DISS. ETH NO. 24402

EFFICIENT NUMERICAL OPTIMAL CONTROL FOR MOTION PLANNING AND CONTROL OF MOBILE ROBOTS

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

MICHAEL PETER NEUNERT

MSc. ETH ME

born on 04.08.1988
citizen of Germany

accepted on the recommendation of

Prof. Dr. Jonas Buchli (ETH Zurich), Examiner
Prof. Dr. Moritz Diehl (University of Freiburg), Co-examiner
Prof. Dr. Scott Kuindersma (Harvard University), Co-examiner

2017

Agile & Dexterous Robotics Lab
Department of Mechanical and Process Engineering
ETH Zurich
Switzerland

© 2017 Michael Peter Neunert. All rights reserved.

Abstract

Robotics holds high promises contributing solutions to our future society's challenges by taking over dangerous, physically intense work or fill up the lack of skilled labor resulting from population aging. To fulfill these promises, robots will need to solve increasingly complex tasks, requiring more versatile hardware designs with more degrees of freedom. Paired with fast development cycles, this poses a major challenge for robotic motion planning and control.

In traditional approaches, reasoning about the structure of the task and the robot itself has led to impressive performance. However, this involves manually re-engineering solutions almost entirely from scratch when the robot design or the given task changes. Therefore, there is a strong need for transferable methods. Ideally, the task or desired motion is specified on a high level and the planners and controllers then reason about the dynamics and kinematics of the robot to find a solution. In the first part of this work, we aim at creating such a general, transferable motion planning and control framework by using Numerical Optimal Control. We verify our method on three entirely different classes of robots namely walking, flying and ground robots. In order for our approach to leverage the full potential of each of the robots, we use full dynamic models of all systems. This allows us for solving complex motion planning tasks such as flying an Unmanned Aerial Vehicle through a narrow, tilted window or discovering different gait patterns on a quadruped robot. In contrast to many state-of-the-art approaches, we execute our planners and controllers on the physical systems and verify them with rigorous hardware experiments. Running Numerical Optimal Control online comes with several practical challenges such as limited computation time, model mismatches between simulation and hardware as well as estimating the state of the robot in its workspace. In the second part of this work, we address these application issues. We speed up our Numerical Optimal Control approach by applying Automatic Differentiation, a programming technique to obtain gradients of our systems more efficiently. We further investigate why off-the-shelf simulators perform poorly in evaluating feedback control performance. Lastly, we propose a lightweight, cost efficient estimation system which localizes a mobile robot relatively to its workspace.

Both parts of this thesis constitute a significant contribution in applying Numerical Optimal Control to physical systems. This work underlines the potential and versatility of such approaches by demonstrating them in hardware experiments on vastly different systems and tasks. Enabled by the efforts of an efficient implementation, we can run our Numerical Optimal Control solver online or even in Model Predictive Control fashion. Our approach outperforms state-of-the-art methods with respect to significantly lower computational time.

Zusammenfassung

Roboter haben das Potenzial zur Lösung zukünftiger, gesellschaftlicher Probleme beizutragen, indem sie für den Menschen gefährliche oder anstrengende Aufgaben übernehmen oder indem sie den Fachkräftemangel ausgleichen, der durch den demografischen Wandel entsteht. Um dieses Potenzial nutzbar zu machen, müssen Roboter immer komplexere Aufgabenstellungen lösen, was eine vielseitige Bauweise mit mehr Freiheitsgraden erforderlich macht. In Kombination mit der hohen Geschwindigkeit der Entwicklungszyklen entsteht so eine immense Herausforderung für die Bewegungsplanung und Regelung von Robotern.

Konventionelle Ansätzen in der Regelungstechnik nutzen die Struktur des Roboters sowie der Aufgabenstellung aus, um spezialisierte Lösungen zu erzeugen, welche beeindruckende Ergebnisse erzielen. Allerdings hat dies zur Folge, dass Lösungen von Grund auf neu erarbeitet werden müssen, sobald sich die Bauweise des Roboters oder die Aufgabenstellung ändert. Daher gibt es einen grossen Bedarf an universellen, transferierbaren Lösungen. Idealerweise wird die Aufgabenstellung oder die gewünschte Bewegung auf abstrakte Weise ausgedrückt und die Planungs- und Regelungsalgorithmen nutzen dann ihr Verständnis über die Kinematik und Dynamik des Roboters, um eine Lösung zu finden. Im ersten Teil dieser Arbeit streben wir es an, ein solch allgemeines, transferierbares Bewegungsplanungs- und Regelungskonzept mithilfe von numerischen Verfahren für optimale Steuerung zu entwicklen. Wir verifizieren unsere Methode mithilfe von drei verschiedenen Roboterklassen: laufende, fliegende und rollende Roboter. Damit unser Ansatz das Potenzial jeder dieser Robotertypen vollumfänglich ausnutzen kann, verwenden wir jeweils ein vollständig dynamisches Systemmodell. Das erlaubt es, komplexe Bewegunsplanungsaufgaben zu lösen, wie z. B. die Flugbahn eines unbemannten Luftfahrzeugs durch ein schmales, geneigtes Fenster zu berechnen oder verschiedenen Gangarten eines vierbeinigen Laufroboters zu erzeugen. Im Vergleich zum aktuellen Stand der Technik werten wir unsere Bewegungsplanung und -regelung auf physikalischen Systemen aus und testen sie umfänglich mit Experimenten. Solche numerischen, optimalen Steuerungen während der Systemlaufzeit auszuführen bringt einige praktische Herausforderungen mit sich, wie z. B. eine eingeschränkte Rechenzeit, Modellungenauigkeiten zwischen Simulation und physiaklischem System sowie die Notwendigkeit einer Zustandsschätzung des Roboters und seines Arbeitsbereichs. Den zweiten Teil dieser Arbeit widmen wir daher diesen praktischen Problemen. Hierbei verkürzen wir die Rechenzeit unserer numerischen, optimalen Steuerungsalgorithmen mithilfe von automatischer Differenzierung, einer Programmiertechnik, welche Gradienteninformationen zu unseren Systemen effizienter berechnet. Des Weiteren untersuchen wir, warum häufig verwendete Simulationsumgebungen nur bedingt dazu geeignet sind, um die Stabilität von Regelkreisen mit Rückführung zu untersuchen. Zu guter Letzt entwickeln wir ein kosteneffizientes Lokalisierungssystem für mobile Roboter, welches den Roboter relativ zu seinem Arbeitsbereich lokalisiert.

Beide Teile dieser Dissertation stellen entscheidende Fortschritte dar, numerische optimale Steuerungen auf physikalischen Systemen auszuführen. Diese Arbeit unterstreicht das Potenzial und die Vielseitigkeit solcher Ansätze mithilfe von Experimenten auf äusserst verschiedenen Systemen mit unterschiedlichen Aufgabenstellungen. Durch die effiziente Implementierung unserer Lösungsalgorithmen für die optimalen Steuerungsprobleme können wir diese Probleme zur Laufzeit lösen oder als modellprädiktive Regelung ausführen. Unser Lösungsansatz übertrifft dabei den Stand der Technik durch eine geringere Rechenzeit.

Acknowledgements

First and foremost, I would like to thank my supervisor Professor Dr. Jonas Buchli for his continuous support and trust throughout the entire duration of my PhD. He has been an extraordinary motivator and leader, creating an exceptionally creative environment, that allows all researchers to thrive both on a professional and personal level. Looking back a bit more than four years the Agile & Dexterous Robotics Lab (ADRL) consisted of not much of more than a few empty desks. Today, it is a highly reputable, multidisciplinary research lab with about twenty of the most talented people. This is a truly remarkable achievement and it has been a great honor and a pleasure to be part of this journey.

I would also like to thank my colleagues of the ADRL. Farbod Farshidian, who departed on this journey with me, has been a great support and a role model. Without him, this thesis would not have been feasible and I feel honored to have him as a colleague and friend. My thanks also go to our PostDocs, Dr. Thiago Boaventura and Dr. Diego Pardo, who shaped this lab with their tremendous experience. I would also like to thank my fellow PhD colleagues, Markus Giftthaler for all our technical discussions and the fruitful collaboration, especially towards the end of the thesis, Timothy Sandy for his comprehensive technical insight on all levels, Alexander Winkler for sharing his expertise on legged locomotion as well as setting benchmarks for all of us, and Johannes Pfleging for widening everyone's horizon. Furthermore, I would like to thank our highly talented engineers, Manuel Lussi for always being willing to go the extra mile, Markus Stäuble for his ability to cope with any challenge that comes his way, Edo Jelavić for his tremendous contribution on making HyQ a better platform, Asutosh Satapathy for providing fresh, outside perspectives to the lab, as well as Lukas Stadelmann and Julio López for their impressive work on Digital Fabrication. The whole ADRL team has made persuing my PhD a fantastic, joyful experience and also created a great atmosphere outside of the lab.

This work would have been impossible without the support of the Dynamic Legged Systems group of the ADVR at IIT Genova. I would like to thank Dr. Claudio Semini for engineering HyQ, a truly outstanding, well perfroming robot, as well as for supporting our activities, even actively with a screwdrivers in hand when needed. Also, my special thanks go to Dr. Marco Frigerio for his patience when dealing with our hardware and software issues as well as sharing his experience on these topics. My gratitude also goes to Victor Barasuol, Michele Focchi, Alex Posatskiy, Jake Goldsmith (ex ADVR, now Boston Dynamics) and all other ADVR members who contributed their time to help us advance in the field of quadruped locomotion.

Special thanks also go to my colleagues at the Autonomous Systems Lab (ASL) and the Robotic Systems Lab (RSL) at ETH Zurich. Foremost, I would like to thank Michael Blöesch, with whom I very much enjoyed collaborating with and who is an incredibly well balanced character, Fadri Furrer and Mina Kamel for wholeheartedly supporting our excursion to flying robots, Prof. Dr. Marco Hutter, Dr. Christian Gehring, Péter Fankhauser, Dario Bellicoso and all other members of the ASL/RSL for their fruitful exchanges inside and outside of the lab as well as the support during the ANYmal experiments.

I would also like to thank Dr. Gonzalo Rey, currently at Moog Inc., for sharing his experience on both a technical and a personal level as well as supporting the activities of our group. Furthermore, I would like to thank all the students I had the honor supervising and working together with over all these years. Special thanks go to Cédric de Crousaz for his tremendous work on our flying robots projects.

My gratitude also goes to my friends which have supported me on a personal level. Special thanks go to Simon Dössegger who has been a fantastic flatmate, a personal adviser and, most importantly, a great friend. Last but not least, I would also like to thank my family for their tremendous support throughout my life without which I would not have made it to this point, as well as my partner Isolda, who continuously brings joy to my life and who is backing me every single day.

June 2017 *Michael Peter Neunert*

Financial Support

This research has been funded through a Swiss National Science Foundation Professorship award to Jonas Buchli and the National Competence Center in Research (NCCR) Robotics.

Contents

Abstract i

Zusammenfassung iii

Acknowledgements v

Preface 1

1 Introduction 3
- 1.1 Motivation and Objectives . . . 5
- 1.2 Approach . . . 8
- 1.3 Hardware Platforms . . . 13
- 1.4 Related Work . . . 17

2 Contributions 31
- 2.1 Summary of Contributions . . . 31
- 2.2 Publication Overview – Part A . . . 34
- 2.3 Publication Overview – Part B . . . 38
- 2.4 Complete List of Publications . . . 41

3 Part A: Efficient Numerical Optimal Control 43
- Paper I: “Fast Non-Linear Model Predictive Control for Unified Trajectory Optimization and Tracking” . . . 45
- Paper II: “Trajectory Optimization Through Contacts and Automatic Gait Discovery for Quadrupeds” . . . 63
- Paper III: “Efficient Whole-Body Trajectory Optimization Using Contact Constraint Relaxation” . . . 83
- Paper IV: “Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds” . . . 97

4 Part B: Applying Numerical Optimal Control to Mobile Robots 117
- Paper V: “Automatic Differentiation of Rigid Body Dynamics for Optimal Control and Estimation” . . . 119
- Paper VI: “Why off-the-shelf physics simulators fail in evaluating feedback controller performance - a case study for quadrupedal robots” . . . 141

Paper VII: "An Open Source, Fiducial Based, Visual-Inertial Motion Capture System" . . . 149

5 Conclusion and Outlook **171**
1 Summary . . . 171
2 Discussion and Future Work . . . 171

Bibliography **173**

Curriculum Vitae **199**

Preface

The present thesis is written as a cumulative doctoral thesis. Thus, the content is centered around the publications that constitute the main contribution of this work. Chapter 1 explains the motivation and relevance of the thesis and puts it into context. The introduction is followed by Chapter 2 which briefly summarizes the contributions made in this work. The publications themselves are located in Chapters 3 and 4. We conclude in Chapter 5, where we review the claimed contributions and provide an outlook for future work.

Chapter 1

Introduction

"One Algorithm – Many Robots and Tasks"

Robotics promises to help solving many of our future society's challenges. Mobile robots are said to take over dangerous tasks such as mining [125] or physically intense work as e.g. common in the construction sector [24, 86] with an example concept illustrated in Figure 1.1. Also, robots will enable entirely new processes and applications such as mass customization of products or novel medical treatments. Furthermore, robots could assist in rescue missions on disaster sites [160] or carry out inspection tasks in industrial settings [25]. Also, robotics can be an effective tool to use precious resources more efficiently, such as in precision agriculture [26, 148] or construction [3]. Robotics is also a part of the solution to social problems like our aging societies. Already as of today, there is a lack of skilled labor in many areas of our economy. And with decreasing birth rates leading to a demographic change towards a "society of elders", our economy has to sustain a larger overall population with a smaller working population. While robotics is often said to possibly take away jobs, it also has the potential to fill those gaps and allow people to work more efficiently and sustainably [18].

All the different applications for robotics raise the question: How can we make our robots execute a given task? This poses several technological challenges. First of all, we need to know where the robot is within the task environment. Equally important, we require knowledge about its internal state, such as in what configuration the robot is, what forces are currently acting on it or how it is interacting with the environment momentarily. These problems are summarized as localization and state estimation. Next, a motion planner needs to figure out how the robot needs to move in order to solve a task. This includes moving to a certain location in the state space while avoiding collisions, i.e. undesired interactions with the environment. In contrast for locomotion the robot is required to make desired interactions with the environment. Additionally, a task might require the robot to manipulate its

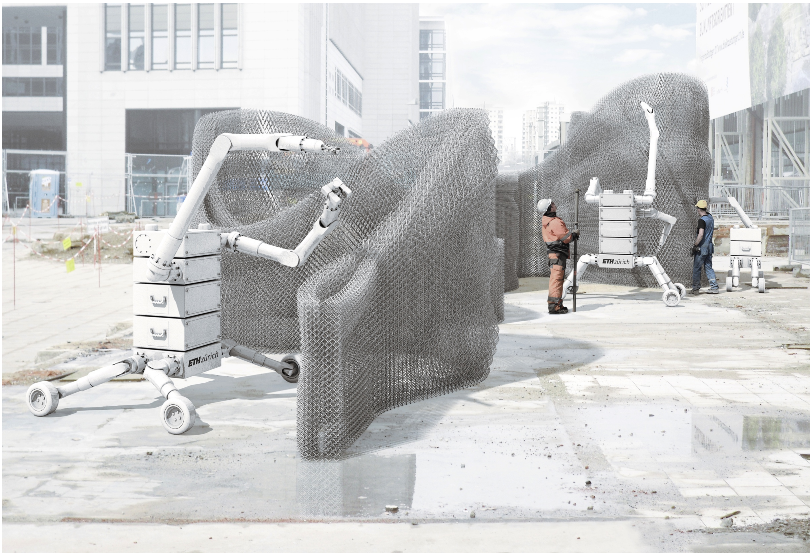
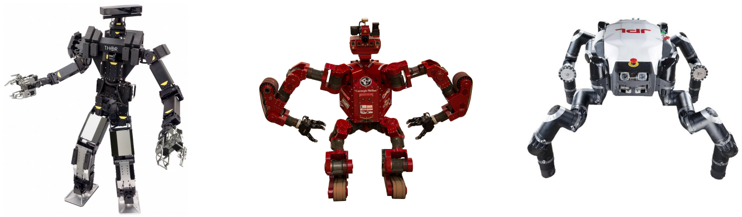


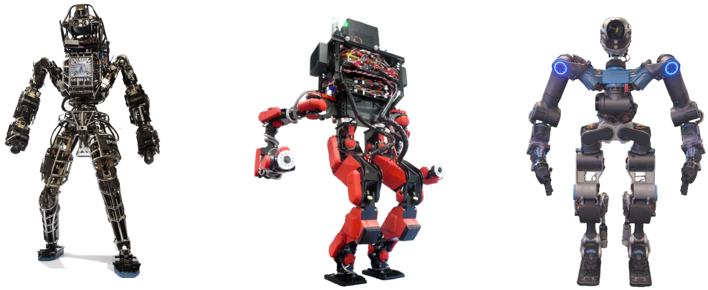
Figure 1.1: Concept drawing of the Insitu Fabricator 2 robot performing a construction task [86].

workspace, i.e. actively alter the state of the environment. Thus, we need some form of mission and motion planning that is also able to reason about contacts. Once we have figured out which motions are required for tackling the task, we have to design an execution layer that makes the robot follow the planned motion. However, usually the derived plans cannot be executed perfectly and we are faced with imperfect state estimates as well as inaccurate representations ("models"), both of our robot and its environment. Furthermore, the execution might be subject to disturbances as well as changes in the environment. To compensate for any mismatches between the planned motion and the executed motion, we can design regulators or motion controllers that try to minimize this deviation and compensate for errors during execution. However, changes or disturbances can be so large, that a plan becomes infeasible or invalid requiring us to plan a new motion. Ideally, this newly planned motion should be available immediately and respect the current state of the robot and its environment. This continuously ongoing process is often referred to as "sense, think, act", the *Robotic Paradigm*[7]. These three elements, have to be understood as building blocks of every robotic system, i.e. each robot has to measure or estimate its state and the state of the environment ("sense"), use this and any other prior information to plan adequate actions ("think") and execute them (“act”). However, this elements do not necessarily form a sequence but can be processes running in parallel. Also, the three building blocks cannot be tackled entirely separately. We will take a closer look at the “think” and “act” parts as well as their interface. More specifically, we are interested in finding an approach that can plan and execute motions for a complex task requiring coordination between all degrees of freedom of a robot subject to contacts. The approach should be independent of the robot category, its morphology, the hardware design as well as the task itself.

