOGPR	SONIG	LGP
MSE ($\cdot 10^{-2}$)	2.86	43.80	13.00
Training Time (s)	3.65	141.98	63.44
Prediction Time (s)	0.18	0.08	2.44

Table A.4: Regression comparison on Spaceclimber data set

Algorithm	SOGPR	SONIG	LGP
MSE ($\cdot 10^{-2}$)	15.47	39.07	31.44
Training Time (s)	45.10	1719.33	311.48
Prediction Time (s)	2.16	0.45	406.32

A.2 SOGPR Model Development with Incoming Training Data

This experiment shows the incremental SOGPR model development for different amounts of incoming training samples. X and y coordinates of a sine wave were used as input samples. A certain amount were used for training and the resulting model was subsequently used to predict outputs for inputs that evenly covered the whole period. One can see in Figure A.1 that the model is heavily influenced by every single sample update. Using two samples, the model prediction meets the target function only directly at the input data. At these spots, the uncertainty is quite low, whereas moving away from the spots increases the uncertainty dramatically. The distinctive sine curve cannot be predicted when using the first seven samples for training. This is because between $x = 0.1$ and $x = 0.6$, no samples are provided, which is also reflected by the higher uncertainty estimation. The eight's sample provides data within this region. Thus, SOGPR adapts and is finally capable to model the target sine curve with more accuracy. Even though there is still some error left at the sine peaks when utilizing ten samples, the overall trend shows that the accuracy is constantly improving with further samples.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure A.1: SOGPR model development - prediction and uncertainty estimation on the example of a sine curve

List of Figures

List of Tables

Acronyms

3D	three-dimensional
AI	artificial intelligence
ANN	artificial neural network
BAGEL	Biologically inspired Graph-Based Language
CBR	case-based reasoning
CMA-ES	Covariance Matrix Adaptation using Evolutionary Strategy
COM	center of mass
CPG	central pattern generator
DFKI	German Research Center for Artificial Intelligence
DMP	dynamic motion primitives
DOF	degrees of freedom
EDR	elevation and depression reflex
FNN	Feedforward Neural Network
GPR	Gaussian Process Regression
IMU	inertia measurement unit
LGP	Local Gaussian Process regression
LWPR	Locally Weighted Projection Regression
MAE	mean absolute error
MSE	mean squared error
PSO	Particle Swarm Optimization

RBF	Radial Basis Function
RL	Reinforcement Learning
RMSE	root mean squared error
ROI	region of interest
SLAM	simultaneous localization and mapping
SIGPR	Sparse Incremental Gaussian Process Regression
SOGPR	Stream Online Gaussian Process Regression
SONIG	Sparse ONlIne Gaussian process regression
SVR	Support Vector Regression
WBC	whole body control
ZMP	zero moment point

Symbols

To have a homogeneous mathematical notation, the matrix notation is used throughout this thesis. A vector is represented by a lowercase bold character, whereas a Matrix is denoted by a bold capitalized bold letter. A single variable is usually written in a lowercase letter whereas a capital letter stands for a quantity.

Overall Approach

$\boldsymbol{a}$	desired action (desired action and meta performance features)
$\boldsymbol{b}$	behavior (motion control parameters)
$\boldsymbol{b}_s$	sample of behavior parameters that were evaluated
$\hat{\boldsymbol{b}}$	supposedly best behavior with respect to action and state context
$\boldsymbol{p}$	behavior performance
$\boldsymbol{p}_s$	behavior performance sample
$\boldsymbol{w}$	performance prioritization (weights for the action and meta performance features)
$\boldsymbol{z}$	state context (external and internal states)
$\boldsymbol{z}_s$	state context sample
A	number of action and meta performance features
B	number of parameters representing a behavior
Z	number of state context features

Regression

k_{mn}	kernel matrix element of row m and column n, representing the computed kernel function between $\boldsymbol{x}_m$ and $\boldsymbol{x}_n$
$\boldsymbol{k}$	kernel vector
l	length scale of Gaussian kernel
$\boldsymbol{m}$	vector of weighting factors
p_*	predicted performance value for a specific behavior parameter and state context input
u_*	prediction uncertainty for a query point
$\boldsymbol{x}_s$	sample input vector
$\boldsymbol{x}_*$	input query point for a prediction
$\boldsymbol{y}$	output samples
y_s	sample output
y_*	predicted output value for a query point
$\boldsymbol{I}$	identity matrix
$\boldsymbol{K}$	kernel matrix
S	number of samples of a data set
$\boldsymbol{X}$	input samples
$\boldsymbol{\epsilon}$	error of prediction
σ_f	scale of Gaussian kernel
σ_n	standard deviation of expected signal noise
σ_*	root of prediction uncertainty
$\boldsymbol{\Phi}$	transformation from lower to higher dimensions