1.1 Motivation and Objectives



(a) THOR – THOR-OS [242] **(b)** CMU – CHIMP [234] **(c)** NASA JPL – RoboSimian [97]



(d) Boston Dynamics – Atlas [27] **(e)** SCHAFT – S-One [94] **(f)** IIT – WALK-MAN [225]

Figure 1.2: Pictures of the robots developed for the DARPA Robotics Challenge.

When trying to solve a particular robotic task, there are benefits of designing a specialized machine. It allows to restrict design decisions to fewer requirements and create a tailored solution to the problem. Due to the numerous tasks we would like to assign to robots, this leads to great diversity of system designs. But as the

DARPA Robotics Challenge [95] has shown, even for a very specific set of tasks, it is possible to make use of vastly different morphologies but also actuation and sensing principles as shown in Figure 1.2. Already as of today, new robotic systems are presented at a high rate and development cycles are likely to become faster. Due to the different ways of approaching the same task, it is difficult to deduct static parametric requirements for the development phase, such as the number of degrees of freedom, the link lengths, the actuator bandwidth or the mass of the robot. Instead it would be beneficial to constantly verify a robot design from a control and motion planning perspective to evaluate its task performance. This would allow us to provide feedback for the system design process as well as sensing and actuation choices. This co-design between hardware and control provides a unique opportunity to create an overall optimal robot. However, in order to keep up with the hardware development cycles, we need motion planners and controllers that are either general or adapt quickly to the task as well as the type or morphology of the robot. In order to design solutions that do not require to be re-engineered almost from scratch, we have to find approaches that are general yet powerful enough to deal with the complexity of today's and future systems. At the same time, we need to have sufficient control over the approach to adjust it easily for a given task and robot. This challenge can be summarized under the leitmotif "One Algorithm – Many Robots and Tasks" which also sets the frame for this thesis.

In order to develop a general yet also practical motion planning and control approach, we define the following criteria for our framework:

Req. 1 It handles high dimensional, nonlinear robotic systems.

Req. 2 It generates solutions for complex tasks that require interaction with the environment.

Req. 3 It transfers between systems of different classes and morphologies without significant adjustments.

Req. 4 It takes into account the entire systems including dynamics and kinematics.

Req. 5 It ensures feasibility and takes execution into account.

Req. 6 It runs online and can react quickly to disturbances or changes in the environment.

Req. 7 It finds locally optimal solutions.

Req. 8 It provides means of specifying the task in a simple manner.

The dimensionality of a robot is usually directly linked with the complexity or diversity of the tasks it is supposed to solve. Legged robots are among the most versatile robots which, in principle, allows them to solve complex manipulation tasks or traverse difficult terrain. Due to their potential versatility, legged robots have become very popular. However, their complexity creates complex planning problems in high dimensional spaces that require significant coordination of their motion. Imagine a humanoid robot that is supposed to pick an object from a shelf is faced with multiple challenges at a time. It needs to approach the shelf and reach out to the box while always maintaining balance and avoiding collisions with its limbs, both legs and arms. Such tasks are a great example where state-of-the-art approaches fail. To reduce the complexity of tasks, many approaches break a task down into smaller, easier to solve subproblems. However, this approach comes with huge drawbacks: The resulting individual motions are no longer fully coordinated with each other. Furthermore, reducing the problem usually excludes favorable solutions and leads to overall lower performance. Therefore, up to this day, we cannot leverage the full potential of legged or other complex robots despite the availability of very mature hardware. But also relatively simple systems such as Unmanned Aerial Vehicles (UAVs) pose challenging planning problems when given a more complex task such as moving through confined spaces or performing dynamic motions such as avoiding an obstacle or flying through a window. Again, introducing heuristics to these problems directly affects optimality and probably also limits the generality of such approaches. Therefore, we set the requirement that our motion planner and controller should be able to handle complex tasks for high dimensional robot planning problems. We also expect that it considers the dynamics and kinematics of these robots, to ensure feasibility but also to leverage the full potential of the robot. For the same reason we expect that a motion planner and controller is in some way optimal. Feasibility is a strict requirement, whereas optimality is desired within the boundaries of feasibility. When designing a complex motion planning approach it is essential to evaluate it with hardware experiments to verify the assumptions made. Therefore, the chosen method should be able to provide motion plans *online*, i.e. while running the robot. Usually, it is difficult to predetermine the possible states of the robot and the environment before running the experiments. Also, we cannot assume that we have a perfect prediction of our system and its environment. There will always be mismatches in our models as well as disturbances and other external effects, we cannot influence. Therefore, it is important that the motion planner is able react to such changes quickly. Last but not least, we need a simple way to specify a task. Creating a general motion planning and control algorithm is not of much use if the task itself needs to be encoded in a complicated, non-portable way. Ideally, we want to specify “what” the robot is supposed to do without specifying “how” to do it.

Such a highly flexible, capable motion planning and control framework is essential for realizing the promises that robotics holds. Recent advances in hardware design and sensing technologies have created a significant gap between the potential that our robots have and the degree to which we can make use of it. This has made motion planning and control one of the major limiting factors for many robotic applications, creating an urgent need to address this issue more fundamentally.

1.2 Approach

In the first part of the thesis, we aim at implementing our vision of “One Algorithm – Many Robots and Tasks” for motion planning and control. This constitutes the main focus of the present thesis. In this section, we describe how we approach the problem using the general framework of Numerical Optimal Control using a very efficient iterative solver that can handle nonlinear systems. We apply this to full Rigid Body Dynamic systems, such that the solver respects feasibility and leverages the full potential of the system. Our distinct goal is it to verify our methods on robotic hardware. Hardware experiments are essential as they reveal whether our assumptions hold and how robust our approaches are. Furthermore, they create additional requirements and restrictions such as the availability of good state estimation, the limit in computational resources and time or the constraints the actuation system imposes. The second part of this thesis tries to tackle the issues that we encountered during our pursuit of applying Numerical Optimal Control to physical robots. In this part, we tackle three subproblems relevant for online Numerical Optimal Control. First, we tackle our solver’s computational burden of calculating linear approximations of our system dynamics by leveraging Auto-Differentiation. Second, we demonstrate how to include simplified models of our actuators and sensors in physical simulators to allow for evaluating controllers before applying them to hardware. And lastly, we solve the issue of localizing mobile robots in their workspace by developing a fiducial based visual-inertial SLAM system.

1.2.1 Part A: Efficient Numerical Optimal Control

In this part, we give a brief introduction which formulation we use for defining our motion planning and control problem. Afterwards, we briefly summarize our approach to solving the problem.

Formulating Motion Planning as Optimal Control

Designing a general motion planning and control framework is a difficult task. Over the years, different communities have approached this problem from different angles [130]. Classical *Motion Planning* approaches deal with discrete states and transitions between them. These methods are usually based on the “Principle of Optimality” [13]. This principle is the foundation of the famous Dynamic Programming (DP) algorithm [15] which is used in many graph search algorithms such as Dijkstra’s Algorithm or A* [130]. However, when applied to higher dimensional problems, such as planning with differential constraints or large state space systems, the famous “curse of dimensionality” [15] occurs which prevents us from finding a globally optimal strategy in reasonable time even when pre-computing solutions offline. This is also, why discretizing a continuous time problem and subsequently solving it with DP usually fails. With a fine discretization, the dimensionality becomes an issue while a coarse discretization impairs the quality of the solution. As a result, we are left with two options: either we reduce the dimensionality of the problem or sacrifice (global) optimality. Especially with the introduction of computers, it has become easy to resort to "rule based" or "logic-based" decision making to reduce the search space. These rule-based approaches where the first examples of *Artificial Intelligence* [130] which in its early years mainly focused on discrete state problems. Yet, for more complex problems such heuristics are complex to define and often exclude desirable solutions. To solve complex planning problems the *Motion Planning* community has created sampling based approaches such as Rapidly-exploring Random Trees (RRTs) [131]. Like most other classical motion planning algorithms, sampling based methods are designed to find a feasible motion for problems with non-convex state constraints, such as navigating through a cluttered state space with non-convex obstacles. Therefore, quality or optimality of the solution is of second priority. Additionally, complex differential constraints of nonlinear physical systems can be difficult to include in such approaches. One example for this are RRTs, where we have to define a distance metric in state space which, for non-euclidean spaces, is not straightforward to do. To tackle systems with differential constraints, *Kinodynamic Planning* [55] has been developed. *Kinodynamic Planning* is concerned with finding state trajectories under velocity and acceleration constraints. In its original form, the system considered is a simple double integrator which is a coarse approximation for a Rigid Body Dynamics system. Later, the term *Kinodynamic Planning* was also used for approaches that considered different dynamical systems [130] which gave it a much broader but much less specific definition. Alternatively, *Trajectory Planning*, especially in the context of robotic arms, is concerned with finding a path in state or phase space for systems under differential constraints. It is then the responsibility of an execution layer to find the control actions that make a system follow the trajectory. By employing second order path constraints, also Rigid Body Dynamics systems can be covered. However, it is not guaranteed that a solution to the inverse control problem exists. *Trajectory Planning* often also contains a time refinement for converting a phase trajectory into a timed trajectory, allowing for time-optimal control [22].

In order to avoid the inverse control problem, we focus on planning in control input space rather than state space which suggests approaching the problem from a *Control Theory* point of view. Together with our requirement of finding locally optimal solution, we naturally find ourselves in the field of *Optimal Control*, which also gave rise to the aforementioned Dynamic Programming algorithm. Optimal Control tries to answer the question: *Given a state of the robot and its environment, their associated dynamics and a task objective, which is the best control action to take next?* This is appealing because Optimal Control in principle not only reasons about the system's capabilities (dynamics) but also its limitations and the task requirements (constraints) as well as the quality of the plan (reward/cost function). Additionally, the broader framework of Control Theory provides us with the tools of reasoning about the stability or risk associated with the execution of a planned motion. Furthermore, it provides us with means to design feedback controllers that react to deviations while following a plan.

Another reason for choosing an Optimal Control perspective is that we are interested in motion planning and control for short time horizons of up to around 10 seconds. This is not an absolute number but rather a rough scale. However, we want to distinguish our approach from fields such as mission planning and navigation, which broadly speaking look at longer time horizons and are concerned with subjects like task scheduling, semantic mapping and obstacle avoidance but less concerned with topics like control, tracking or estimation. While also for short horizons mapping and obstacle avoidance can be important, the requirements towards these topics are different, as we have seen in the discussion about classical motion planning. Furthermore, we want to distinguish our approach from purely reactive Optimal Control that only reasons about the current and maybe the next step instead of the more long term evolution of the state over time. Yet, a clear distinction between reactive, short and long term planning is not always possible and often a combination of both can be applied. However, generally speaking, the focus of this work is short time horizon planning.

As mentioned before, approaching Optimal Control by using Dynamic Programming, i.e. tabulating in space, is infeasible for large, complex systems. Therefore, we avoid discretizing our state space and instead try to solve the problem in continuous space using numerical methods. As we will see over the course of this section as well as in Section 1.4, some of these approaches directly originate from the field of Control Theory and are specifically designed for solving Optimal Control problems. Other approaches rely on tools that stem from fields such as planning, physics, machine learning or mathematical optimization, which only after their original development have been applied to Optimal Control problems. While we take a control perspective in this work, we will also provide an outlook to such approaches.

Efficient Numerical Optimal Control applied to Rigid Body Dynamics

The shortcomings of classical motion planning methods described above is our motivation to study Numerical Optimal Control. The increase of computational power in recent years as well as advances on the algorithmic side make Numerical Optimal Control a viable option for solving complex Optimal Control problems in reasonable time. Thus it has gained significant popularity lately. Yet, most applications of Numerical Optimal Control in robotics that have been presented in the past, only showcase simulation results or resort to simplified models, raising the question whether these methods can keep up the promises the community associates with it. Furthermore, the lack of hardware results does not provide insight of what is required to successfully apply Numerical Optimal Control to robotic hardware. While processing power has brought these methods into reach for online operation, many implementations still suffer from long computation times that result in planning times that – by far – exceed the time horizon of the plan.

Therefore, we focus on highly efficient Numerical Optimal Control. The most efficient approaches are Differential Dynamic Programming (DDP) [147] approaches that iteratively solve a local Dynamic Programming problem around an initial guess, leveraging gradient information. As presented in Section 1.4, first-order variants of these methods have shown great results on different systems. However, computational complexity, insufficient constraint handling and the lack of an openly available, fast solver have hindered convincing hardware results. Therefore, we aim at eliminating these shortcomings and investigate to what extend these methods scale to hardware experiments. Hopefully, this will provide an insight if Numerical Optimal Control can serve as a general framework for motion planning and control for complex tasks on a multitude of robots.

Furthermore, we focus on optimizing over full Rigid Body Dynamics models. Dynamic models are essential for leveraging the full potential of the system. Especially in mobile robotics, reasoning on a kinematic level can be insufficient. On legged systems, kinematic approaches do not allow to reason about dynamic stability which is essential for many gaits. Additionally, kinematic approaches cannot reason about the limitations of a force based actuation system and thus limit feasibility. Furthermore, we need to reason on a force / acceleration level when interacting with the environment, as required by tasks that involve locomotion, manipulation or aerodynamics. The Rigid Body Dynamics formulation provides us with a general dynamic modelling framework that allows us to obtain a reasonably accurate model for most mobile robots. Rigid Body Dynamics are well established in robotics and frequently used for control and simulation.

Solving a Numerical Optimal Control problem for full Rigid Body Dynamics system online remains a challenge. However, for the reasons introduced before, we explicitly want to stay away from applying heuristics to simplify the problem. Instead, we aim at exploiting and extending the capabilities of modern Optimal Control solvers and algorithms for Rigid Body Dynamics computation.

1.2.2 Part B: Applying Numerical Optimal Control to Mobile Robots

Applying Numerical Optimal Control to hardware is a major challenge, since it poses requirements on the computation time, the quality of the state estimate or the actuation and control performance. If addressed poorly, these issues can prevent successful hardware results. Therefore, in this second part, we focus on the more practical considerations of using Numerical Optimal Control on physical systems. We take a look on how to increase the efficiency of our solver, how to verify the resulting controllers prior to hardware experiments and how to obtain accurate state estimates.

Efficient Derivative Computation for Rigid Body Dynamics

Since we are applying gradient-based Numerical Optimal Control, we have to find efficient means of computing these gradients. The common choices for gradient computations are analytical derivatives, Numerical Differentiation and Auto Differentiation. Analytical derivatives are accurate but difficult to derive manually. Symbolic computation engines provide means of automating the derivation but the complexity of the resulting expressions is usually very large. Numerical Differentiation is simple to setup but generally slow. Hence, we will focus on Automatic

Differentiation to obtain fast derivatives in an automated, error-free way. Automatic Differentiation is a well established tool in the optimization and Numerical Optimal Control communities. However, it has rarely been applied to complex Rigid Body Dynamics systems, since efficient modelling frameworks for Rigid Body Dynamics do not support Auto Differentiation. Therefore, we take a look at a very efficient tool for Rigid Body Dynamics computation and modify it to support Auto Differentiation. We then combine it with our Numerical Optimal Control solver to significantly decrease computation time. During these studies, we also take a look at a major speed limitation of Auto Differentiation. Auto-Differentiation needs to built and traverse an expression graph both for the original function and its derivative at runtime. This creates significant overhead and limits performance. Hence, we further extend our approach by applying a code generation library that creates static code from the expression graph which can be pre-compiled for efficiency.

Verifying Controllers in Simulation

Simulators are a common tool to verify planners and controllers. Additionally, they can act as a "sanity check" before applying trajectories to hardware. These simulators usually assume a perfect model with no disturbances or modelling errors. At the same time, simulators often provide perfect state information which is not available on the physical system. However, physical robots do not behave like their perfect, deterministic simulation counterparts. Instead they are subject to stochastic processes such as noise, biases and disturbances. Furthermore, the physical system is influenced by offsets, delays, actuator dynamics and other imperfections. So far, many simulators simply ignore most of these effects, providing only limited sensor, input-output and actuator models as well as limited perturbation means. Still, simulation results are frequently used to verify controllers which raises the question how significant the difference between the model and physical system is for such purposes. Therefore, we will analyze Rigid Body Dynamics simulators from a control perspective and investigate to what degree they can reason about control stability. This will provide insight to what degree we can rely on simulation results and if hardware results are really required.

State Estimation for Mobile Robotics

An accurate estimate of the internal state of the robot as well as its environment is essential for the planning and during execution of motions. Especially when planning and executing a task with respect to the workspace such as reaching a certain location or manipulating an object, we require an accurate estimation of the pose in the workspace. Most robots, including the platforms we are considering in this work, come with fairly good state estimators for their internal state, i.e. their joint states and loads. With the help of an Inertial Measurement Unit (IMU) usually the orientation of the links or bodies can be estimated as well. However, even when equipped with perception sensors, all of these modalities do not provide a localization in the workspace. To overcome this issue, we can either make use of complex (semantic) mapping approaches, that are difficult to set up or use motion capture systems, which are highly expensive and tedious to calibrate. Therefore, we aim at developing a localization system that allows localizing mobile robots in their workspace. The system should be easy to set up and inexpensive in both computation and hardware costs. Therefore, we propose a visual-inertial, Extended Kalman Filter (EKF) based Simultaneous Localization and Mapping (SLAM) system that relies on artificial landmarks. The system only relies on a single camera and an IMU, i.e. sensors that are available on most mobile robots including Unmanned Aerial Vehicles (UAVs), service robots or legged systems.

1.3 Hardware Platforms

In order to validate if our approaches transfer between different systems, we put different robotic platforms in place for experiments. We focus on four systems in total: two quadrupeds (four legged robots), namely the hydraulically actuated "HyQ" and the electrically actuated "ANYmal", the hexrotor "Firefly" and the ball balancing robot namely "Rezero". This set of robots constitutes an interesting and challenging benchmark for motion planning and control, since it includes systems of different complexity and categories including walking, ground and flying robots. Furthermore, all three systems show nonlinear dynamics with coupling between the system states which poses a challenge to conventional control approaches. Furthermore all systems are underactuated, i.e. we cannot control all states directly. The nonlinear, coupled dynamics paired with underactuation also create difficult motion planning states since we cannot decouple the system or plan for states individually. Instead, dynamically consistent, coordinated motion planning is required.

1.3.1 HyQ – The Hydraulic Quadruped

HyQ [205], as shown in Figure 1.3, is a hydraulically actuated, four legged robot (quadruped) developed by Claudio Semini and his group at IIT Genova. The overall robot weighs about 80kg and is powered by an offboard hydraulic pump. Each leg of HyQ has three degrees of freedom including Hip Adduction Abduction (HAA), Hip Flexion Extension (HFE) and Knee Flexion Extension (KFE). Each degree of freedom is force controlled and equipped with a force sensor as well as absolute and relative encoders for joint angle measurements. All sensors are connected to custom made electronics connected via an EtherCAT bus to a main, desktop grade computer. The main computer runs a real-time Linux and is responsible for computing the hydraulic force controller and an optional joint position controller centrally.

As part of this thesis, we have equipped HyQ with a sensor fusion algorithm [19] that fuses data from the joint encoders and an Inertial Measurement Unit to estimate the base pose and velocity. Together with the leg encoder data, this provides a full estimate of the entire state. Furthermore, we have built a custom,

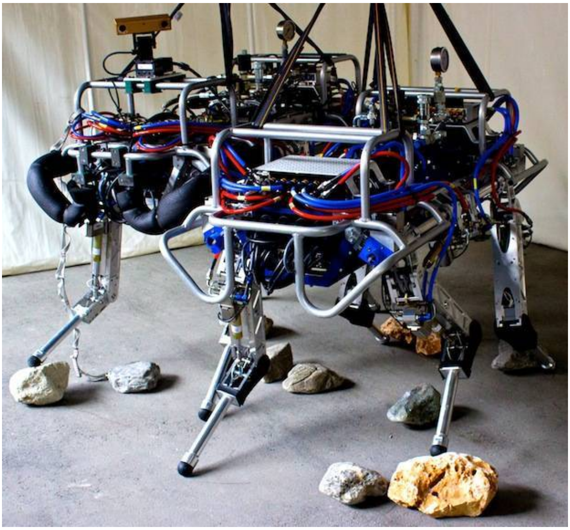


Figure 1.3: Picture of the original HyQ (left) developed at IIT Genoa as well as HyQ Blue (right), a copy of the robot currently operated the Agile & Dexterous Robotics Lab at ETH Zurich.

low latency TCP/IP bridge that allows for computing setpoints for the force and joint position controller from an external computer. These improvements have not only enabled the work in this thesis [166, 168] but also the ones presented in [70, 71, 181, 237, 238]

Using a standard Rigid Body Dynamics model, the robot has in total twelve control inputs (three per leg) and 18 degrees of freedom (six for the base, three per leg).

1.3.2 ANYmal

ANYmal [107] shown in Figure 1.4 is an electrically actuated quadruped developed by Marco Hutter et al. at ETH Zurich. ANYmal uses Series Elastic Actuators (SEAs) which consist of electric motors, a backlash-free gearbox and an elastic element (spring). The elastic element protects the gearbox from impacts and can store mechanical energy. All actuator components including the motor electronics are packed in an integrated actuator called ANYdrive. Neglecting internal degrees of freedoms in the actuator, ANYmal has the same basic degrees of freedom as HyQ, namely HAA, HFE and KFE. However, the ANYmal's actuator are endstop free

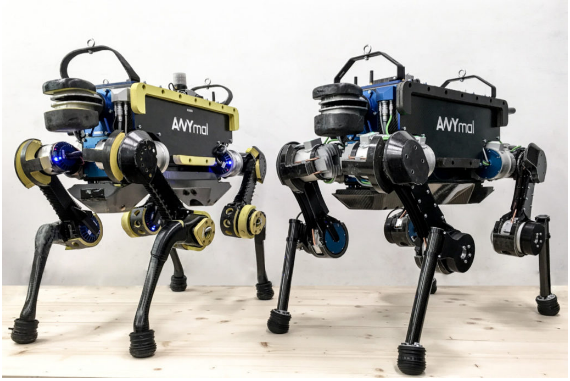


Figure 1.4: Picture of the electrically actuated quadruped ANYmal.

and thus do not have hard kinematic limits. All actuators are fully torque controlled using the on-board controller. A centralized computer connected to the drives via EtherCAT is used for whole-body motion planning and control. Furthermore, ANYmal is equipped with an IMU as well as force sensors at the feet. The robot runs the same sensor fusion algorithm [19] as HyQ. Additionally, it is equipped with a perception system using laser range finders and cameras. However, the perception system is not used in this work.

1.3.3 Rezero

Rezero [76], shown in Figure 1.5, is a ball balancing robot, a so called “ballbot”. Rezero has been built by a group of eight undergraduate students (including the author of this thesis) in 2009/2010. Since the robot only has a single point of contact with the ground, it is not statically stable but needs to balance at all times. It does so by turning the ball using three omnidirectional wheels. This allows Rezero to be omnidirectional itself and change its orientation (yaw angle) simultaneously. Each wheel is actuated by an electrical brushless motor which is force controlled by the motor electronics and connected to a central micro controller via a CAN bus. The balancing feedback controller is computed on the micro controller which receives setpoints from a desktop grade computer mounted on the robot. The state of the robot is measured by an IMU on the main body, estimating tilt angles and orientation as well as encoders on the wheels, estimating the rotation angles and velocities of the ball.

During this work, we have upgrade the robot’s micro-controller to an ARM based architecture with an onboard Floating Point Unit. The resulting increase in computational power allows us to test more complex motion planning and control

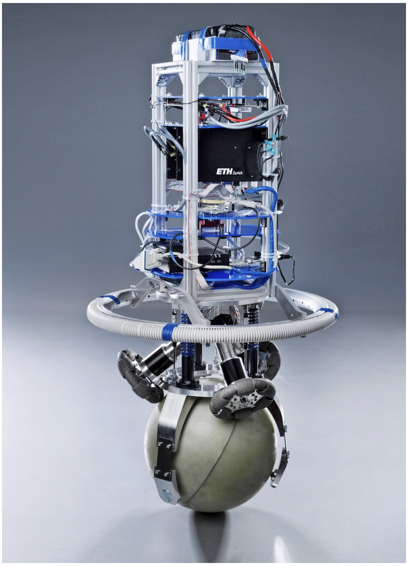


Figure 1.5: Picture of the ball-balancing robot (ballbot) Rezero.

approaches on the system including [68, 165, 182].

We assume no slip between the wheels and the ball as well as the ball and the ground. Hence, Rezero has in total five degrees of freedom (three for the orientation of the main body and two for the ball angles) and three control inputs (one per wheel).

1.3.4 AscTec Firefly

The Firefly [8], as shown in Figure 1.6, is a UAV with six rotors built by Ascending Technologies (AscTec). It is a research grade multirotor equipped with six speed controlled motors. Furthermore, it is equipped with an IMU to estimate the pose and as well as the velocities of the main body. The UAV features a desktop grade onboard computer which computes stabilizing controllers but can also be used for motion planning and high level controllers.

As most UAVs, the Firefly is usually modelled as a single floating rigid body. Each rotor can be controlled individually, leading to six control inputs. However, all rotors can only create lift along the vertical principle axis of the UAV. Thus, the resulting overall force is also parallel to this axis and no forces can be created in the horizontal plane of the main body. Hence, the control input dimensionality

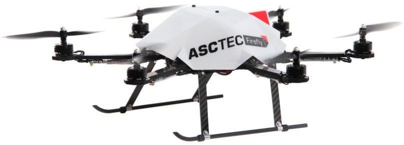


Figure 1.6: Picture of the Ascending Technologies Firefly [8], a Unmanned Aerial Vehicle with six rotors used for testing in this work.

can be reduced to four (thrust and three moments around the principle axes) which can then be mapped back to the six rotors.

1.4 Related Work

The publications which constitute the basis of this paper-based thesis each contain a brief yet specific literature review. In this section, we would like to present a slightly "bigger picture" of related work. Furthermore, we provide a categorization which hopefully allows the reader to understand the relation between the state of the art and the approach chosen in this work as discussed in subsection 1.2.

1.4.1 Part A: Efficient Numerical Optimal Control

There are several well-written literature reviews on Numerical Optimal Control [50, 53, 54] and Trajectory Optimization [16, 197], which provide great insight into these methods from a mathematical Control Theory and point of view. Furthermore, they present the different categories of Numerical Optimal Control approaches. In this subsection, we will briefly summarize the different categories and approaches. Additionally, we will review the different approaches Numerical Optimal Control can be applied to Rigid Body Dynamics under contacts. Lastly, we will review related work that applies Numerical Optimal Control to mobile robots.

Before we start our review, it is important to clarify the nomenclature. Numerical Optimal Control and Trajectory Optimization are frequently used interchangeably. Some authors [16, 197] try to provide a definition for distinguishing the two by associating Optimal Control with the problem of finding a function representation (with "infinite variables" [16]) for the control trajectory while Trajectory Optimization tries to find a parametrized representation. Other categorizations distinguish whether the approach finds a globally optimal solution based on sufficient conditions for optimality or "just" a locally optimal solution based on necessary conditions for optimality [220]. For the scope of the present work, a rigorous definition of the term "Trajectory Optimization" is not strictly required. As we will see in the following paragraphs, the term is not necessary for a clear classification of approaches but probably adds more ambiguity than it resolves, which potentially makes it even obsolete in the first place. Thus, some authors [53, 54] do not use this term altogether. Throughout this work, we are using “Trajectory Optimization” interchangeably with “Numerical Optimal Control”.

Numerical Optimal Control Formulations

There are two categories of approaches for solving a Numerical Optimal Control problem [16, 53, 54, 197]. The first category are indirect methods that are based on the Calculus of Variations leading to a boundary value problem. Alternatively, direct methods parametrize the infinite-dimensional control policy to a finite dimensional policy which is an approximation of the actual solution. This leads to an numerical optimal control problem which constitutes a special form of a nonlinear optimization problem, also called a Nonlinear Program (NLP). For reasons elaborated in [197], indirect methods are often difficult to set up and thus restricting the possibility of creating general solvers. Since this speaks against our leitmotif “One Algorithm – Many Robots and Tasks”, this thesis focuses on direct methods.

The most prominent direct approaches for solving Numerical Optimal Control problems are transcription methods that convert (“transcribe”) the optimal control problem from its specialized form into a standard NLP. The benefit of using a standard NLP formulation is that these methods can leverage off-the-shelf, mature NLP solvers such as [88, 110, 230]. There are two main classes of approaches here, namely sequential and simultaneous approaches. Sequential approaches only use the control input as a decision variable and thus the decision vector is a minimal representation. Simultaneous approaches optimize over both state and input variables constrained together via the system dynamics. While this seems like an over-parametrization at first, the resulting problem is well structured [50, 54] which can be leveraged by the NLP solver. A very well-known sequential transcription method is Direct (Single) Shooting. Direct Shooting integrates the system dynamics to obtain a trajectory (the “shot”) and subsequently optimizes the control inputs via the NLP to optimize the cost subject to the constraints. While single shooting is simple, it is very sensitive to the initial guess. Less sensitive to an initial guess is Direct Multiple Shooting (DMS) which partitions the problem in several intervals in time. For each interval, a separate shot is computed. To avoid discontinuous solutions, continuity constraints between the shots need to be introduced. The benefit is that each shot can be initialized by an artificial start state while over the course of iterations the solver resolves continuity constraint violations. The decision variables are the control inputs and the initial state each shot. Therefore, DMS can be seen as a simultaneous approach with a strong connection to a sequential shooting approach. The most prominent simultaneous approach is Direct Collocation (DirCol). In DirCol state and input trajectories are approximated with (usually polynomial) basis functions divided into collocation intervals. The decision variables in the NLP are the parameters of the basis. Since all direct transcription methods create an NLP, the solver can be chosen independently. Most solvers are based on

Sequential Quadratic Programming (SQP) or Interior Point (IP) methods, which represent different approaches of handling inequality constraints. The choice of solver is highly dependent on the system, the transcription method and parameters as well as the problem formulation [182]. For a more detailed review of both, direct transcription methods and suitable solvers we refer to [16, 50, 53, 54, 197].

Another way of solving Numerical Optimal Control problems is Dynamic Programming (DP) [15]. Solving the general, globally optimal, Dynamic Programming problem is usually intractable for larger problems due to the "curse of dimensionality"[15]. To overcome this problem, Differential Dynamic Programming [147] has been proposed, which iteratively solves a local Dynamic Programming problem. DDP first computes a forward simulation of the system dynamics. The nonlinear system dynamics are then quadratically approximated around the obtained trajectory. In a backward pass, which is derived from the principle of optimality and that uses Riccati-like equations, an update to the control law is then computed. DDP is a second order method which relies on the explicit computation of the Hessian matrix for the backward pass. In later work, first order methods based on a Gauss-Newton Hessian approximation have been proposed under the names iterative Linear Quadratic Control (iLQG) [222] and Sequential Linear Quadratic Control (SLQ) [210]. The backward pass of DDP methods is used to improve the control input. As a byproduct, it also provides feedback gains that correspond to the solution of the finite-horizon, time-varying Linear Quadratic Regulator (TVLQR) problem around the previously computed state trajectory. In iLQG [222] the explicit usage of these feedback gains during the forward integration is described, whereas the original DDP [147] and SLQ [210] algorithms only rely on the feedforward control input. Another difference is that iLQG proposes a quadratic approximation in case of non-quadratic cost functions which has only been introduced to SLQ later [201]. DDP-style algorithms can be implemented very efficiently making them applicable for Model Predictive Control [61]. This is possible since, in contrast to NLP solvers, DDP approaches are specifically designed for numerical optimal control problems and thus directly leverage the structure of the problem. This can lead to linear complexity in time horizon while NLP solvers generally have polynomial complexity, unless they explicitly exploit sparsity. It is worth noting DDP-style algorithms, in its original form, solve an unconstrained Optimal Control problem and are tied to a piece-wise constant parametrization for the control input trajectory. There have been efforts of including constraints in DDP methods [180]. For first-order variants, [222] proposes a simple clamping of the control inputs to satisfy input constraints. As [218] later shows, this is suboptimal and they propose a Quadratic Programming (QP) approach for the backward pass which has been previously applied to DDP as well [136]. The issue here is that solving QPs is expensive and breaks linear time complexity. The same applies to variants that handle pure state and state-input constraints [201]. By the time of writing this thesis, early promising results in constraint methods that maintain linear time complexity have been presented by [71, 83].

While derived very differently, DDP and Direct Transcription based approaches have common properties and make use of similar mathematical tools. When not considering the use of the feedback law in the forward simulation of DDP-methods, these methods are closely related to direct single shooting, with only the backward pass often being derived and implemented differently. Quoting [201]: “Our algorithm can be categorized as a single shooting SQP method but with the key characteristic that the QP subproblem is solved via a constrained LQ-OCP (Linear Quadratic Optimal Control Problem, *editor’s note*) [...]”. But also the backward pass of direct transcription approaches and DDP-methods are related. DDP-methods rely on Riccati equations derived from Optimal Control whereas LQ-OCP solvers in direct transcription methods can make use of Riccati factorization schemes [54]. This allows to break the otherwise cubic complexity in time horizon of NLP solvers down to the same linear complexity as DDP-methods. The full relation between both approaches has been shown in recent work [84] which explains how DDP methods can be converted to Direct Multiple Shooting problems. This is done via so called “‘lifting”, where the state trajectory is added to the optimal control problem. This research finally connects two streams of research that have been in large extends been developed independently. The convergence of both streams yields immediate benefits to both lines of research. The “lifting” trick allows DDP-methods to use standard LQ-OCP solvers which can handle constraints, both in states and inputs. Furthermore, DDP-methods, generally being Single Shooting Methods, can now also be formulated as Direct Multiple Shooting problems, which greatly helps parallel computation as well as initialization on unstable systems. In return, the Optimal Control perspective on DMS allows for making use of feedback gains. These feedback gains have been a numerical byproduct of LQ-OCP solvers for a long time but often disregarded in practice. However, they now gained a meaningful interpretation as feedback gains.

In many state-of-the-art approaches, first order methods of the algorithms described above are used. In these methods the Hessian is often approximated using Gauss-Newton [84, 211], as done e.g. in SLQ, iLQR and frequently in Direct Single and Multiple Shooting. Some NLP solvers, e.g. [230], can also make use of Quasi-Newton approaches such as the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm. In contrast, second order methods such as the original DDP algorithm as well as second order NLP solvers directly rely on an explicitly computed Hessian. Second order methods often provide better gradient information and thus allow for larger step sizes. Therefore, while they are more expensive to compute, the overall run time until convergence can still be lower, due to fewer iterations being required. In applications where a good initial guess is available, e.g. from a previous iteration in a Model Predictive Control setting, first order methods usually regain their advantage.

Apart from DDP and direct transcription methods, sampling approaches such as Path Integrals [114, 215, 221] or black box optimization [96, 108] can be used to solve the Numerically Optimal Control problem. Some of these approaches [96, 221] entered the field via Reinforcement Learning. Sampling algorithms are often designed to not require gradient information, such that they can be run online or on an unidentified system. This makes it easier to set up the problem and makes theses systems less restrictive on the representation of the system dynamics, constraints and cost function. However, most of these methods are still implicitly gradient-based approaches and thus need to estimate the gradients from the samples. This process requires a significant amount of data per optimization parameter. Thus, these approaches scale less well for large dimensional policies and generating the samples, either in simulation or on hardware, is an expensive, time consuming process. However, recent advances in computational power bring these approaches closer to online usage [235]. In recent years, neural networks have been used as a sophisticated basis for the optimal policy. Their expressiveness allows for learning complex, more general policies which can prevent repeating the tedious, data and time intensive training when conditions or tasks are changed. However, the increase in required training data and time make these approaches very expensive to tune and, if data samples are not carefully selected, subject to over-fitting affecting their “generality”.