Target Application

$\boldsymbol{a}_{robot}$	acceleration vector
f_{com_cpg}	scaling factor for the walking gait specific body shifts
f_{ps}	shortage of the lift and down phase
$\boldsymbol{g}$	gravity vector
h_{step}	step height
l	limb name
$\boldsymbol{l}_{step}$	longitudinal and lateral step length
$\boldsymbol{o}_{body}$	desired translation between $LocoFrame$ and $COMFrame$
$\boldsymbol{o}_{l}$	desired foot position offset of every limb
$\boldsymbol{p}_{efficiency}$	performance features characterizing efficiency
$\boldsymbol{p}_{posture}$	performance features characterizing posture
$\boldsymbol{p}_{stability}$	performance features characterizing stability
$\boldsymbol{p}_{vel}$	performance features characterizing the robot's movement speed
$\boldsymbol{q}_{ref}$	quaternions of reference rotation
$\boldsymbol{q}_{*}$	quaternions of actual rotation
$\triangle\overline{s}_{arc}$	average arc length of a planar position change within a time step
$\boldsymbol{s}_{base}$	distance between front and rear limbs and left and right limbs
t_{cycle}	time of a complete step cycle
t_{down}	portion of step cycle to lower the limb
t_{inc}	normalized time increment between to update hooks
t_{lift}	portion of step cycle to lift the limb
t_{period}	time update between two update hooks
t_{shift}	portion of step cycle to shift the limb
t_{stance}	duty factor, i.e foot ground contact time during a step cycle

t_{swing}	air time proportion during a step cycle
$\hat{t}_{swing}$	maximum air time of a leg for a statically stable walking pattern
$\boldsymbol{v}_{theo}$	theoretical velocity in longitudinal and lateral direction
N_{limbs}	number of limbs that are used for locomotion
${}^{Loco}_{COM}\boldsymbol{T}$	transformation from $LocoFrame$ to $COMFrame$
${}^{Loco}_{Robot}\boldsymbol{T}$	transformation from $LocoFrame$ to $RobotFrame$
${}^{Robot}_{COM}\boldsymbol{T}$	transformation from $RobotFrame$ to $COMFrame$
${}^{Loco}_{EE_l}\boldsymbol{T}$	transformation from $LocoFrame$ to EE_lFrame
${}^{Robot}_{EE_l}\boldsymbol{T}$	transformation from $RobotFrame$ to EE_lFrame
${}^{LB_l}_{EE_l}\boldsymbol{T}$	transformation from a LB_lFrame to $EndEffectorFrame$
${}^{Robot}_{LB_l}\boldsymbol{T}$	transformation from $RobotFrame$ to LB_lFrame
$\boldsymbol{\varphi}_{body}$	desired rotation between $LocoFrame$ and $COMFrame$
$\boldsymbol{\varphi}_{err}$	error between sensed and desired robot inclination
φ_{limbs}	phase shift of the diagonal limbs
$\boldsymbol{\varphi}_{ref}$	desired robot inclination in world frame
$\boldsymbol{\varphi}_{robot}$	sensed robot inclination in world frame
$\boldsymbol{\varphi}_{slope}$	surface inclination
ψ	turn angle within a step cycle

Case-Based Behavior Adaptation

$\boldsymbol{e}$	context evaluations of a case
$\hat{e}$	context evaluation of a case with highest state similarity
$\boldsymbol{p}_{\hat{e}}$	performance of the context evaluation with the highest state context similarity
$\overline{p}_i$	mean of i-th performance feature
$\overline{p^2}_i$	mean of the i-th squared performance feature

s_a	action context similarity of a case
s_c	overall case similarity
s_e	state context similarity of a specific context evaluation
s_z	state context similarity of a case
$\boldsymbol{w}_z$	state context feature weighting
$\boldsymbol{z}_e$	state context of a specific context evaluation
C	number of cases in a case base
D	number of discretized state context classes
K	number of nearest neighbors that are considered for case adaptation
$\sigma^2_{p_i}$	variance of the i-th performance feature