Numerical Optimal Control Toolboxes

Due to the increase in popularity of Numerical Optimal Control, there is an increasing number of toolboxes available in this domain. These toolboxes tie in at different levels. Some toolboxes provide a whole toolchain from modelling the system, formulating the optimal control problem and solving the latter. In the field of Rigid Body Dynamics, these toolkits include [219, 224]. One level below there are toolboxes that focus directly on the Optimal Control Problem [51, 105], by implementing direct transcription methods and providing a backend for solving the resulting optimization problems. Other toolboxes such as [6, 59] are not focused on Optimal Control but provide a general interface for setting up mathematical optimization problems, ensuring that the full potential of the underlying solvers are used. This is done by automatically computing derivatives and providing sparsity patterns to the solver. There are also dedicated Numerical Optimal Control solvers such as [88, 110, 230] where it is up to the user to transcribe the Optimal Control problem and provide derivatives and sparsity patterns. In the end, the user is left with a trade-off to make. Easy-to-use, high level toolboxes provide a great level of convenience and ease of setup. However, they require the user to formulate the problem in either a toolbox specific way, using specialized syntax or with the help of third party tools such as Matlab. However, this is an issue when dealing with Rigid Body Dynamics algorithms since implementing them efficiently is not straightforward [72]. There are efficient Rigid Body Dynamics libraries e.g. [73, 79] but they are implemented in C++ and thus not compatible with these high level Numerical Optimal Control Toolboxes. On the other hand, toolboxes with lower level access provide a greater level of flexibility and allow for interfacing the solver with custom implementations of the model. However, this comes at the cost of a more tedious, error-prone setup procedure since the user has to provide the linearizations and sparsity patterns.

Numerical Optimal Control for Rigid Body Dynamic Systems in Contact

Contact dynamics are physical effects that describe the interaction between two bodies which are in touch with each other. Contacts are in general nonlinear physical effects that have proven to be difficult to model. Depending on the chosen model, these effects can also be non-smooth or numerically stiff, posing challenges for Numerical Optimal Control. Contacts[1] can be modelled either as elastic or inelastic. In the elastic case, bodies are assumed to be deformed by contacts and the repelling contact force is a function of the penetration. Elastic contacts, as studied by Hertz [100], are an accepted method of modelling contacts in the field of mechanics [33, 72, 109]. These methods are often also referred to as "penalty" methods or "soft contacts". However, considering rather physically stiff materials as used in robotics, an elastic model paired with high physical stiffness leads to very numerically stiff differential equations. Alternatively, contacts can be approximated as entirely inelastic ("hard") contacts. As long as the contact state between the two bodies does not change, we obtain a very smooth, numerically well behaved system. However, at a contact switch the dynamics are not longer continuous. Since an inelastic contact does not allow for penetration, a discrete change of velocity, i.e. an impulse, will occur at impact. These effects are subject to research in the field of non-smooth mechanics [33, 90] trying to solve the challenges involved with discrete changes in state space. This has also led to the development of the time stepping method [158] which allows for handling infinite forces occurring at impulses.

Another physical effect that creates an interaction force between two bodies is friction. In Rigid Body Dynamics, we often consider dry friction which describes contact forces lateral to the surface normal or penetration direction[2]. Dry friction includes both static friction "stiction" and kinetic friction. As contacts, friction is a highly nonlinear effect. When modelling friction as Coulomb friction there are also discontinuities at the transition from static to kinetic friction. In the Coulomb friction model friction is furthermore a function of the normal force, creating a dependency between normal contact forces and friction, i.e. the "friction cone". Inelastic contact dynamics under friction are expensive to compute, since a Linear Complementary Problem (LCP) has to be solved [33, 72, 90]. Instead elastic contacts only require to compute relatively simple force laws and apply the resulting force to the Rigid Body Dynamics system. However, the stiffer elastic dynamics require smaller integration steps, limiting the speed advantage. Especially due to their increased robustness over penalty methods, constraint based contacts using LCP solvers have become the method of choice for most physics engines [44, 139, 212].

When using elastic contacts in a Trajectory Optimization problem, the contact model defining the relation between penetration and contact force, can simply be added to the system dynamics as e.g. demonstrated in [211, 222]. Hence, the system dynamics consists of the contact model and the rigid body dynamics. Given a state

---

[1] We are focusing on non-adhesive contacts here, ignoring their adhesive counterparts.

[2] Joint and/or actuator friction is also an important effect to consider but lies outside of the scope of this discussion.

of the system (which can optionally include the state of the environment), we can compute the penetration which is then used by the contact model to compute a contact force. Furthermore, friction can be added to the contact force. The contact force is then added as an external force to the rigid body dynamics. Therefore, the Numerical Optimal Control algorithm is not concerned with the contact dynamics and model itself but indirectly reasons about it via the dynamical system. Hence, the continuity or "smoothness" is important such that its gradients are well defined [211, 223]. Unfortunately, as described above, both friction and normal contact forces are not necessarily smooth with respect to the state of the Numerical Optimal Control problem, deteriorating gradient quality. Smoothing out the contact model on the other hand might decrease physical accuracy. Therefore, contact models by themselves are an active line of research (see e.g. [9]) and they are partially developed specifically for a certain Numerical Optimal Control approach [211, 223]. These contact models try to find an optimal trade-off between physical accuracy and numerical properties. The benefit of using an explicit contact model is that the Numerical Optimal Control is provided with smooth dynamics and can reason about contacts without having to handle complex constraints or discrete decision variables.

Inelastic contacts are also popular in Numerical Optimal Control, since the absence of a numerically stiff contact model allows for larger integration step sizes for solving the differential equations. Instead of a difficult to tune contact model, contacts are considered for as constraints. There are different possible formulations for these constraints. Probably one of the most general form are complimentary constraints where the product of contact surface distance (and optionally velocity) and contact force has to be equal to zero. This means that either the bodies can be at distance but have no contact force acting between them or they are at zero distance with a contact force acting upon them. In this case contact forces are added as decision variables to the problem. Numerical Optimal Control combined with inelastic contacts and complimentary constraints has been studied in the field of non-smooth mechanics [244, 245]. Later, such approaches have become popular in legged robotics [188]. These approaches are very appealing since the underlying NLP optimizes for the state and control trajectory as well as contact sequences, timings and locations. However, the contact constraints makes such approaches also very expensive to compute and not applicable to online applications. If a contact sequence, but not necessarily the timing or location, is assumed to be given, the constraint can be separated into a pure state constraint preventing ground penetration and a pure input constraint preventing contact forces on bodies which are not in contact at a given time [189]. Alternatively, the contact sequence can be used to project the dynamics into the constraint-consistent subspace [2, 152, 200] treating contact forces as internal forces rather than decision variables [181]. This effectively corresponds to a hybrid system or multi phase approach where different dynamics or constraints are used based on the contact configuration, which is also leveraged in [70, 188, 204]. A projection approach has less decision variables but requires pre-computing the projection. While the contact sequence needs to be given in these formulations, it is still possible to optimize for contact timings. This can be done either as part of the Numerical Optimal Control itself by adding the timings as decision variables [181, 204] or by using an additional optimization routine [70]. When using a shooting method without an explicit contact model, the contact configuration can be determined during the forward integration (the “rollout”) while still using a hybrid dynamics [47] or constraint-based approach. However, in this case, the Numerical Optimal Control approach does not necessarily reason about contact sequences explicitly but might just “tolerate” them [47].

There are advantages for both, elastic and inelastic contact modelling. In general, elastic contacts lead to smoothness of the underlying problem which helps convergence speed in gradient based approaches. However, when a contact sequence is specified, inelastic contacts have the benefit of less stiff system dynamics. When it comes to control and stabilization of a system, inelastic contacts are very popular as we will discuss later in this section.

Motion Planning and Control for Legged Robots

Prior to the increase in popularity of optimization approaches, motion planning in legged and especially quadruped locomotion has often been tackled with controllers and planners specialized in creating a particular gait. To reduce the complexity of the systems, early approaches resort to simplified models. In the early days of legged locomotion, Linear Inverted Pendulum Modes [112] and Spring Loaded Inverted Pendulums (SLIP) were applied to model walking. These models represent the robot as a single mass, concentrated at the Center of Mass (CoM), that is in contact with the ground over the pendulum “stick”. Despite their simplicity, these models can lead to impressive performance especially when combined with robustness criteria such as the Zero Moment Point (ZMP) [228] as done in [43, 113, 236, 248]. Due to their low-dimensionality they can be efficiently used in Numerical Optimal Control [144, 236] and Optimization [156] approaches as well. If a quasi or purely linear representation is chosen, this can even allow for Model Predictive Control schemes [99, 247]. However, in such approaches angular momentum is either neglected or constrained to be zero, limiting the solution space. If not only the linear motion the CoM is considered, but also its angular momentum, we arrive at Centroidal Dynamics [178] or (single) Floating Rigid Body (FRB) models. These models have become very popular in Numerical Optimal Control for motion planning in legged robotics, e.g. [46, 71, 102, 127, 128, 247]. While these models can account for all masses and inertias from the link by projecting them to the CoM, they cannot reason about internal forces which might lead to plans that exceed the system’s actuation limits. As mentioned in the previous subsection, full Rigid Body Dynamics models have also been used in Numerical Optimal Control for legged locomotion [143, 154, 181, 188, 217]. However, hardware results are rare and computation time is often far from online applications. When online computation is not required, gradient-free, blackbox optimization can be used as shown for a full Rigid Body Dynamcis model [195] or even more complex models that include the entire actuation system [35]. These approaches do not require gradient information but still show impressive results even on hardware [35], they require a large amount of data samples which is computationally expensive to generate.

One challenge in motion planning for legged robots is finding contact sequences. One option is to predetermine them [43, 113, 144, 248] which works especially well in humanoid robots, where the number of gait pattern types is very limited. However, when considering multiple endeffectors such as quadrupeds or humanoid robots with arm support, pre-specifying gait pattern becomes more complex. While certain gait patterns can be associated with certain tasks, e.g. climbing stairs with a handrail, or a specific speed, e.g. gallop for fast locomotion, such discrete choices as well as transitions between them can become problematic. To overcome this limitation, heuristic or simplified models can be used [247]. However, even if the contact sequence is fixed, timing of the different flight and stance phases remains an issue. While also the timings can be pre-specified, they can be an implicit [156, 217] or explicit [154, 181] outcome of the optimization. As an alternative, planning can be performed in phase space where the timing of the trajectories is done in a secondary step [37, 185].

But not only contact sequences and timings are important but also their locations, especially in uneven terrain. Early fixed gait planners and controllers resort to compliant force controllers and heuristic reactive planners to cope with terrain [11, 82]. Active terrain handling in motion planning is a major challenge. Roughly speaking there are three common options for contact planning in rough terrain. First, a (random) sampling based planner can try to find a sequence of collision free footholds in the terrain [138, 183, 193, 232, 241]. Alternatively, the terrain can be handled in a continuous manner, e.g. by extracting a reward function based on some contact quality metric [113, 248]. As a third option, a discrete set of possible contact patches can be extracted which then allows for using a discrete search approaches. In the latter approach a quality metric can be assigned to each patch, allowing a combination of discrete and continuous optimization [46, 128]. All methods are usually employed as part of a hierarchical motion planning frameworks where first footholds are planned prior to the motion plan for the entire robot [102, 113, 128, 236, 247]. The benefit of a hierarchical planner is that each level solves a smaller, simpler subproblem. The footstep planner often resorts to simplified kinematic or dynamic models or other heuristics. However, it is not ensured that the higher level motion planners are able to find a dynamically and kinematically consistent trajectory. There are methods that can optimize over dynamics as well as contacts sequences, timings and locations simultaneously [156, 217]. However, these methods are gradient based approaches, i.e. they rely on smooth contact information [217] or physics [156]. Approaches like the former one have not yet been demonstrated on complex terrain while the latter violates physics and kinematics, limiting transferability to hardware.

Feedback Control and Trajectory Stabilization for Legged Robots

When it comes to legged robots, stabilization of a fixed set point or a trajectory is a complex problem. A legged robot is subject to often unilateral contacts and other complex constraints such as friction cones, actuator limits etc. This forces a feedback controller to reason about both the dynamics of the robot as well as contacts. One option is to assume that contacts are generally elastic. For very soft materials [10] admittance control, where we try to control the interaction force based on the deformation of very stiff materials, can be an option. However, given the stiffness of most state-of-the-art robots, the dynamics of the contacts based on the deformation of the contact bodies have a bandwidth that greatly exceeds the bandwidth of common actuators. Therefore, an admittance control approach is often infeasible. Additionally, position controlled joints can be a problem with unexpected contacts which are very likely given uncertain state estimates or complex terrain. The robot will react stiff to these disturbances which can affect stability of the overall system.

To avoid the stiff contact problem, compliant impedance based control has become very popular [11, 82] where the robot's joints are force controlled. Force control can be combined with simple model free approaches such as Virtual Model control [191]. When a fairly accurate dynamic model of the robot is available, inverse dynamics control can be applied, where feedforward joint torques are computed based on a desired acceleration. For legged systems in contact, projection based inverse dynamics have become popular [2, 152] which are equivalent [200] to Operational Space Control [120]. These methods assume rigid contacts and project the dynamics into the contact constraint satisfactory subspace. This allows for computing the inverse dynamics without explicitly considering the current or resulting contact forces. One reason why these approaches have become more popular than standard inverse dynamics is that on physical systems the contact configuration is often easier to estimate than the contact forces. This is especially true in the absence of end-effector force sensing. The projection is based on the assumption that the endeffectors in contact have zero velocity. However, these methods do not consider that often contacts are unilateral and subject to friction limits. Therefore, it is not guaranteed that for a given desired acceleration there exists a solution to the inverse dynamics problem that satisfies all contact constraints. To overcome this issue, a constrained optimization based approach solving Quadratic Programs (QPs) has been proposed [199]. Furthermore, robust inverse dynamics approaches have been developed [49].

Any inverse dynamics controller needs to be provided with a desired acceleration. Given a full state trajectory, we can compute a full state feedback controller which outputs a whole-body acceleration. However, such a full state feedback controller might not be available or not even desirable. Desired accelerations are often computed by reactive control laws expressed in "task space", e.g. an end-effector frame or the whole body frame. They are then transformed to coordinate (or joint) space using the Operational Space Control [120] framework. Legged robots often have to fulfill multiple tasks at the same time. Such tasks can be to maintain balance, i.e. controlling the center of mass position and orientation [101, 128], move limbs to certain locations, create certain contact forces etc. The tasks do not necessarily involve the entire state space of the robot and sometimes can be realized in different ways. However, they can also be conflicting requiring to trade off between them or to introduce a hierarchy. Therefore, the Hierarchical Task

Space [206] formulation has been created, where lower priority tasks get projected in the Nullspace, i.e. the remaining “degrees of freedom”, of higher priority tasks. Such a hierarchical approach can also be applied to constrained optimization based inverse dynamics [101, 128].

These inverse dynamics approaches are often combined with simple feedback controllers or sometimes even Optimal Control approaches such as LQRs [101] which generate the desired acceleration. As such, these approaches are purely reactive. However, reference trajectories can also be computed using Trajectory Optimization in task space, which e.g. leads to “centroidal dynamics” approaches (e.g. [102]) as discussed in the previous subsection. Optimizing trajectories in task space nicely reduces the dimensionality of the problem, allowing even for receding horizon computations [102]. However, as an inverse dynamics approach the contact sequence is assumed to be given, which prevents such approaches from discovering behavior like reactive stepping.

With the exception of heuristic based methods, reactive stepping can only be achieved with a receding horizon motion planner. However, no existing motion planning or Numerical Optimal Control approach that optimizes over contact locations and timings can be run in true MPC mode, i.e. solve the full planning and control problem in a single control cycle. Therefore, reactive or receding horizon inverse dynamics planners play an important role to stabilize the robot as well as possible in a given contact configuration while the contact aware planner computes an updated plan.

Motion Planning and Control for UAVs and Ballbots

While both, a ballbot and a UAV are nonlinear systems, they can be well approximated with linear systems and linear controllers show good performance for tracking fixed setpoints or trajectories [29, 171]. Therefore, for most applications such simple feedback controllers are sufficient.

In comparison to legged robots, motion planning for these systems benefits from the fact that both ballbots and UAVS are lower dimensional systems and usually do not make contact with the environment (except for the ball in ballbots). This allows geometric planning of trajectories for UAVs [64]. Geometric planning can also be applied to ballbots. In [171] a parameter optimization for trajectories based on hyperbolic secants is presented. As usual, such a dimensionality reduction limits the solution space to a specific subset of motions, affecting generality. Additionally, trajectories are planned in state space rather than input space, require to solve the inverse control problem for which a solution is not guaranteed to exist. This also affects the work presented in [170]. While reasoning about the dynamics of ballbots and UAVs, researchers have also discovered that they are (partially) differential flat systems [45, 209] even for more complex configurations [214]. Differential flatness greatly simplifies the planning problem and allows for fast trajectory planning [150]. However, deriving a differentially flat representation of a system can be challenging and thus this approach cannot necessarily be transferred to other robots such as legged systems. While there exist some approaches that aim at automating this process [207], they are computationally expensive. Additionally, not all systems are partially or fully differentially flat. As shown e.g. in [161] only special types of Rigid Body Dynamics systems are differentially flat. Motion planning for UAVs can also efficiently be tackled with randomized planners [233]. While all the above mentioned specialized motion planning solvers for ballbots and UAVs perform very well in practice, they all demonstrate a significant amount of engineering and reasoning about the specifics of a problem. Therefore, they do not fulfill our vision of a generalized approach. However, they still provide a great benchmark for our development.

Trajectory Optimization is also becoming popular in UAVs. As in legged robots, some approaches rely on simplified dynamic models and results are impressive [98]. Due to the relatively low state space dimensionality of UAVs, Numerical Optimal Control using a full Rigid Body Dynamics model can be applied efficiently as well as shown for planning in [182] and for MPC in [116]. This also applies to systems where UAVs are augmented with manipulators. [116] (published after the work on MPC for UAVs in this thesis [165]) shows that Numerical Optimal Control is a very effective method for combining motion planning and control in a single algorithm rather than resorting to a classical seperated approach.

Since there exists only a limited number of ballbot platforms, a fully dynamic approach for Numerical Optimal Control for motion planning on such systems has, to the best of our knowledge, not been tackled before. The only example of such an approach, apart from this thesis, is work related to this thesis [182].

1.4.2 Part B: Applying Numerical Optimal Control to Mobile Robots

The second part of this thesis is distributed between three relatively independent subtopics. Therefore, a general literature review for all of these subtopics is not applicable. Therefore, we provide a short high level literature review here, whereas detailed related work is covered more extensively in the papers themselves.