Model-Based Behavior Adaptation

$adapt_\delta$	boolean to decide whether a fixed or adaptive minimum sample importance is used
c	cost that is minimized by the optimization loop
$\boldsymbol{k}_s$	applied kernel function between current sample and all kernel elements
k_{ss}	applied kernel function between current sample and itself
n	actual kernel matrix size
t_{load}	overall time to load a trained model
$\bar{t}_{pred}$	average prediction time over all samples during model training
$\bar{t}_{solve}$	average time to find the supposedly best behavior for the current context
$\bar{t}_{train}$	average time to train a sample during model training
t_{train}	overall time to train all training samples
$type_{forget}$	forgetting strategy during replacement update
$type_\delta$	type of sample importance metric
$\boldsymbol{u}$	prediction uncertainties for all performance features

$\boldsymbol{x}_{sim}$	kernel element with the most similar input values with respect to the incoming sample
y_{sim}	corresponding output of the kernel element with the most similar input values with respect to the incoming sample
$\boldsymbol{K}_{new}$	new kernel matrix
$\boldsymbol{K}_{old}$	previous kernel matrix
$\boldsymbol{K}_r$	residual sub matrix of the kernel matrix
$\boldsymbol{K}_{r^{-1}}$	residual sub matrix of the inverse kernel matrix
M	maximum kernel matrix size
$\boldsymbol{X}_n$	input matrix of all kernel elements
α_{mem}	memorizing factor when sample learning is triggered
$\boldsymbol{\delta}$	importance of kernel elements
δ_{min}	minimum sample importance to trigger case learning
δ_s	sample importance

Target Application Inputs and Outputs

In the following, summarizing lists of behavior parameters, state context features, and performance features are provided for the selected target application. The corresponding name in the software or in the plots together with a brief description is given for every input or output.

Behavior Parameters

`body_rot_roll`	desired body roll (rad)
`body_rot_pitch`	desired body pitch (rad)
`body_rot_yaw`	desired body yaw (rad)
`body_shift_x`	desired body shift in longitudinal direction (m)
`body_shift_y`	desired body shift in lateral direction (m)
`body_shift_z`	desired body height (m)
`com_cpg_scale`	desired scale of the walking gait specific body shifts
`phase_shift`	desired phase shift of the diagonal limbs
`phase_shortage`	desired shortage of the lift and down phase
`step_length_x`	desired longitudinal step length (m)
`step_length_y`	desired lateral step length (m)
`step_length_z`	desired step height (m)
`step_base_x`	desired distance between front and rear limbs (m)
`step_base_y`	desired distance between left and right limbs (m)

t_cycle	desired time of a complete step cycle (s)
t_down	desired portion of step cycle to lower the limb
t_lift	desired portion of step cycle to lift the limb
t_shift	desired portion of step cycle to shift the limb
turn_rate	desired turn angle within a step cycle (rad)

State Context Features

obstacle_height	peak height difference in ROI
roughness	standard deviation of ROI heights to least square plane
slope_x	slope in longitudinal direction
slope_y	slope in lateral direction

Behavior Performance Features

body_height	z-distance from $LocoFrame$ to $RobotFrame$
body_width	greatest y-distance between all feet
dsa	dynamic stability angle
epd	energy per distance
power	accumulated power consumption of all electronic devices
ssm	static stability margin
velocity_rot	angular velocity around z-axis
velocity_x	longitudinal velocity
velocity_y	lateral velocity
vibration	angular velocity of body around x- and y-axis

Bibliography

[Aamodt and Plaza, 1994] Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1):39–59.

[Aha et al., 2005] Aha, D. W., Molineaux, M., and Ponsen, M. (2005). Learning to win: Case-based plan selection in a real-time strategy game. In *International Conference on Case-Based Reasoning*, pages 5–20. Springer.

[Albiez, 2007] Albiez, J. (2007). *Verhaltensnetzwerke zur adaptiven Steuerung biologisch motivierter Laufmaschinen*. GCA-Verlag.

[Amodei et al., 2016] Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., et al. (2016). Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182.

[Arbib, 1992] Arbib, M. A. (1992). Schema theory. *The Encyclopedia of Artificial Intelligence*, 2:1427–1443.

[Argall et al., 2009] Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

[Bartsch, 2014] Bartsch, S. (2014). Development, control, and empirical evaluation of the six-legged robot spaceclimber designed for extraterrestrial crater exploration. *KI-Künstliche Intelligenz*, 28(2):127–131.

[Bartsch et al., 2012] Bartsch, S., Birnschein, T., Römmermann, M., Hilljegerdes, J., Kühn, D., and Kirchner, F. (2012). Development of the six-legged walking and climbing robot spaceclimber. *Journal of Field Robotics*, 29.