Derivatives of Rigid Body Dynamics

As we have seen in Subsection 1.4.1, gradient-based Optimal Control is frequently applied to Rigid Body Dynamics systems. Also, state estimation or parameter optimization rely heavily on derivative information. Due to the nonlinearity of Rigid Body Dynamics, derivatives are often tackled using numerical differentiation [61, 181] since it is simple to setup. However, the computational overhead is significant. As a result, analytic formulations for derivatives of Rigid Body Dynamics have been derived [81]. However, the resulting expressions are fairly complex to implement and no open source implementation is available. Also, higher order derivatives of Rigid Body Dynamics pose an issue. This is one potential reason why first-order Numerical Optimal Control problems are often favored over second-order methods. To avoid the derivative problem altogether, derivative-free algorithms such as [141] have been developed. While they do not require derivatives and can show convergence comparable to second-order methods, their computational effort is significant. Another way of avoiding derivatives is to resort to sampling based or black box optimization approaches [35, 108, 195, 221]. However, since gradient information is not leveraged, a significant amount of sampling is required, greatly affecting runtime. Many of the optimal control toolboxes reviewed in Subsection 1.4.1 thus leverage Automatic-Differentiation. However, these toolboxes usually require models in specific modelling languages in which efficient implementations of Rigid Body Dynamics algorithms [72] are not available. On the other hand, many tools that implement these algorithms [44, 73, 139, 208, 224] are not compatible with Auto-Differentiation tools, despite the availability of a large selection of such tools.

Simulators for Rigid Body Dynamics

Simulators are very well established in robotics research. Most general purpose simulators are based on physics engines that implement Rigid Body Dynamics algorithms. Often, these engines originate from the field of computer games and animation [44, 212] which value speed and numerical stability higher than physical accuracy or control relevant effects. However, there are some engines that have been developed specifically for robotics [139, 224]. While not complete physics engines, Rigid Body Dynamics tools such as [73, 79] can be augmented with simple integrators and contact models to act as physics engines. Physics engines are usually wrapped with graphical user interfaces and robotic tools to form complete robotics simulators [124, 202]. However, most simulators are designed from a relatively "high-level" perspective, meaning they are very well suited for evaluating high level motion planning and kinematic planning. However, they often lack a control focus. By default, most simulators are fully deterministic. However, physical systems are subject to stochastic processes such as measurement noise, sensor biases or disturbances in the system. While it is perfectly feasible to include such stochastic models, most simulators do not contain elaborate sensor or disturbance models. Also, actuator dynamics or communication delays of the robots input/output system are usually not considered. There are a few exceptions [124, 224] that show a development in this direction. However, these efforts are far from complete. For all the aforementioned reasons, there can be significant difference in the behavior of a physical systems and their simulation counterparts. Despite this fact, simulations are still frequently used for evaluating controller performance.

State Estimation for Mobile Robots

To successfully execute Numerical Optimal Control on robots, we need an estimation of their state. Usually mobile robots are equipped with intrinsic sensors such as encoders and force sensors that measure part of the state of the system. Inertial Measurement Units (IMUs) which have become very affordable in recent years provide orientation estimates out-of-the-box. Odometry or kinematics (in the case of legged systems), which are frequently fused with raw measurements from an IMU

[19, 203], can provide an estimate of the position or entire pose of the robot. Also, visual inertia systems [20, 77] or point cloud based solutions [42, 186] can be used to estimate the pose. However, in both cases, the position estimates are usually with respect to some arbitrary initial frame. Thus, we do not have an estimate of the relative pose between the robot and any part of the workspace. However, this is central to any task where the robot is meant to interact with its environment. There are various mapping approaches available either in 3D e.g. [104] or 2.5D elevation mapping, e.g. [66, 121] that create a representation of the environment at runtime. While these approaches are reasonably accurate, they do not provide semantic information about points of interest in the workspace. There are also methods that provide semantic information such as e.g. [4, 226]. However, they are relatively complex for our needs. Furthermore, these approaches are not necessarily robust and computationally expensive. As an alternative, motion capture systems such as [177, 192, 227] can be used. These systems provide highly accurate measurements of any object or robot that is equipped with suitable markers. The issue is that these systems are expensive, tedious to setup and calibrate and/or do not work outdoors.

Chapter 2

Contributions

This chapter describes the main contributions of the thesis. We have grouped the publications into two parts found in Chapters 3 and 4. Below, we provide an overview of the contributions made in the according publications. We also briefly put each publication into context and relate them to one another as well as to the overall thesis.

2.1 Summary of Contributions

2.1.1 Part A: Efficient Numerical Optimal Control

The first part of the conducted research constitutes a major step towards running Numerical Optimal Control online on robotic hardware. This step is enabled by implementing an efficient nonlinear Optimal Control solver based on the well established algorithms Sequential Linear Quadratic (SLQ) control and iterative Linear Quadratic Regulator (iLQR). Due to the efficiency of the solver, we are able to run SLQ in Model Predictive Control fashion on four different mobile robots, outperforming classical Model Predictive Controllers in computational speed while solving tasks that can hardly be tackled with classical control approaches. When applying the same approach to a legged robot, we are able to not only generate motions but also optimize for contact timings and sequences. As a result, our approach is able to discover different periodic gait patterns of a quadrupedal robot, which has not been demonstrated before in conjugation with a fully dynamically consistent motion planner. Furthermore, we apply the same approach in MPC fashion on quadruped hardware, which is the first application of whole body MPC through contacts for dynamic tasks on legged hardware. While SLQ and iLQR in its original form cannot handle constraints, we propose a method to include state and input constraints in our formulation without breaking the linear time complexity. This enables us to replace an explicit, stiff contact model by numerically well behaving constraints. The constraint formulation itself is general and extends beyond the contact problem. The contributions of the first part of this work can be summarized as follows:

Contr. A1 This work proposes a simple, yet effective way to incorporate state and input constraints into SLQ without breaking linear time complexity, tackling one of the major drawbacks of this class of algorithms.

Contr. A2 This is one of the earliest works of handling contacts in SLQ-type algorithms as constraints rather than smooth contacts.

Contr. A3 To the best of our knowledge, this work created the first open source C++ implementation of SLQ/iLQR.

Contr. A4 The same Model Predictive Control algorithm is demonstrated on three different robot types: two quadrupeds, a hexrotor and a ground robot.

Contr. A5 This work is the first demonstration of SLQ applied to locomotion on legged hardware, both as a planner and as MPC.

Contr. A6 We apply Numerical Optimal Control to highly dynamic tasks under contact changes, which forms a contrast to previously published work where tasks where usually quasi-static or without contact change.

Contr. A7 SLQ is demonstrated to be able to discover periodic gait patterns and adapt these when running in MPC fashion.

Contr. A8 This is the first application of SLQ for large time horizon MPC on mobile and legged robots.

2.1.2 Part B: Applying Numerical Optimal Control To Mobile Robots

The second part of this work tackles issues related to applying Numerical Optimal Control to mobile robots. Related work often falls short in presenting hardware experiments, not because of the underlying approach but because implementing Optimal Control on a physical systems comes with additional challenges. The relationship and interaction of these practical issues with the aforementioned fundamental question are far from trivial. Hardware experiments ruthlessly uncover these interactions and addressing them rigorously is key to the success of hardware test. While not the main focus of this work, this part contains important contributions in solving practical challenges in executing Numerical Optimal Control to physical systems. By avoiding shortcuts but instead tackling the fundamental problems involved, the contributions of this part extend beyond applications of Numerical Optimal Control.

Numerical Optimal Control usually requires gradient information from the underlying system. While gradients can be computed numerically, this is an expensive process. Analytical gradients are error prone and tedious to compute for complex systems. Hence, we introduce Auto-Differentiation into the field of Rigid Body Dynamics. While Auto-Differentiation is a well established tool in the numerical control community, it is not very well adopted in the robotics community, especially not in legged locomotion. This work carefully analyzes the potential performance gain for Auto-Differentiation for Rigid Body Dynamics algorithms and illustrates it with practical examples which would not be feasible without Auto-Differentiation. We also propose a reference implementation based on well established tools and provide it as open source software. This framework is not only useful for Optimal Control but also other derivative-based approaches including estimation or parameter optimization.

As mentioned before, results in the field of Numerical Optimal Control are often only demonstrated in simulation. Therefore, we investigate the expressiveness of Rigid Body Dynamics simulations regarding control performance. Based on a theoretical linear control analysis, we demonstrate that state-of-the-art physics simulations are – in their current form – not suitable for assessing feedback control stability. We also propose how to change our simulators to overcome this issue.

The last contribution aims at solving one of the main challenges of mobile robotics namely the localization of themselves with respect to their workspace. This problem is not only essential to Optimal Control but to any motion planning or control approach. Instead of resorting to expensive motion capture systems, we propose a lightweight localization system that relies on visual and inertial sensors, already present in most robots. We propose an EKF-SLAM approach based on 6 degree of freedom landmarks compared to the regular 3 degree of freedoms. These landmarks are based on paper printable fiducials. While localization based on fiducials is common, this is the first visual-inertial sensor fusion with fiducials, solving three major issues of fiducial based localization: occlusion, motion blur and pose ambiguity.

The contributions of the second part of this work can be summarized as:

Contr. B1 We propose the usage of Auto-Differentiation for control and optimization based on Rigid Body Dynamics. Trajectory Optimization examples illustrate the advantages of the approach.

Contr. B2 We extend the Open Source Rigid Body Dynamics framework RobCoGen with support for Auto-Differentiation which makes it the first Rigid Body Dynamics C++ engine to support Auto-Differentiation.

Contr. B3 This work analyzes physics engine from a control perspective and demonstrates why they are not suitable for assessing control stability.

Contr. B4 We develop an cost-efficient open-source framework for localizing a mobile robot with respect to its workspace. By not relying on a specific motion model, this system is applicable to most mobile robots.

Contr. B5 A visual-inertial approach is proposed that estimates fiducial and robot poses simultaneously, solving the issues of occlusion, pose ambiguity or motion blur of vision-only fiducial localization.

Contr. B6 An extension of standard EKF-SLAM from 3-DoF to 6-DoF landmarks is proposed.

2.2 Publication Overview – Part A

Paper I

Michael Neunert, Cédric de Crousaz, Fadri Furrer, Mina Kamel, Farbod Farshidian, Roland Siegwart, Jonas Buchli, "Fast Non-Linear Model Predictive Control for Unified Trajectory Optimization and Tracking". In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

Summary

This paper shows how SLQ can be applied as a MPC for trajectory optimization and tracking on two mobile platforms, namely the ballbot Rezero, described in Subsection 1.3.3, and the hexrotor "Firefly" described in Subsection 1.3.4. The successful hardware experiments on these two test platforms already underline the capabilities of Trajectory Optimization: We apply the same approach to a flying and a ground robot. Due to the low dimensionality of their state space, which is only 10 for the ballbot and 12 for the UAV, we can solve the Trajectory Optimization problem online at high rates of up to 100 Hz for time horizons of several seconds. The benefit of SLQ is that apart from the trajectory it also provides stabilizing feedback gains that match the trajectory. These feedback gains correspond to the solution to the finite-horizon time-varying LQR solution and allow us to run an additional stabilizing controller at higher rates. Thus, unlike conventional MPC, we do not have to solve the Trajectory Optimization problem exactly at control rate.

Contribution

To the best of our knowledge this has been the first demonstration of Trajectory Optimization for long time horizon Model Predictive Controller on mobile robots which does not resort to simplified models (*Contr. A8*). The publication shows hardware experiments for two out of three platforms used throughout this thesis (*Contr. A4*). This work also serves as an application example of our custom SLQ/iLQR solver implementation, which was later open sourced (*Contr. A3*).

Context

This initial work on applying SLQ to physical robots serves as a test-bench and convincingly illustrates the capabilities of this approach. The obtained results are a great motivation to apply SLQ to more complex problems such as legged locomotion under contacts shown in Paper II. Furthermore, the demonstrated window passing task of the UAV shows that penalty-based soft constraints work well in practice. We later exploit this in Paper III in a more rigorous fashion in our extension of SLQ to incorporate state and input constraints. Due to the simplicity of the models, we can use analytical derivatives generated by a symbolic toolbox. Thus, linearization of the system dynamics is not an issue in this particular case which is why Auto-Differentiation, as presented in Paper V, is not used in this work. The hexrotor is executing several tasks that require a localization in the workspace. As the localization method described in Paper VII had not been developed yet, we resorted to an external motion capture system.

Paper II

Michael Neunert, Farbod Farshidian, Alexander W. Winkler, Jonas Buchli, "Trajectory Optimization Through Contacts and Automatic Gait Discovery for Quadrupeds". In *Robotics and Automation Letters (RA-L)*, 2017.

Summary

Motivated by the potential of SLQ applied to a UAV and a ground robot, this paper aims at bringing SLQ to more complex, legged systems. Apart from the increase in model complexity, legged systems also have to make and break contacts with the environment to perform a task. To avoid discontinuities in state space, which SLQ cannot handle well, we propose a penalty based contact model. A smooth contact model allows for automatically discovering contact sequences and timings, which in this case allowed us to discover gaits and other motions requiring dynamic contact switches. We illustrate how this additional freedom can be leveraged by the solver for different tasks and initial conditions.

Contribution

This paper shows that SLQ can be used for discovering periodic gait patterns (*Contr. A7*). Furthermore, it is the first time SLQ has been applied to locomotion tasks on legged systems (*Contr. A5*) and thus, together with Paper I, we complete the contribution of running the same algorithm on three different platforms (*Contr. A4*). The demonstrated tasks are dynamic and include dynamic contact changes (*Contr. A6*). Again, our custom C++ SLQ solver has been used in this work (*Contr. A3*), leading to very short optimization times.

Context

In contrast to Paper I, the state and input dimensionality is significantly increased. This allows for more complex motions to be generated. While this is a strength of the approach, it also requires more cost function tuning. Furthermore, an explicit contact model requires tuning as well. This has motivated us to look for an easy to tune, numerically robust approach presented in Paper III. Also, due to the model complexity, we could only partially use analytical derivatives resulting in a significant overhead in computation for linearization. This has led to the effort of obtaining faster derivatives in Paper V. While not explicitly described in the paper, in demos we are using our localization system presented in Paper VII to obtain a relative pose estimate between the quadruped and the pallet for the push task presented in the paper. Since HyQ is a powerful robot that could potentially break itself during hardware experiments, we performed initial tests in our improved simulation environment Paper VI to assess performance before hardware experiments.

Paper III

Michael Neunert, Farbod Farshidian, Jonas Buchli, “Efficient Whole-Body Trajectory Optimization Using Contact Constraint Relaxation”. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS)*, 2016.

Summary

Most SLQ-type approaches model contacts by using an explicit contact model. One reason for this is that algorithms of this class cannot handle state constraints without sacrificing linear time complexity, the major advantage over NLP methods. However, a contact model is difficult to tune and platform dependent. To overcome this limitation, we investigate if we can treat contacts as constraints to our solver. In order to handle constraints in our SLQ formulation, we add violation terms to our cost function. We prevent our solver from violating this “soft constraint” by increasing its weighting over the course of iterations. While we are not using a log barrier, this method is inspired by interior point solvers which also convert constraints to loss functions. We demonstrate that this approach is very effective in reducing constraint violation to a negligible amount. Using a constraint-based contact handling greatly reduces tuning efforts, which further contributes to our vision of platform-independent, transferable Optimal Control.

Contribution

This publication presents how state and input constraints can be handled in an SLQ-type algorithm without loosing linear time complexity (*Contr. A1*), overcoming one of the major drawbacks of these methods. When dealing with contacts, a constraint-based approach avoids the complex tuning process of a soft contact model and does not require to make a trade-off between physicality and numerical properties (*Contr. A2*).

Context

This work allows us to compare a constraint-based contact handling with an explicit contact model shown in Paper II. In this approach, we rely on fixed contact timings. While this prevents us from optimizing for contact timings, it has two great advantages. First, tuning is greatly simplified. For all motions on the same platform, we can use a single cost function to generate all gait types and no tuning of contact model parameters is required. Second, having control over the timings can be beneficial if we want to optimize for a given gait pattern or adjust the stepping frequency. E.g. in trotting, a higher stepping frequency does not yield lower costs in our metric but can greatly improve robustness, which SLQ-type algorithms do not account for. Since our constraints are added to the cost function, we need to provide first and second order derivatives. Since the involve kinematics, we already made us of our Rigid Body Dynamics Auto-Differentiation framework presented in Paper V.

Paper IV

Michael Neunert, Markus Stäuble, Markus Giftthaler, Camine D. Bellicoso, Jan Carius, Christian Gehring, Marco Hutter, Jonas Buchli, "Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds". In *ArXiv preprint arXiv:1712.02889 [cs.RO]*, 2017.

Summary

Previous work demonstrated both the potential of SLQ for whole-body control through contacts as well as running SLQ in MPC fashion on robotic hardware. Hence, this paper combines both approaches and evaluates them on two quadrupeds, HyQ and ANYmal. In this work, we decide to use the explicit contact model in order for the algorithm to be able to reason about contacts. Hence, without specifying contact locations, timings or patterns explicitly, we demonstrate very dynamic tasks such as squad jumps or trotting. Furthermore, by disturbing the robots during execution, we can show how the MPC controller is able to adapt gait patterns and contact timings. One key sucess factor is also the achieved speed up of the algorithm in comparison to previous implementations, reaching MPC rates of up to 180 Hz for 500 ms time horizons.

Contribution

This paper is the first of its kind that shows Model Predictive Control through contacts for locomotion and dynamic motions using a whole body model (*Contr. A6*) being directly applied to robotic hardware (*Contr. A5*). By actively reasoning about contacts, it is the first time that SLQ-MPC is shown to be able to adjust contact sequences and timings on the fly, both for gaits and single motions *Contr. A7*. The SLQ solver combined with Direct Multiple Shooting ideas, all open sourced (*Contr. A3*) is able to run at unmatched rates of 180 Hz at a time horizon of 500 ms. By running hardware experiments on two very different quadrupeds, namely hydraulically actuated, large scale HyQ and electrically actuated, medium scale ANYmal, we underline the generality of our chosen approach (*Contr. A4*).

Context

This publication being chronologically the newest, combines most of the work of this thesis. Motivated by the results of applying MPC on other types of robots in Paper I as well as the demonstrated potential of SLQ for legged robots in Paper II, this paper is the natural combination of these two. The results in this paper were further facilitated by the experience gained during hardware experiments of SLQ on hardware presented in Paper III. The last missing piece is fast enough runtime such that the whole-body MPC approach scales to complex quadruped models. One key ingredient here is Auto-Differentiation presented in Paper V.

2.3 Publication Overview – Part B

Paper V

Markus Giftthaler†, Michael Neunert†, Markus Stäuble, Marco Frigerio, Claudio Semini and Jonas Buchli (†these authors contributed equally to this work), "Automatic Differentiation of Rigid Body Dynamics for Optimal Control and Estimation". In *Advanced Robotics*, 2017.

Summary

Obtaining efficient derivatives is a major challenge for gradient based approaches which are frequently used in Optimal Control, estimation and optimization. This is also the case for the Optimal Control approaches that heavily depend on linear approximations of the system dynamics. Prior to this work, these derivatives had to be either derived manually, computed with numerical differentiation or generated with a symbolic toolbox. However, as the paper illustrates, these methods come with major drawbacks either in efficiency or accuracy. This paper overcomes these limitations and provides both fast and accurate derivatives in an automated fashion.

Contribution

While Auto-Differentiation is a common tool in Optimization, it has not yet been studied for Rigid Body Dynamics. This work presents a rigorous evaluation as well as application examples that underline the potential performance gain (*Contr. B1*). Current C++ frameworks for Rigid Body Dynamics algorithms lack support for Auto-Differentiation. Therefore, we add Auto-Differentiation compatibility to the open-source Rigid Body Dynamics code generator "RobCoGen", making it the first Auto-Differentiation compatible Rigid Body Dynamics framework in C++ (*Contr. B2*).

Context

While this approach has only been developed after the work presented in Part A, this paper replicates some of the examples presented in Paper II. It shows the speed-up that Auto-Differentiation can provide to Nonlinear Optimal Control problems.

Paper VI

Michael Neunert, Thiago Boaventura, Jonas Buchli, "Why off-the-shelf physics simulators fail in evaluating feedback controller performance - a case study for quadrupedal robots". In *Advances in Cooperative Robotics: Proceedings of the 19th International Conference on Climbing and Walking Robots and Support Technologies for Mobile Machines (CLAWAR), World Scientific*, 2016.

Summary

Most motion planning and control algorithms are often evaluated in simulation, either as a verification step to reduce risks in hardware experiments or as a replacement for experiments themselves. Especially with planners that do not use simplified models, the verification is often conducted with the same model as also used for planning. This clearly limits the validity of simulation results but unfortunately is still accepted practice. This issue is further aggravated by off-the-shelf simulators not modelling sensor noise, actuator dynamics or signal delays. However, these properties are well known to be important when accessing control stability. Therefore, in this paper, we examine physics engines from a control perspective and analyze how informative they are for assessing control stability. This will eventually tell us how useful they are to validate our own approaches.

Contribution

Simulators are still considered acceptable means of assessing control performance. In this work, we demonstrate that off-the-shelf simulators, in their current form, are not suitable to assess control stability not even for very simple systems or controllers (*Contr. B3*). Additionally, we show how physics engines can be easily improved to account for these shortcomings.

Context

While all publications in this thesis that focus on Numerical Optimal Control are validated with hardware experiments, this work on simulations helped to assess the stability of the used controllers before executing them on the physical hardware. This ultimately helped to speed up the development and prevent hardware damages. Furthermore, it provides an insight which additional effects, such as actuator dynamics, we should maybe included in the model used in the Trajectory Optimization step. Most importantly however, this work shows that simulation only results are not sufficient for accessing the stability and robustness of Numerical Optimal Control approaches.

Paper VII

Michael Neunert, Michael Bloesch, Jonas Buchli, “An Open Source, Fiducial Based, Visual-Inertial Motion Capture System”. In *Proceedings of the 19th International Conference on Information Fusion (FUSION)*, 2016.

Summary

The Trajectory Optimization approaches optimize trajectories for all dimensions of the state space. Therefore, we need to measure or estimate the entire state of our robots. While most of our platforms and many other platforms provide a reasonable state estimate, position estimates remain an issue. Usually, they are taken with respect to an arbitrary initial frame and are often integrated odometry or kinematic measurements. Therefore, they are prone to drift. Furthermore, they do not provide a localization with respect to the robot’s environment or objects in it, which can be a requirement of some tasks. In this work, we propose a visual-inertial estimation approach that provides a relative localization of the robot with respect to artificial landmarks that can be attached to “locations of interest” in the robot’s workspace.

Contribution

The proposed system relies on sensing that is already available on most robots and thus is more cost efficient than motion capture systems (*Contr. B4*). Affordable paper printed fiducials allow to mark points of interest in the robot’s workspace and a general motion model allows for implementing the system on most mobile robots (*Contr. B4*). In contrast to existing fiducial based localization systems, we avoid solving the ill-posed problem of extracting the fiducial’s pose from a single measurement of its corners but rather estimate its pose over multiple observations while considering measurement uncertainty. This solves the ambiguity problem of fiducial based systems (*Contr. B5*). Since our landmarks are characterized by a pose rather than just positions, we propose an extension of the regular EKF-SLAM approach to handle 6-DoF rather than 3-DoF landmarks (*Contr. B6*). Furthermore, we include inertial measurements which provide accurate state estimates during fiducial occlusions or fast motions that are subject to visual motion blur (*Contr. B5*).

Context

In principle the developed system could have been used in all experiments presented in Part A. However, since this system has only been developed after some of these publications have already been published, it could only be used in Paper II and Paper III. During the hardware experiments in Paper III the system has been used to localize the pallet for the push task. While not published, both systems have been used jointly in many live demonstrations of HyQ throughout the course of the doctoral studies.

2.4 Complete List of Publications

This is a complete list of all publications written over the course of the PhD studies including non-first author publications. This list is sorted by author list position of the PhD candidate and year.

- M. Giftthaler*, **M. Neunert***, M. Stäuble, and J. Buchli. The Control Toolbox - an open-source C++ library for robotics, optimal and model predictive control. `https://adrlab.bitbucket.io/ct`, 2018. arXiv:1801.04290 [cs.RO], (*these authors contributed equally to this work.)

- **M. Neunert**, M. Stäuble, M. Giftthaler, C. D. Bellicoso, J. Carius, C. Gehring, M. Hutter, and J. Buchli. Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds. In *ArXiv preprint*, 2017. To be published in Robotics and Automation Letters (RA-L) 2018 and to be presented at the IEEE International Conference on Robotics and Automation (ICRA) 2018

- M. Giftthaler*, **M. Neunert***, M. Stäuble, M. Frigerio, C. Semini, and J. Buchli. Automatic differentiation of rigid body dynamics for optimal control and estimation. *Advanced Robotics*, 31(22):1225–1237, 2017. doi: 10.1080/01691864.2017.1395361. URL `https://doi.org/10.1080/01691864.2017.1395361`. (*these authors contributed equally to this work.)

- **M. Neunert**, F. Farshidian, A. W. Winkler, and J. Buchli. Trajectory optimization through contacts and automatic gait discovery for quadrupeds. *IEEE Robotics and Automation Letters (RA-L)*, 2(3):1502–1509, 2017. doi: 10.1109/LRA.2017.2665685

- **M. Neunert**, F. Farshidian, and J. Buchli. Efficient whole-body trajectory optimization using contact constraint relaxation. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 43–48, 2016. doi: 10.1109/HUMANOIDS.2016.7803252

- **M. Neunert**, C. De Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli. Fast Nonlinear Model Predictive Control for Unified trajectory optimization and tracking. In *IEEE International Conference on Robotics and Automation*, pages 1398–1404, 2016. doi: 10.1109/ICRA.2016.7487274

- **M. Neunert**, M. Bloesch, and J. Buchli. An Open Source, Fiducial Based, Visual-Inertial Motion Capture System. In *19th International Conference on Information Fusion (FUSION)*, pages 1523–1530, 2016

- **M. Neunert**, T. Boaventura, J. Buchli, and A. Dynamics. Why off-the-shelf physics simulators fail in evaluating feedback controller performance - a case study for quadrupedal robots. *Advances in Cooperative Robotics: Proceedings of the 19th International Conference on Climbing and Walking Robots and the*

Support Technologies for Mobile Machines, CLAWAR 2016, pages 464–472, 2016

- **M. Neunert**, F. Farshidian, and J. Buchli. Adaptive real-time nonlinear model predictive motion control. In *IROS 2014 Workshop on Machine Learning in Planning and Control of Robot Motion*, 2014

- M. Giftthaler, **M. Neunert**, M. Stäuble, J. Buchli, and M. Diehl. A Family of Iterative Gauss-Newton Shooting Methods for Nonlinear Optimal Control. arXiv:1711.11006 [cs.SY], 2017

- F. Farshidian, **M. Neunert**, A. W. Winkler, G. Rey, and J. Buchli. An efficient optimal planning and control framework for quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017

- D. Pardo, **M. Neunert**, A. W. Winkler, and J. Buchli. Projection based whole body motion planning for legged robots. *arXiv preprint arXiv:1510.01625*, 2015

- F. Farshidian, **M. Neunert**, and J. Buchli. Learning of closed-loop motion control. In *IEEE International Conference on Intelligent Robots and Systems*, pages 1441–1446, 2014. doi: 10.1109/IROS.2014.6942746

- A. W. Winkler, F. Farshidian, **M. Neunert**, D. Pardo, and J. Buchli. Online Walking Motion and Foothold Optimization for Quadruped Locomotion. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017

- D. Pardo, L. Moller, **M. Neunert**, A. W. Winkler, and J. Buchli. Evaluating Direct Transcription and Nonlinear Optimization Methods for Robot Motion Planning. *IEEE Robotics and Automation Letters (RA-L)*, 1(2):946–953, 2016. doi: 10.1109/LRA.2016.2527062

- C. De Crousaz, F. Farshidian, **M. Neunert**, and J. Buchli. Unified Motion Control for Dynamic Quadrotor Maneuvers Demonstrated on Slung Load and Rotor Failure Tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2223–2229, 2015. doi: 10.1109/ICRA.2015.7139493

Chapter 3

Part A: Efficient Numerical Optimal Control

Paper I: “Fast Non-Linear Model Predictive Control for Unified Trajectory Optimization and Tracking”

Authors

Michael Neunert, Cédric de Crousaz, Fadri Furrer, Mina Kamel, Farbod Farshidian, Roland Siegwart, Jonas Buchli

Abstract

This paper presents a framework for real-time, full-state feedback, unconstrained, nonlinear model predictive control that combines trajectory optimization and tracking control in a single, unified approach. The proposed method uses an iterative optimal control algorithm, namely Sequential Linear Quadratic control, in a Model Predictive Control setting to solve the underlying nonlinear control problem and simultaneously derive the optimal feedforward and feedback terms. Our customized solver can generate trajectories of multiple seconds within only a few milliseconds. The performance of the approach is validated on two different hardware platforms, an AscTec Firefly hexacopter and the ball balancing robot Rezero. In contrast to similar approaches, we perform experiments that require leveraging the full system dynamics.

Published in:
Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 1398–1404, 2016

Award:
This paper has been the runner-up for the *Best PhD Paper Award 2016 of the National Centre of Competence in Research (NCCR) Robotics*.

I.1 Introduction

While recent developments in robotics progress with increasing speed, we still do not see many real world applications. One major challenge for such applications are dynamic environments. Unpredicted changes in its surroundings may require the robot to react quickly and replan its motion within a short time. Despite such sudden changes, the motions should be smooth and adjusted according to the application or current task. Many motion control problems in mobile robotics are solved by addressing trajectory planning and tracking separately. This classical approach consists of a trajectory generation block that is responsible of generating a feasible trajectory and a feedback controller policy that tries to track the generated trajectory. Since the trajectory planner is required to solve a complex planning problem, respecting the system dynamics and constraints, it cannot react very dynamically to changes. The controller on the other hand is often tracking a fixed trajectory with no authority to adjust it. Furthermore, the planner requires knowledge about the controller to precisely predict the system's behavior.

In this work, we are proposing to address this issue with an integrated approach that solves the motion planning and tracking problem by generating optimal feedforward and feedback control gains by solving a single optimal control problem. The unconstrained optimal control problem is solved using a Sequential Linear Quadratic (SLQ) solver [211], which is a very efficient iterative, nonlinear, and locally optimal approach. This kind of approach can also be favorable if the state estimation is likely to be discontinuous and thus requires fast replanning. Such scenarios frequently occur during loop closures in Simultaneous Localization and Mapping (SLAM) applications or after regaining measurements after sensor outages, as it is often the case with Global Positioning System (GPS) measurements in urban environments. Being able to rapidly generate new intermediate trajectories instead of letting the controller act on large state deviations prevents aggressive maneuvers. Therefore, we propose to use SLQ in a Model Predictive Control (MPC) setting by repeatedly solving the optimal control problem at run time.

Related Work

Recently, MPC approaches have become more popular in robotics [5, 30, 89, 118, 137], thanks to increasing computation capabilities and novel efficient algorithms. In [105, 194] a framework for fast Nonlinear Model Predictive Control (NMPC) is presented and used in [115] for fast attitude control of a Unmanned Aerial Vehicle (UAV), while in [229] a code generator for embedded implementation of a linear MPC based on an interior-point solver is shown. Most of these methods are solving a constrained MPC problem, which is computationally complex and thus require a trade-off between time horizon and policy lag.

To overcome this problem, we are solving an unconstrained MPC problem using SLQ. One well-known variant of SLQ called iterative Linear Quadratic Gaussian (iLQG) is presented in [222] and its capabilities are demonstrated in various simulations. Following research [60, 217] has shown the feasibility of such

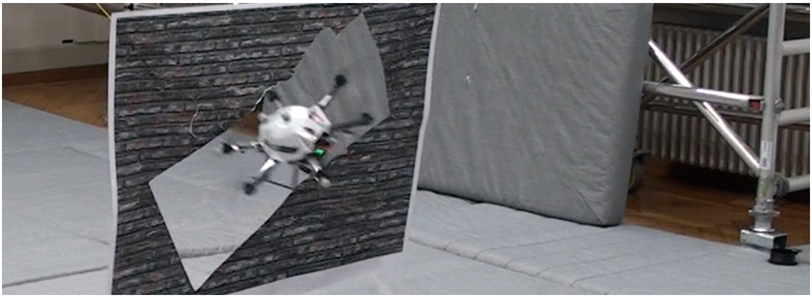


Figure 3.1: Flight through a tilted window with an Asctec Firefly hexacopter, with a simultaneously optimized trajectory and gains from the proposed SLQ-MPC.

approaches in simulation. In recent work by others [80], and the authors [69], SLQ was successfully demonstrated on hardware. However, in these projects the control is not recomputed during run time, which limits these approaches in terms of disturbance rejection, reaction to unperceived situations and imperfect state estimation. These limitations can be mitigated by resolving the optimal control problem in an MPC manner. First implementations of such an approach were demonstrated in [62, 123]. However, in this work time horizons are short and additional tracking controllers are used instead of directly applying the feedforward and feedback gains to the actuation system. We believe that by allowing SLQ-MPC to directly act on the actuation system, performance can be significantly improved. To the best of our knowledge, the presented work is the first application of SLQ-based MPC applied to highly dynamic, nonlinear mobile robots, directly controlling their actuation systems. In experiments, we demonstrate the capabilities of the approach in tasks that leverage the full system dynamics. Thus, the resulting behaviors cannot be easily reproduced by simple controller and planner schemes.

With our efficient implementation of the SLQ solver, we are able to solve the MPC problem for time horizons of multiple seconds in only a few milliseconds. This performance is unmatched by many other MPC algorithms. Yet, the proposed framework does not only provide a feedforward trajectory but also computes optimal feedback gains which can be used by a higher rate closed loop controller. Therefore, the proposed approach can effectively substitute a tracking controller while simultaneously providing the functionality as a local planner. By initializing the optimal control solver with the infinite horizon Linear Quadratic Regulator (LQR) solution, feedback gains converge to steady-state gains towards the end of the trajectory, which can stabilize the system even if the solver fails to compute an updated trajectory and updated gains in time.

I.2 Finite Horizon Optimal Control Framework

Problem Statement

In this work, we are solving an unconstrained, nonlinear optimal control problem in a nonlinear model predictive control manner. We assume a general nonlinear system

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{3.1}$$

where $x(t)$ and $u(t)$ denote the system's state and control input, respectively. $f(x(t), u(t))$ denotes the transition vector. $f(x(t), u(t))$ is assumed to be differentiable with respect to the state $x(t)$ and to the control input $u(t)$. Furthermore, we assume that all states are observable. Our goal is to find a linear, time-varying feedback and feedforward controller of the form

$$\boldsymbol{u}(\boldsymbol{x}, t) = \boldsymbol{u}_{ff}(t) + \mathcal{K}(t)\boldsymbol{x}(t) \tag{3.2}$$

with the control gain matrix $\mathcal{K}(t)$ and feedforward control action $u_{ff}(t)$. The time-varying controller is found by iteratively solving a finite-horizon optimal control problem that minimizes a cost function

$$J = h\left(\boldsymbol{x}(t_f)\right) + \int_{t=0}^{t_f} l\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right) dt \tag{3.3}$$

where $l(\boldsymbol{x}, \boldsymbol{u})$ and $h(\boldsymbol{x})$ are intermediate and terminal cost.

Overview of the Control Framework

Our control framework consists of two elements. The first element is the optimal control solver that repeatedly solves the finite horizon control problem, designing a feedforward trajectory and time-varying feedback gains. The second element is a closed loop controller which applies these gains. This second, inner control loop can run at a higher rate than the outer solver, which allows the solver to take slightly more time in designing new control trajectories, while the inner loop ensures a high control frequency. This hierarchical approach allows for rejecting disturbances and for adjusting to changes in the environment according to their time scales. Low amplitude, high frequency disturbances are handled by the feedback controller while larger amplitude, lower frequency disturbances can be compensated for by the optimal control loop. In contrast to cascaded control loops, we use a single optimal control algorithm to design the feedforward trajectory and feedback gains. Therefore, feedback gains are not fixed but tuned to match the trajectory.

Optimal Control Solver

To solve the unconstrained nonlinear optimal control problem as stated in eq. (3.1) and eq. (3.3), we use a custom solver implementing SLQ control (see algorithm 1).