[Bartsch et al., 2016] Bartsch, S., Manz, M., Kampmann, P., Dettmann, A., Hanff, H., Langosz, M., von Szadkowski, K., Hilljegerdes, J., Simnofske, M., Kloss, P.,

et al. (2016). Development and control of the multi-legged robot mantis. In *Proceedings of ISR 2016: 47st International Symposium on Robotics*, pages 1–8. VDE.

[Bijl et al., 2017] Bijl, H., Schön, T. B., van Wingerden, J.-W., and Verhaegen, M. (2017). System identification through online sparse gaussian process regression with input noise. *stat*, 1050:16.

[Brinkmann et al., 2019] Brinkmann, W., Cordes, F., Koch, C. E. S., Wirkus, M., Dominguez, R., Dettmann, A., Vögele, T., and Kirchner, F. (2019). Space robotic systems and artificial intelligence in the context of the european space technology roadmap.

[Brinkmann et al., 2020] Brinkmann, W., Dettmann, A., Danter, L. C., Schulz, C., Stark, T., Brandt, A., and Kirchner, F. (2020). Enhancement of the six-legged robot mantis for assembly and construction tasks in lunar mission scenarios within a multi-robot team. In *In 15th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS '20)*. Online.

[Brooks, 1986] Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

[Burattini and Rossi, 2010] Burattini, E. and Rossi, S. (2010). Periodic activations of behaviours and emotional adaptation in behaviour-based robotics. *Connection Science*, 22(3):197–213.

[Calinon et al., 2013] Calinon, S., Kormushev, P., and Caldwell, D. G. (2013). Compliant skills acquisition and multi-optima policy search with em-based reinforcement learning. *Robotics and Autonomous Systems*, 61(4):369–379.

[Christensen et al., 2015] Christensen, D. J., Andersen, J. C., Blanke, M., Furno, L., Galeazzi, R., Hansen, P. N., and Nielsen, M. C. (2015). Collective modular underwater robotic system for long-term autonomous operation. In *ICRA 2015*.

[Cordes et al., 2011] Cordes, F., Dettmann, A., and Kirchner, F. (2011). Locomotion modes for a hybrid wheeled-leg planetary rover. In *2011 IEEE International Conference on Robotics and Biomimetics*, pages 2586–2592. IEEE.

[Cruse et al., 1998] Cruse, H., Kindermann, T., Schumm, M., Dean, J., and Schmitz, J. (1998). Walknet - a biologically inspired network to control six-legged walking. *Neural networks*, 11(7-8):1435–1447.

[Csató and Opper, 2002] Csató, L. and Opper, M. (2002). Sparse on-line gaussian processes. *Neural computation*, 14(3):641–668.

[Cully et al., 2015] Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B. (2015). Robots that can adapt like animals. *Nature*, 521(7553):503.

[Cummings, 2001] Cummings, B. (2001). *Human Anatomy and Physiology Laboratory Manual (Benjamin Cummings Series in Human Anatomy and Physiology)*, volume 7. Benjamin Cummings Publishing Company, Subs of Addison Wesley Longman, Inc.

[de Gea Fernández et al., 2017] de Gea Fernández, J., Mronga, D., Günther, M., Knobloch, T., Wirkus, M., Schröer, M., Trampler, M., Stiene, S., Kirchner, E., Bargsten, V., et al. (2017). Multimodal sensor-based whole-body control for human–robot collaboration in industrial settings. *Robotics and Autonomous Systems*, 94:102–119.

[Denny et al., 2018] Denny, J., Sandström, R., and Amato, N. M. (2018). A general region-based framework for collaborative planning. In *Robotics Research*, pages 563–579. Springer.

[Dettmann et al., 2015a] Dettmann, A., Bartsch, S., and Kirchner, F. (2015a). An experience-based interface for abstracting the motion control of kinematically complex robots. In *Symposium on Advanced Space Technologies in Robotics and Automation*.

[Dettmann et al., 2015b] Dettmann, A., Born, A., Bartsch, S., and Kirchner, F. (2015b). Experience-based adaptation of locomotion behaviors for kinematically complex robots in unstructured terrain. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4504–4511. IEEE.

[Dettmann et al., 2018a] Dettmann, A., Kuehn, D., Van Opdenbosch, D., Stark, T., Peters, H., Natarajan, S., Kasperski, S., Boeckmann, A., Garcea, A., Steinbach, E., et al. (2018a). Exploration in inaccessible terrain using visual and proprioceptive data. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*.

[Dettmann et al., 2018b] Dettmann, A., Kühn, D., and Kirchner, F. (2018b). Improved locomotion capabilities for quadrupeds through active multi-point-contact feet. In *2018 4th International Conference on Control, Automation and Robotics (ICCAR)*, pages 156–161. IEEE.

[Dettmann et al., 2020] Dettmann, A., Kühn, D., and Kirchner, F. (2020). Control of active multi-point-contact feet for quadrupedal locomotion. *International Journal of Mechanical Engineering and Robotics Research*, 9(4):481–488.

[Dettmann et al., 2014] Dettmann, A., Langosz, M., von Szadkowski, K. A., and Bartsch, S. (2014). Towards lifelong learning of optimal control for kinematically complex robots. In *Proceedings of IEEE ICRA14 Workshop on Modelling, Estimation, Perception and Control of All Terrain Mobile Robots*.

[Dettmann et al., 2016] Dettmann, A., Sonsalla, R., and Moessner, J. (2016). Modular test course - possibilities, parts, and how-to use it. In Florian, C., Christensen, L., and Kirchner, F., editors, *Proceedings of the RIC Project Day Workgroups Locomotion & Mobility and Navigation & Planning*, volume 16-03 of *series DFKI Documents*, pages 7–12. DFKI GmbH.

[Dey, 2001] Dey, A. K. (2001). Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7.

[Drucker et al., 1997] Drucker, H., Burges, C. J., Kaufman, L., Smola, A. J., and Vapnik, V. (1997). Support vector regression machines. In *Advances in neural information processing systems*, pages 155–161.

[Eberhart and Kennedy, 1995] Eberhart, R. and Kennedy, J. (1995). Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks*, volume 4, pages 1942–1948. Citeseer.

[Englsberger et al., 2011] Englsberger, J., Ott, C., Roa, M. A., Albu-Schäffer, A., and Hirzinger, G. (2011). Bipedal walking control based on capture point dynamics. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4420–4427. IEEE.

[Englsberger et al., 2014] Englsberger, J., Werner, A., Ott, C., Henze, B., Roa, M. A., Garofalo, G., Burger, R., Beyer, A., Eiberger, O., Schmid, K., et al. (2014). Overview of the torque-controlled humanoid robot toro. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 916–923. IEEE.

[Fabisch et al., 2015] Fabisch, A., Metzen, J., Krell, M., and Kirchner, F. (2015). Accounting for task-difficulty in active multi-task robot control learning. *Kunstliche Intelligenz*.

[Fondahl et al., 2012] Fondahl, K., Kuehn, D., Beinersdorf, F., Bernhard, F., Grimminger, F., Schilling, M., Stark, T., and Kirchner, F. (2012). An adaptive sensor foot for a bipedal and quadrupedal robot. In *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 270–275. IEEE.

[Gassmann, 2007] Gassmann, B. (2007). *Modellbasierte, sensorgestützte Navigation von Laufmaschinen im Gelände*. PhD thesis, University Karlsruhe.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

[Grollman and Jenkins, 2008] Grollman, D. H. and Jenkins, O. C. (2008). Sparse incremental learning for interactive robot control policy estimation. In *2008 IEEE International Conference on Robotics and Automation*, pages 3315–3320. IEEE.

[Grotzinger et al., 2012] Grotzinger, J. P., Crisp, J., Vasavada, A. R., Anderson, R. C., Baker, C. J., Barry, R., Blake, D. F., Conrad, P., Edgett, K. S., Ferdowski, B., et al. (2012). Mars science laboratory mission and science investigation. *Space science reviews*, 170(1-4):5–56.

[Gu et al., 2008] Gu, J., Cao, Q., and Huang, Y. (2008). Rapid traversability assesment in 2.5 d grid based map on rough terrain. *Int. Journal of Advanced Robotics*, 5(4):389–394.

[Hansen and Ostermeier, 2001] Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, pages 159–195.

[Herzog et al., 2015] Herzog, A., Rotella, N., Schaal, S., and Righetti, L. (2015). Trajectory generation for multi-contact momentum control. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 874–880. IEEE.

[Hildebrand, 1965] Hildebrand, M. (1965). Symmetrical gaits of horses. *Science*, 150(3697):701–708.

[Hiltz et al., 2001] Hiltz, M., Rice, C., Boyle, K., and Allison, R. (2001). Canadarm: 20 years of mission success through adaptation.

[Hüllermeier, 2007] Hüllermeier, E. (2007). *Case-based approximate reasoning*, volume 44. Springer Science & Business Media.