Algorithm 1 SLQ Algorithm

```
Given
- System dynamics: $\boldsymbol{x}(t+1) = \boldsymbol{f}\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right)$
- Cost function $J = h\left(\boldsymbol{x}(t_f)\right) + \sum_{t=0}^{t_f-1} l\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right)$
- Initial stable control law, $\mathbf{u}(\boldsymbol{x}, t)$
repeat
    - t(0) simulate the system dynamics:
    $\tau : \boldsymbol{x}_n(0), \boldsymbol{u}_n(0), \boldsymbol{x}_n(1), \boldsymbol{u}_n(1), \ldots, \boldsymbol{x}_n(t_f - 1), \boldsymbol{u}_n(t_f - 1), \boldsymbol{x}_n(t_f)$
    - Linearize the system dynamics along the trajectory $\tau$:
    $\delta\boldsymbol{x}(t+1) = \boldsymbol{A}(t)\delta\boldsymbol{x}(t) + \boldsymbol{B}(t)\delta\boldsymbol{u}(t)$
            $\{\boldsymbol{A}(t)\}_0^{t_f-1}$, $\boldsymbol{A}(t) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}|_{\boldsymbol{x}(t),\boldsymbol{u}(t)}$
            $\{\boldsymbol{B}(t)\}_0^{t_f-1}$, $\boldsymbol{B}(t) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}|_{\boldsymbol{x}(t),\boldsymbol{u}(t)}$
    - Quadratize cost function along the trajectory $\tau$:
    $\tilde{J} \approx p(t) + \delta\boldsymbol{x}^T(t_f)\boldsymbol{p}(t_f) + \frac{1}{2}\delta\boldsymbol{x}^T(t_f)\boldsymbol{P}(t_f)\delta\boldsymbol{x}(t_f)$
            $+ \sum_{t=0}^{t_f-1} q(t) + \delta\boldsymbol{x}^T\boldsymbol{q}(t) + \delta\boldsymbol{u}^T\boldsymbol{r}(t)$
            $+\frac{1}{2}\delta\boldsymbol{x}^T\boldsymbol{Q}(t)\delta\boldsymbol{x} + \frac{1}{2}\delta\boldsymbol{u}^T\boldsymbol{R}(t)\delta\boldsymbol{u}$
    - Backwards solve the Riccati-like difference equations:
            $\boldsymbol{P}(t) = \boldsymbol{Q}(t) + \boldsymbol{A}^T(t)\boldsymbol{P}(t+1)\boldsymbol{A}(t) +$
                    $\boldsymbol{K}^T(t)\boldsymbol{H}\boldsymbol{K}(t) + \boldsymbol{K}^T(t)\boldsymbol{G} + \boldsymbol{G}^T\boldsymbol{K}(t)$
            $\boldsymbol{p}(t) = \boldsymbol{q} + \boldsymbol{A}^T(t)\boldsymbol{p}(t+1) + \boldsymbol{K}^T(t)\boldsymbol{H}\boldsymbol{l}(t) + \boldsymbol{l}^T(t)\boldsymbol{g} + \boldsymbol{G}^T\boldsymbol{l}(t)$
            $\boldsymbol{H} = \boldsymbol{R}(t) + \boldsymbol{B}^T(t)\boldsymbol{P}(t+1)\boldsymbol{B}(t)$
            $\boldsymbol{G} = \boldsymbol{B}^T(t)\boldsymbol{P}(t+1)\boldsymbol{A}(t)$
            $\boldsymbol{g} = \boldsymbol{r}(t) + \boldsymbol{B}^T(t)\boldsymbol{p}(t+1)$
            $\boldsymbol{K}(t) = -\boldsymbol{H}^{-1}\boldsymbol{G}$ % feedback update
            $\boldsymbol{l}(t) = -\boldsymbol{H}^{-1}\boldsymbol{g}$ % feedforward increment
    Line search
    - Initialize $\alpha = 1$
    repeat
        - Update the control:
        $\mathbf{u}(\boldsymbol{x}, t) = \boldsymbol{u}_n(t) + \alpha\boldsymbol{l}(t) + \boldsymbol{K}(t)\left(\boldsymbol{x}(t) - \boldsymbol{x}_n(t)\right)$
        - Forward simulate the system dynamics:
        $\tau : \boldsymbol{x}_n(0), \boldsymbol{u}_n(0), \boldsymbol{x}_n(1), \ldots, \boldsymbol{x}_n(t_f - 1), \boldsymbol{u}_n(t_f - 1), \boldsymbol{x}_n(t_f)$
        - Compute new cost:
        $J = h\left(\boldsymbol{x}(t_f)\right) + \sum_{t=0}^{t_f-1} l\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right) dt$
        - decrease $\alpha$ by a constant $\alpha_d$:
        $\alpha = \alpha/\alpha_d$
    until found lower cost or number of maximum line search steps reached
until maximum number of iterations or converged $(\boldsymbol{l}(t) < l_t)$
```

SLQ optimizes a time-varying feedforward and feedback controller with respect to a given cost function by iteratively linearizing the system around a reference trajectory and computing a quadratic approximation of the cost function around the trajectory and then backwards solving the resulting time-varying, linear quadratic optimal control problem. SLQ needs to be provided with a model of the system dynamics as shown in eq. (3.1). In case the system is not statically stable, a stabilizing controller needs to be provided. The possibly nonlinear model and the stabilizing controller are then used to generate a reference trajectory. For the following control update step around the trajectory, SLQ requires a linearized model. Such a model can be computed numerically or analytically. While numerical differentiation has been shown to work [80], we are using analytically derived linearizations [65], as they are more accurate and faster to compute.

Cost Function Structure In order to apply our optimal control solver, a cost function in the form of eq. (3.3) needs to be provided. Since in SLQ a quadratic approximation of the cost function is computed, convergence can be improved by already choosing a quadratic cost function. Additionally, quadratic costs are frequently used in control and thus researchers have developed an intuition in tuning these. The cost functions used in this work are of the following form

$$\begin{aligned} J = \bar{x}(t_f)^T H \bar{x}(t_f) + \int_{t=0}^{t_f} & \bar{x}(t)^T Q \bar{x}(t) + \bar{u}(t)^T R \bar{u}(t) \\ & + W(x,t)\, dt \end{aligned} \tag{3.4}$$

where $\bar{x}(t)/\bar{u}(t)$ represent deviations of state/input from a desired state/input. H, Q and R are the final cost, intermediate cost and input cost weighting matrices respectively. To be able to include multiple waypoints in our solver, we add an intermediate waypoint cost $W(x,t)$ defined as

$$W(x,t) = \sum_{n=0}^{N} \hat{x}(t)^T W_p \hat{x}(t) \sqrt{\frac{\rho_p}{2\pi}} \exp\left(-\frac{\rho_p}{2}(t-t_p)^2\right) \tag{3.5}$$

where N is the number of waypoints, $\hat{x}(t)$ is the deviation from current to waypoint state, W_p is the waypoint cost matrix, t_p the time at which the desired waypoint should be reached, and ρ_p the temporal spread of the waypoint. W_p and t_p can be chosen separately for each waypoint. Even though we include waypoints as soft constraints, results show that our formulation of immediate waypoints is able to enforce a certain state on a trajectory well (see fig. 3.6). At the same time, such soft constraints allow the controller to slightly violate these constraint if not dynamically feasible. This relaxes the requirements for a high-level planner to generate perfectly feasible trajectories, which is a strict requirement when using hard constraints. Including waypoints in the cost function has another advantage: the waypoint weighting matrix W allows to configure the importance of individual states which makes tuning more intuitive. Since the weighting matrix W_p does not only influence the feedforward trajectory but the feedback gains as well, SLQ also designs optimal feedback gains that match this importance weighting.

Extending SLQ to MPC

In this work, we are using SLQ in an MPC scheme. Algorithm 2 summarizes the proposed approach. The main concept is to run SLQ in an infinite loop, continuously recomputing the control trajectory while adjusting its parameters. The MPC loop is triggered by a new state measurement and is run until convergence.

State Prediction We define policy lag as the time between the initial state for which a control policy was designed and the time this policy gets executed. A long policy lag can have severe consequences on the control performance, especially for complex dynamic systems with a rapidly changing state. To compensate for a potential policy lag, one can estimate or measure the lag and predict the first state where the new policy will be executed. This state can then be used as the initial condition for the policy design.

Initialization SLQ needs to be provided with an initial controller. The convergence speed of SLQ depends on this initial guess. When running an MPC at a high rate, subsequently designed control trajectories usually do not deviate significantly from each other and thus provide a good initialization for the next design step. However, it is not trivial to initialize SLQ with the previous controller if the time horizon is changing, especially when the time horizon increases. As an alternative, SLQ can be initialized with any stable controller. Since for the presented problems SLQ converges in less than three iterations without warm starting, we initialize it with an LQR controller stabilizing the system around its initial state.

Waypoints As shown in eq. (3.5), our waypoints are localized in time. To ensure that a waypoint is reached at a specific time, we shift the waypoint time t_p in eq. (3.5) by the time elapsed since the start of the execution of the plan.

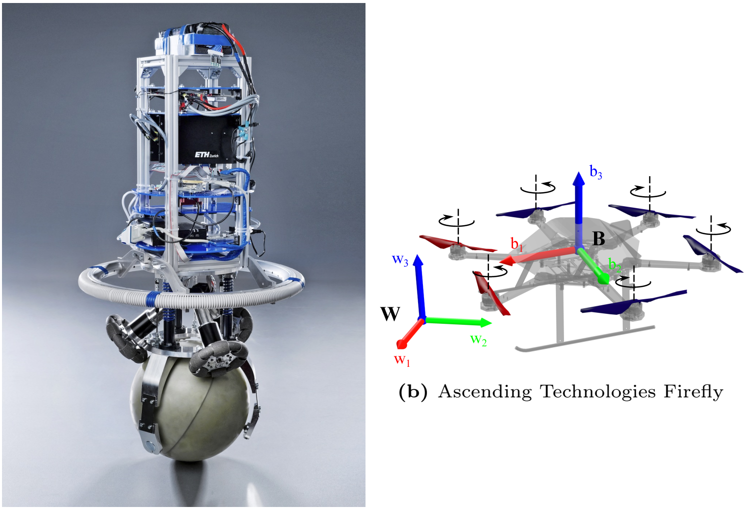
Final Cost Manually tuning the final cost H in eq. (3.4) is difficult. Thus, we set H to the solution of the infinite horizon LQR Ricatti equation. The input weighting R is chosen to be equal for both SLQ and LQR. Thus, the final gains of each trajectory correspond to the LQR gains.

I.3 Test Platforms: Ballbot and Hexacopter

Two platforms were used to demonstrate the applicability of the presented framework. The hardware as well as the dynamic models for both systems are described below.

Algorithm 2 SLQ-MPC Algorithm

```
Given
- System dynamics: $\boldsymbol{x}(t+1) = \boldsymbol{f}\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right)$
- Input cost weights $\boldsymbol{R}$ for SLQ/LQR
- State cost weights $\boldsymbol{Q}_{slq}$ for SLQ
$J = \bar{x}(t_f)^T H \bar{x}(t_f) + \sum_{t=0}^{t_f-1} \bar{x}(t)^T Q_{slq} \bar{x}(t) + \bar{u}(t)^T R \bar{u}(t) + W(x,t)$
- State cost weights for LQR $Q_{lqr}$
repeat
    Adjust Setup
    - Acquire current state $x(0)$
    - Optional: Predict state after policy lag by forward simulating dynamics with previous controller
    $x(0) = x_{t_{lag}}$
    - Optional: Update the time horizon
    - $t_f = distance(x(0), x(t_f))$
    Initialize SLQ with Controller
    - Linearize the system dynamics around current state $x(0)$:
    $\delta\boldsymbol{x}(t+1) = \boldsymbol{A}(t)\delta\boldsymbol{x}(t) + \boldsymbol{B}(t)\delta\boldsymbol{u}(t)$
    $\boldsymbol{A}(0) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}|_{\boldsymbol{x}(0),\boldsymbol{u}(0)}$
    $\boldsymbol{B}(0) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}|_{\boldsymbol{x}(0),\boldsymbol{u}(0)}$
    - Compute LQR at linearized state
    $\boldsymbol{K}(0) = lqr(\boldsymbol{Q}_{lqr}, \boldsymbol{R}, \boldsymbol{A}(0), \boldsymbol{B}(0))$
    - Initialize SLQ with LQR around current state
    $slq.init(\boldsymbol{K}(0), \boldsymbol{x}_0)$
    Optional: Update Cost Function
    - Update desired and final state $x_d(t)$, $x(t_f)$
    - Update intermediate weights $W(x,t)$
    - Linearize the system dynamics around final state $x(t_f)$:
    $\delta\boldsymbol{x}(t+1) = \boldsymbol{A}(t)\delta\boldsymbol{x}(t) + \boldsymbol{B}(t)\delta\boldsymbol{u}(t)$
    $\boldsymbol{A}(t_f) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}|_{\boldsymbol{x}(t_f),\boldsymbol{u}(t_f)}$
    $\boldsymbol{B}(t_f) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}|_{\boldsymbol{x}(t_f),\boldsymbol{u}(t_f)}$
    - Solve LQR Ricatti equation at final state
    $\boldsymbol{P}(t_f) = ricatti(\boldsymbol{Q}_{lqr}, \boldsymbol{R}, \boldsymbol{A}(t_f), \boldsymbol{B}(t_f))$
    - Set final cost of SLQ cost function to Ricatti solution $\boldsymbol{H} = \boldsymbol{P}(t_f)$
    Solve using SLQ
    - Run SLQ algorithm
    - $[\boldsymbol{u}_{ff}(t), \boldsymbol{K}(t), \boldsymbol{x}(t)] = slq.solve(\boldsymbol{f}, \boldsymbol{Q}_{slq}, \boldsymbol{R}, \boldsymbol{H}, \boldsymbol{W}, t_f)$
    - Send control to closed-loop controller
    $control.apply(\boldsymbol{u}_{ff}(t), \boldsymbol{K}(t))$
    - Optional: Record policy lag $t_{lag}$
until finished
```



(a) Ballbot Rezero. *Photo Rezero: Gerhard Born, Ringier AG*

(b) Ascending Technologies Firefly

Figure 3.2: Visualization of both hardware platforms used during the experiments. Both systems are highly dynamic, unstable systems and thus allow for evaluating the performance of the presented approach.

Ballbot

System Description For our first hardware tests, we use the ball balancing robot ("ballbot") Rezero, shown in fig. 3.2. Rezero has very interesting dynamics that render it a suitable platform for the validation of the framework. It is a statically unstable, non-minimum phase system which does not allow for using simple motion planning algorithms. Additionally, the robot is fully torque controlled, allowing for directly optimizing control torques. The controller executing the trajectory runs at 200 Hz on an ARM Cortex M4 while the MPC solver runs on an onboard Intel Core i7 computer.

Model Description A full nonlinear 3D system model of Rezero has been derived analytically [65]. This model describes the robot as two rigid bodies: the ball and the upper body. Furthermore, it assumes that no slip and no rolling friction occurs. Additionally, the actuation dynamics are neglected. The robot's state is defined as

$$x = \begin{bmatrix} \theta_x & \dot{\theta}_x & \theta_y & \dot{\theta}_y & \theta_z & \dot{\theta}_z & \varphi_x & \dot{\varphi}_x & \varphi_y & \dot{\varphi}_y \end{bmatrix}^T \tag{3.6}$$

containing the body angles with respect to gravity (θ_x, θ_y, θ_z) and its derivatives ($\dot{\theta}_x$, $\dot{\theta}_y$, $\dot{\theta}_z$), which represent the angular velocities. Furthermore, the state includes the rotational angles of the ball (φ_x, φ_y) as well as their derivatives ($\dot{\varphi}_x$, $\dot{\varphi}_y$). These states represent the ball's position with respect to a reference position (e.g. start position) and the ball's velocity respectively. We denote the control output, the three motor torques (τ_1, τ_2, τ_3), by u. Through onboard sensors (an inertial measurement unit and motor encoders), full-state feedback is provided. For the full model description see [65].

Hexacopter

System Description The second platform is a Firefly hexacopter by Ascending Technologies, as depicted in fig. 3.2. A Firefly has a Flight Control Unit (FCU) based on an ARM Cortex-M4 micro controller. The optimal control algorithm is computed on an onboard computer with an Intel Core i7 processor, sending velocity commands to the FCU. The current pose estimate obtained from a Vicon tracking system is sent to the computer via a wireless network.

Model Description We are using a widely used dynamic model for Micro Aerial Vehicles (MAVs) as e.g. described in [28, 133, 142]. We assume that the axis of each propeller is parallel to the z–axis of the body frame $\mathbb{B}$ as shown in fig. 3.2. Each propeller i generates a thrust f_i along its axis that is proportional to the square of the propeller's rotational speed n_i. As the motor dynamics are considerably faster compared to those of the vehicle body, they are neglected. The generated thrust and moment from the i-th propeller is given by $f_i = k_n n_i^2$ and $M_i = (-1)^i k_m f_i$, where k_n and k_m are positive constants and M_i is the generated moment. Note that the sign of each moment M_i depends on the rotation direction of the i-th propeller.

Let $\boldsymbol{p}$ be the position and $\mathbf{v}$ the velocity of the center of mass expressed in the inertial frame $\mathbb{W}$, g the acceleration of gravity, T the total thrust generated by the propellers, m the total mass, ω the angular velocity of the vehicle expressed in the body-frame and $\mathbf{J}$ the inertia matrix of the vehicle with respect to the body frame. $\mathbf{R}$ is the rotation matrix from the body to the inertial frame. The system dynamics can be described by the following equations [28, 133]:

$$\dot{\mathbf{p}} = \mathbf{v}, \tag{3.7a}$$

$$\dot{\mathbf{v}} = \mathbf{R}\begin{bmatrix} 0 & 0 & T/m \end{bmatrix}^T + \begin{bmatrix} 0 & 0 & -g \end{bmatrix}^T, \tag{3.7b}$$

$$\dot{\mathbf{R}} = \mathbf{R}\lfloor \boldsymbol{\omega} \times \rfloor, \tag{3.7c}$$

$$\mathbf{J}\dot{\boldsymbol{\omega}} = \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} + \mathcal{A}\begin{bmatrix} f_1 & \dots & f_6 \end{bmatrix}^T, \tag{3.7d}$$

where the operator $\lfloor \times \rfloor$ represents skew symmetric calculation, $\mathcal{A}$ is an appropriate matrix that maps the propellers thrust into moments around each axis of the body frame. For a hexacopter the definition of $\mathcal{A}$ can be found in [1].