[Hwangbo et al., 2019] Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., and Hutter, M. (2019). Learning agile and dynamic motor skills for legged robots. *arXiv preprint arXiv:1901.08652*.

[Jalali and Leake, 2013] Jalali, V. and Leake, D. (2013). A context-aware approach to selecting adaptations for case-based reasoning. *Modeling and Using Context*, pages 101–114.

[Johnson et al., 2015] Johnson, M., Shrewsbury, B., Bertrand, S., Wu, T., Duran, D., Floyd, M., Abeles, P., Stephen, D., Mertins, N., Lesman, A., et al. (2015). Team

ihmc's lessons learned from the darpa robotics challenge trials. *Journal of Field Robotics*, 32(2):192–208.

[Kajita et al., 2003] Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., and Hirukawa, H. (2003). Biped walking pattern generation by using preview control of zero-moment point. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 2, pages 1620–1626. IEEE.

[Katz et al., 2019] Katz, B., Di Carlo, J., and Kim, S. (2019). Mini cheetah: A platform for pushing the limits of dynamic quadruped control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6295–6301. IEEE.

[Kira and Arkin, 2004] Kira, Z. and Arkin, R. C. (2004). Forgetting bad behavior: memory for case-based navigation. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 4, pages 3145–3152. IEEE.

[Kolodner, 1992] Kolodner, J. L. (1992). An introduction to case-based reasoning. *Artificial Intelligence Review*, 6(1):3–34.

[Kolvenbach et al., 2018] Kolvenbach, H., Bellicoso, D., Jenelten, F., Wellhausen, L., and Hutter, M. (2018). Efficient gait selection for quadrupedal robots on the moon and mars. In *14th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2018)*. ESA Conference Bureau.

[Krell, 2018] Krell, M. M. (2018). Generalizing, decoding, and optimizing support vector machine classification. *arXiv preprint arXiv:1801.04929*.

[Kuehn et al., 2018] Kuehn, D., Dettmann, A., and Kirchner, F. (2018). Analysis of using an active artificial spine in a quadruped robot. In *2018 4th International Conference on Control, Automation and Robotics (ICCAR)*, pages 37–42. IEEE.

[Kuehn et al., 2020] Kuehn, D., Dettmann, A., and Kirchner, F. (2020). Design and evaluation of an active artificial spine in a hominid robot. *International Journal of Mechanical Engineering and Robotics Research*, 9(3):387–394.

[Kuehn et al., 2011] Kuehn, D., Grimminger, F., Beinersdorf, F., Bernhard, F., Burchardt, A., Schilling, M., Simnofske, M., Stark, T., Zenzes, M., and Kirchner, F. (2011). Additional dofs and sensors for bio-inspired locomotion: Towards active spine, ankle joints, and feet for a quadruped robot. In *2011 IEEE International Conference on Robotics and Biomimetics*, pages 2780–2786. IEEE.

[Kuehn et al., 2017] Kuehn, D., Schilling, M., Stark, T., Zenzes, M., and Kirchner, F. (2017). System design and testing of the hominid robot charlie. *Journal of Field Robotics*, 34(4):666–703.

[Kühn, 2016] Kühn, D. (2016). *Design and Development of a Hominid Robot with Local Control in Its Adaptable Feet to Enhance Locomotion Capabilities*. PhD thesis, Universität Bremen.

[Kuindersma et al., 2016] Kuindersma, S., Deits, R., Fallon, M., Valenzuela, A., Dai, H., Permenter, F., Koolen, T., Marion, P., and Tedrake, R. (2016). Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots*, 40(3):429–455.

[Langosz, 2018] Langosz, M. (2018). *Evolutionary Legged Robotics*. Doctoral dissertation, University of Bremen, Germany.

[Langosz et al., 2013] Langosz, M., Quack, L., Dettmann, A., Bartsch, S., and Kirchner, F. (2013). A behavior-based library for locomotion control of kinematically complex robots. In *Nature-Inspired Mobile Robotics*, pages 495–502. World Scientific.

[Laskov et al., 2006] Laskov, P., Gehl, C., Krüger, S., and Müller, K.-R. (2006). Incremental support vector learning: Analysis, implementation and applications. *Journal of machine learning research*, 7(Sep):1909–1936.

[Lawrence et al., 2003] Lawrence, N. D., Seeger, M., and Herbrich, R. (2003). Fast sparse gaussian process methods: the informative vector machine. In *Advances in Neural Information Processing Systems 15*. Citeseer.

[Leake et al., 1996] Leake, D. B. et al. (1996). *Case-based reasoning: Experiences, lessons & future directions*. AAAI press Menlo Park.

[Leake and Sooriamurthi, 2001] Leake, D. B. and Sooriamurthi, R. (2001). When two case bases are better than one: Exploiting multiple case bases. In *International Conference on Case-Based Reasoning*, pages 321–335. Springer.

[Lehman and Stanley, 2008] Lehman, J. and Stanley, K. O. (2008). Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336.

[Levine et al., 2016] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.

[Levitis et al., 2009] Levitis, D. A., Lidicker Jr, W. Z., and Freund, G. (2009). Behavioural biologists do not agree on what constitutes behaviour. *Animal behaviour*, 78(1):103–110.

[Liu et al., 2011] Liu, W., Principe, J. C., and Haykin, S. (2011). *Kernel adaptive filtering: a comprehensive introduction*, volume 57. John Wiley & Sons.

[Maes and Brooks, 1990] Maes, P. and Brooks, R. A. (1990). Learning to coordinate behaviors. In *AAAI*, volume 90, pages 796–802.

[McDonnell and Cunningham, 2006] McDonnell, N. and Cunningham, P. (2006). A knowledge-light approach to regression using case-based reasoning. In *European Conference on Case-Based Reasoning*, pages 91–105. Springer.

[McGhee and Iswandhi, 1979] McGhee, R. and Iswandhi, G. I. (1979). Adaptive locomotion of a multilegged robot over rough terrain. *Transactions on Systems, Man and Cybernetics*, 9(4):176–182.

[Metta et al., 2008] Metta, G., Sandini, G., Vernon, D., Natale, L., and Nori, F. (2008). The icub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pages 50–56. ACM.

[Michaud, 2002] Michaud, F. (2002). Emib–computational architecture based on emotion and motivation for in-tentional selection and configuration of behaviour-producing modules. *Cognitive Science Quarterly, 3*, 4:340–361.

[Morales et al., 2005] Morales, M., Tapia, L., Pearce, R., Rodriguez, S., Amato, N., et al. (2005). C-space subdivision and integration in feature-sensitive motion planning. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3114–3119. IEEE.

[Naish-Guzman and Holden, 2008] Naish-Guzman, A. and Holden, S. (2008). The generalized fitc approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1064.

[Nguyen-Tuong and Peters, 2010] Nguyen-Tuong, D. and Peters, J. (2010). Incremental sparsification for real-time online model learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 557–564.

[Nguyen-Tuong and Peters, 2011] Nguyen-Tuong, D. and Peters, J. (2011). Incremental online sparsification for model learning in real-time robot control. *Neurocomputing*, 74(11):1859–1867.

[Nguyen-Tuong et al., 2009] Nguyen-Tuong, D., Peters, J. R., and Seeger, M. (2009). Local gaussian process regression for real time online model learning. In *Advances in Neural Information Processing Systems*, pages 1193–1200.

[Pantic, 2005] Pantic, M. (2005). Introduction to machine learning & case-based reasoning. *London: Imperial College*.

[Papadopoulos and Rey, 2000] Papadopoulos, E. and Rey, D. a. (2000). The Force-Angle Measure of Tipover Stability Margin for Mobile Manipulators. *Vehicle System Dynamics*, 33(1):29–48.

[Parrella, 2007] Parrella, F. (2007). Online support vector regression. *Master's Thesis, Department of Information Science, University of Genoa, Italy*.

[Peng et al., 2017] Peng, X. B., Berseth, G., Yin, K., and Van De Panne, M. (2017). Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41.

[Radford et al., 2015] Radford, N. A., Strawser, P., Hambuchen, K., Mehling, J. S., Verdeyen, W. K., Donnan, A. S., Holley, J., Sanchez, J., Nguyen, V., Bridgwater, L., et al. (2015). Valkyrie: Nasa's first bipedal humanoid robot. *Journal of Field Robotics*, 32(3):397–419.

[Ranganathan et al., 2011] Ranganathan, A., Yang, M.-H., and Ho, J. (2011). Online sparse gaussian process regression and its applications. *IEEE Transactions on Image Processing*, 20(2):391–404.

[Rasmussen and Williams, 2006] Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian processes for machine learning*, volume 1. MIT press Cambridge.

[Römmermann et al., 2008] Römmermann, M., Edgington, M., Metzen, J. H., de Gea, J., Kassahun, Y., and Kirchner, F. (2008). Learning walking patterns for kinematically complex robots using evolution strategies. In Rudolph, G., Jansen, T., Beume, N., Lucas, S., and Poloni, C., editors, *Parallel Problem Solving from Nature – PPSN X*, pages 1091–1100, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Rönnau et al., 2014] Rönnau, A., Heppner, G., Nowicki, M., and Dillmann, R. (2014). Lauron v: A versatile six-legged walking robot with advanced maneuverability. In *2014 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 82–87. IEEE.

[Ros et al., 2006] Ros, R., Veloso, M., De Màntaras, R. L., Sierra, C., and Arcos, J. L. (2006). Retrieving and reusing game plays for robot soccer. In *European Conference on Case-Based Reasoning*, pages 47–61. Springer.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[Rumelhart et al., 1985] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

[Schirrmeister et al., 2017] Schirrmeister, R. T., Springenberg, J. T., Fiederer, L. D. J., Glasstetter, M., Eggensperger, K., Tangermann, M., Hutter, F., Burgard, W., and Ball, T. (2017). Deep learning with convolutional neural networks for eeg decoding and visualization. *Human brain mapping*, 38(11):5391–5420.

[Schneider et al., 2011] Schneider, A., Paskarbeit, J., Schäffersmann, M., and Schmitz, J. (2011). Biomechatronics for embodied intelligence of an insectoid robot. In *International Conference on Intelligent Robotics and Applications*, pages 1–11. Springer.

[Schölkopf et al., 1999] Schölkopf, B., Bartlett, P. L., Smola, A. J., and Williamson, R. C. (1999). Shrinking the tube: a new support vector regression algorithm. In *Advances in neural information processing systems*, pages 330–336.

[Schölkopf et al., 2002] Schölkopf, B., Smola, A. J., Bach, F., et al. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

[Schwendner and Joyeux, 2011] Schwendner, J. and Joyeux, S. (2011). Self localisation using embodied data for a hybrid leg-wheel robot. In *2011 IEEE International Conference on Robotics and Biomimetics*, pages 124–129. IEEE.

[Seeger et al., 2003] Seeger, M., Williams, C., and Lawrence, N. (2003). Fast forward selection to speed up sparse gaussian process regression. In *Artificial Intelligence and Statistics 9*, number EPFL-CONF-161318.

[Sentis and Khatib, 2006] Sentis, L. and Khatib, O. (2006). A whole-body control framework for humanoids operating in human environments. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2641–2648. IEEE.

[Snelson and Ghahramani, 2006] Snelson, E. and Ghahramani, Z. (2006). Sparse gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264.

[Spenneberg and Kirchner, 2007] Spenneberg, D. and Kirchner, F. (2007). The bio-inspired scorpion robot: design, control & lessons learned. *Climbing and Walking Robots: towards New Applications*, (October).

[Stulp et al., 2013] Stulp, F., Raiola, G., Hoarau, A., Ivaldi, S., and Sigaud, O. (2013). Learning compact parameterized skills with a single regression. In *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 417–422. IEEE.

[Triebel et al., 2006] Triebel, R., Pfaff, P., and Burgard, W. (2006). Multi-level surface maps for outdoor terrain mapping and loop closing. In *Int. Conf. on Intelligent Robots and Systems*.

[Urdiales et al., 2006] Urdiales, C., Perez, E. J., Vázquez-Salceda, J., Sànchez-Marrè, M., and Sandoval, F. (2006). A purely reactive navigation scheme for dynamic environments using case-based reasoning. *Autonomous Robots*, 21(1):65–78.

[Van Vaerenbergh et al., 2010] Van Vaerenbergh, S., Santamaría, I., Liu, W., and Príncipe, J. C. (2010). Fixed-budget kernel recursive least-squares. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 1882–1885. IEEE.

[Van Vaerenbergh et al., 2006] Van Vaerenbergh, S., Via, J., and Santamaría, I. (2006). A sliding-window kernel rls algorithm and its application to nonlinear channel identification. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 5, pages V–V. IEEE.

[Vijayakumar and Schaal, 2000] Vijayakumar, S. and Schaal, S. (2000). Locally weighted projection regression: An o (n) algorithm for incremental real time learning in high dimensional space. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, volume 1, pages 288–293.

[Vukobratović and Borovac, 2004] Vukobratović, M. and Borovac, B. (2004). Zero-moment point - thirty five years of its life. *International journal of humanoid robotics*, 1(01):157–173.

[Wu et al., 2015] Wu, R., Yan, S., Shan, Y., Dang, Q., and Sun, G. (2015). Deep image: Scaling up image recognition. *arXiv preprint arXiv:1501.02876*, 7(8).