I.4 Evaluation and Experiments

In order to assess the performance of our algorithm, we performed different control tasks on both hardware platforms. Furthermore, we studied the runtime of our optimal control solver to quantify its performance.

MPC and Optimal Control Parameters

In our approach, we tried to keep the number of parameters low and made them intuitively tunable. Yet, parameter tuning cannot be avoided, especially given the generality of the approach. Therefore, we briefly describe how to set these parameters and reason about our choices for them.

Cost Function Parameters The final cost matrix H in eq. (3.4) can be tuned to influence the systems approach towards the goal point and the final gains of the trajectory. In the ballbot examples hand-tuned values for H worked well. However, in the hexacopter experiments we are setting H to approximate the infinite time horizon optimal control problem, i.e. the LQR solution centered around the desired goal position. In general, if only diagonal weightings are considered the cost function has about 40 parameters to be tuned. However, considering a pre-tuned LQR controller providing H and R, we only have 24 open parameters. Given the symmetry in the system and not using unintuitive intermediate position gains in Q, we end up with less than 10 general parameters and less than 10 waypoint specific weightings. While the latter are possibly task dependent, we keep these the same throughout all experiments with the exception of an added roll term in section I.4. Since all parameters have an intuitive meaning and the solver is insensitive to them, tuning the costs is simpler than expected.

Time Horizon Choosing the time horizon for the optimal control problem is a common issue. Ideally, one would set the horizon based on the minimum time required to reach the final state without exceeding given state and input constraints. In practice however, many high level planners already present the controller with a desired time horizon. In our experiments, we heuristically set the distance to the p-norm between current and goal position which works well.

Solver Settings In most cases, our optimal control solver converges in one to three iterations. To have some safety margin, we set the maximum number of iterations to five. When updating the controller, the solver can apply a line search scheme. However, the solver did not make use of it in any of the experiments.

Results of Ballbot Experiments

First, we run a set of experiments on the physical hardware of Rezero. Due to the design of feedback controllers, SLQ is able to handle modelling errors on Rezero quite well [69]. However, large disturbances still pose a problem when the control gains (both feedforward and feedback) are not recomputed. To assess to what extend MPC can improve the situation, we are performing tests with go-to tasks where the robot is supposed to reach a desired state in a cost-efficient way. In a first test, the goal state is kept constant, while in a second test it is varied through user inputs. During both tests, the robot will be disturbed significantly by pushing/dragging it far away from its desired state.

Fixed Go-To Task under Disturbance In the first experiment, Rezero is given the task to stay at its initial state which is encoded in the final term of the cost function. During the test we perturb the robot manually to various distances to the initial state. These disturbances would eventually exceed the stability margin of most static feedback controllers (see comparison with LQR in the video[1] attachment), unless they are low-gain and hence show bad tracking performance. In fig. 3.3 an overhead position plot of one return to the initial position after a perturbation is illustrated. The black circles indicate the starting point (after the perturbation has finished) and the final position. The optimized, predicted trajectories are indicated using a gradient that starts with blue (beginning of the trajectory) and ends in red (end of trajectory). These trajectories are obtained by forward simulating a noiseless model of the system dynamics. In green, the actually executed trajectory is shown. As seen in the plot, the MPC controller gradually adjusts the plan to a circular "swing-in" motion. This behavior seems to be favourable in terms of the cost function as it allows a more graceful approach. Due to the limited time horizon, the initial optimization does not converge to this solution and prefers a straight goal approach, but ultimately, the controller converges to a circular balancing trajectory. This trajectory can also be observed in tests with the LQR, which suggests that this behavior emerges from the system dynamics.

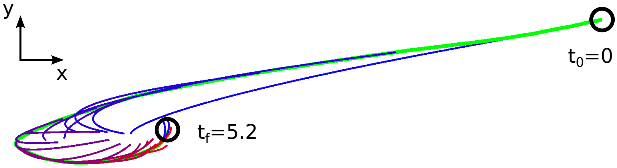


Figure 3.3: Overhead plot of continuously recomputed, predicted MPC paths (blue to red gradient) and the executed path (green) of a go-to task on Rezero. The time horizon is adapted online based on the goal distance.

Varying Go-To Task under Disturbance To show that our approach is also able to handle faster varying changes of the goal state and therefore also the cost function,

---

[1] `https://youtu.be/Y7-1CBqs4x4`

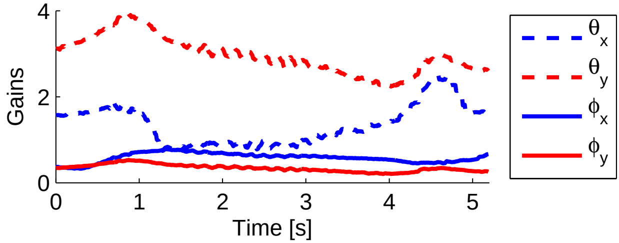


Figure 3.4: Evolution of tilt angle and ball rotation angle feedback gains during a go-to task. The slowly varying gains validate the smooth behavior that the MPC structure generates.

we conduct a test where the goal state is set by an operator through a joystick. The commands given consist of different variations of the goal point including ramp and step inputs. The resulting behavior can best be observed in the video[1] attachment. The robot is able to handle disturbances robustly even under dynamic changes of the cost function. Since the ballbot is a service robot platform, the cost function weights have been tuned for compliant, non-aggressive behavior.

Results of Hexacopter Experiments

Two Window Task On the hexacopter we evaluate the performance of our approach with a dynamic re-planning task. In this task we emulate a wall which has two windows. The left one is horizontal and has a size of 1.0 x 0.6 m, whereas the hexacopter's bounding box (with Vicon markers) is roughly 0.64 x 0.2 m. The window on the right is tilted by 30° on the wall plane. This window is smaller and only spans 0.85 x 0.4 m which prohibits horizontal passes of the hexacopter. We emulate a simple hexacopter level planner that sends two desired intermediate waypoints around the window to be passed, one 0.15 m before and one 0.15 m behind. This way, the controller is prevented from cutting corners. These waypoints are 13 Degrees of Freedom (DoF) waypoints, i.e. amongst time and position, they also include orientation as well as linear and angular velocities. In real world applications, a MAV is subject to quick changes to the environment such as suddenly appearing obstacles, requiring quick changes of plans. We simulate such a scenario by switching between both window waypoint sets unpredictably. Figure 3.5 shows how our control approach solves such task changing requests. The hexacopter first plans a trajectory through the left window. After obtaining the replanning request, the controller lays a trajectory starting at the current state through the waypoints associated to the right, angled window. As benchmarks presented in section I.4 show, this replanning is completed in less than 25 ms, which demonstrates the potential of the presented approach: The high-level planner is only required to plan waypoints instead of a fully dynamically compliant trajectory. The

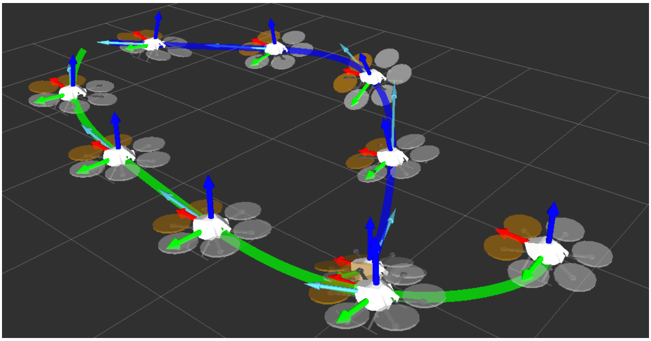


Figure 3.5: At the bottom right the hexacopter is shown in its initial state. First, it plans a trajectory through a waypoint on the left, shown in green. During execution (shortly after the second hexacopter on the green trajectory), the intermediate waypoint is manually switched to the right, and a new trajectory in blue, is optimized through this waypoint.

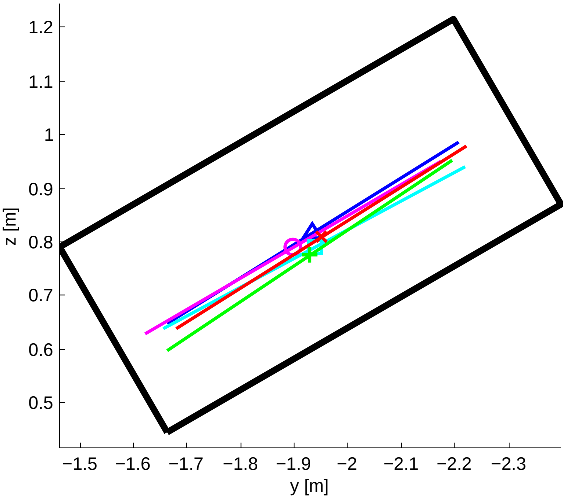


Figure 3.6: Five y- and z-positions of the Center of Gravity with its rolling angles ϕ of a hexacopter flying through a 30°–tilted window. The length of the five lines shows the diameter of the hexacopter, which is 64 cm.

controller itself is given the freedom to plan an optimal, model-based, dynamically consistent trajectory. During flight the MPC controller is faced with different initial conditions. Thus all trajectories have to be optimized online. Passing the window precisely can be regarded as task critical. Therefore, we evaluate the precision of the window passing both in terms of state and time. Figure 3.6 shows the position and orientation of the hexacopter at the desired window waypoint time projected onto the wall plane. We measured the following y and z positions and roll angles ϕ:

$$\begin{aligned}
\boldsymbol{y_{pos}} &= [-1.866, -1.903, -1.863, -1.848, -1.872]\, m \\
\boldsymbol{z_{pos}} &= [0.816, 0.789, 0.790, 0.809, 0.775]\, m \\
\boldsymbol{\phi} &= [-31.79, -30.40, -28.08, -32.23, -33.70]^\circ
\end{aligned}$$

The center of the window is located at $x = 0.0\,\mathrm{m}$ $y = -1.87\,\mathrm{m}$, and $z = 0.83\,\mathrm{m}$, with a roll angle of $-30°$. During the five window passages we measured deviations of the y-position of in the range of $\pm 3\,\mathrm{cm}$, in the z-position we were slightly below the waypoint but never more $6\,\mathrm{cm}$, and the roll-angle ϕ differed less than $\pm 4°$ over the five courses. The two intermediate waypoints were set to different y-values, namely $y = -2.13\,\mathrm{m}$ as we encountered a static offset in preliminary test flights. The z and ϕ values were set to the measured values of the window's elevation and tilting angle. In our experiments we focused on precision, repeatability and consistency of the trajectory execution, even under different initial conditions, and not on absolute accuracy. Hence, we did not incorporate an integral part in the controller which would eliminate the aforementioned offset in y-position.

Optimal Control Solver Benchmark

To verify the speed of the implementation, we conducted a runtime speed test. For this test, we used the hexacopter, as the state and control space are larger than for the ballbot model. The dynamics and derivatives, on the other hand, are slightly more complex for the ballbot system model. Hence, we experienced comparable timings on both systems.

During the window passing experiments, we recorded the times needed to solve the MPC problem summarized in algorithm 2, including both LQR computations for initialization. To reduce measurement noise, we grouped the timings over all approximately 4000 planned trajectories by their time horizon, using bins of 200 ms and plotted their average policy lag in fig. 3.7. Since the complexity of algorithm 1 and algorithm 2 scales linear with time, the policy lag also scales linearly. These timings indicate that the policy lag for trajectories of up to 4 s are generally below 25 ms. The low policy lag demonstrates that state prediction and a better initial guess is not necessary for the presented applications. However, quick tests have shown that e.g. the SLQ frequency, integrator type, or number of iterations could be reduced to further decrease the policy lag. Additionally, in the experiments and benchmarks, we used our single core implementation of the algorithm. The multi core version also available would further speed up the performance by an order of

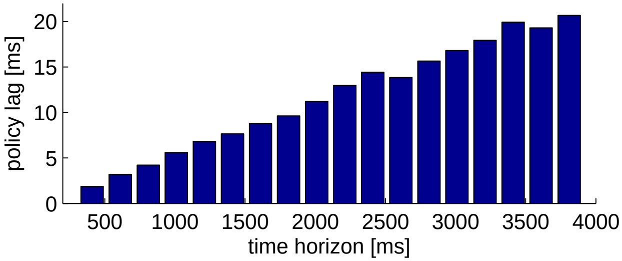


Figure 3.7: Policy lag of approximately 4000 optimized trajectories over time horizons of up to 4 s, during flight experiments with a hexacopter.

50-400 % depending on the CPU model. Thus we assume that SLQ-MPC is also applicable to more complex systems.

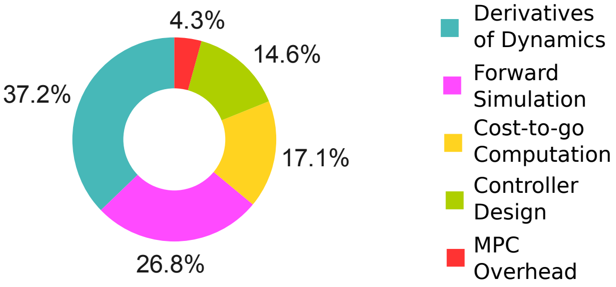


Figure 3.8: Runtime fractions of the MPC framework. The largest fractions are forward dynamics and derivatives (> 60 %) and the iLQG algorithm itself (∼30 %) while the overhead of MPC is low (∼4 %).

With the help of a profiler, we investigated how much time the implementation requires for the individual steps. Our results, illustrated in fig. 3.8, show that most time is spent for the calculation of the forward simulations and derivatives used within the SLQ algorithm. The computation of the cost and the control design update step, which are also part of the SLQ algorithm, account for about 30 % of the runtime. The overhead of the MPC, the state prediction, and data handling remains low with 4 %.

I.5 Conclusion

In this paper we applied SLQ-MPC to solve a nonlinear optimal control problem in a receding horizon fashion for simultaneous trajectory optimization and tracking control. The approach is applied to two different systems, the ballbot Rezero and an AscTec Firefly hexacopter. Hardware experiments demonstrate the capabilities of the approach to plan dynamic trajectories and suited feedback gains for very different tasks. On the ballbot, we are able to design a compliant behavior ideal for crowded environments. On the hexacopter, we demonstrate agile flight maneuvers with accurate tracking and high repeatability. These tasks underline the importance of directly applying the optimized gains to the system and the neccessity for long time horizons. The experiments also exhibit the dynamic replanning capabilities of the approach and display how the algorithm could be combined with a high level planner. Benchmark results show that the MPC problem can be solved in only few milliseconds, even for time horizons of several seconds, indicating that the approach can scale well to more complex systems. In the future we plan to integrate the controller with a high level planner. Also, we will add integral action or model parameter estimation to compensate for steady state errors.

Paper II: “Trajectory Optimization Through Contacts and Automatic Gait Discovery for Quadrupeds”

Authors

Michael Neunert, Farbod Farshidian, Alexander W. Winkler, Jonas Buchli

Abstract

In this work we present a Trajectory Optimization framework for whole-body motion planning through contacts. We demonstrate how the proposed approach can be applied to automatically discover different gaits and dynamic motions on a quadruped robot. In contrast to most previous methods, we do not pre-specify contact-switches, -timings, -points or gait patterns, but they are a direct outcome of the optimization. Furthermore, we optimize over the entire dynamics of the robot, which enables the optimizer to fully leverage the capabilities of the robot. To illustrate the spectrum of achievable motions, we show eight different tasks, which would require very different control structures when solved with state-of-the-art methods. Using our Trajectory Optimization approach, we are solving each task with a simple, high level cost function and without any changes in the control structure. Furthermore, we fully integrate our approach with the robot's control and estimation framework such that we are able to run the optimization online. Through several hardware experiments we show that the optimized trajectories and control inputs can be directly applied to physical systems.

Published in:
Robotics and Automation Letters (RA-L), pp. 1502–1509, 2017

Notes:
Presented at the IEEE International Conference on Robotics and Automation (ICRA)

II.1 Introduction

In motion planning and control, one major challenge is to specify a high level task for a robot without specifying how to solve this task. In legged locomotion such tasks include reaching a goal position or manipulating an object without specifying gaits, contacts, balancing or other behaviors. Trajectory Optimization (TO) recently gained a lot of attention in robotics research since it promises to tackle some of these problems. Ideally, it would solve complex motion planning tasks for robots with many degrees of freedom, leveraging the full dynamics of the system. However, there are two challenges of TO. Firstly, TO problems are hard problems to solve, especially for robots with many degrees of freedom and for robots that make or break contact. Therefore, many approaches add heuristics or pre-specify contact points or sequences. However, this then defines again how the robot is supposed to solve the task, affecting optimality and generality. Secondly, TO cannot be blindly applied to hardware but requires an accurate model as well as a good control and estimation framework. In this work, we are addressing both issues. In our TO framework, we only specify high level tasks, allowing the solver to find the optimal solution to the problem, optimizing over the entire dynamics and automatically discovering the contact sequences and timings. Second, we also demonstrate how such a dynamic task can be generated online. This work does not present a general tracking controller suitable for all trajectories. However, we show a successful integration of TO with our estimation and control framework, allowing the execution of several tasks on hardware even under disturbances.

TO tries to solve a general, time-varying, non-linear optimal control problem. There are various forms of defining and tackling such a control problem. An overview can be found in [16]. With the increase of computational power, TO can be applied to higher dimensional systems like legged robots. Thus, it gained a lot of attention in recent years and impressive results were demonstrated [156, 188, 217]. Yet, these approaches do not present hardware results and [156, 188] do not discuss how to stabilize the trajectory. One of the conceptually closest related work is [218]. However, no gait discovery or very dynamic motions are shown and hardware results are missing. Later work [123] includes hardware results but the planning horizon is short, the motions are quasi-static and contact changes are slow. In [144] TO through contacts is demonstrated on hardware. While the results are promising, the approach is only tested on a single leg platform and for very simple tasks. In general, work that demonstrates TO through contacts applied to physical, legged systems is rare [157], especially on dynamic motions and torque controlled robots. TO with dynamic motions demonstrated on quadruped hardware is presented in [35]. The results of this work are convincing and the authors are also considering actuator dynamics. However, their approach differs in key areas to the presented work: Contact sequences are pre-specified and their approach optimizes over control parameters for a fixed control structure, rather than whole body trajectories and control inputs. Additionally, they are using black box optimization. TO is also used for kinematic [48] or kinodynamic planning [238]. However, these approaches fall short for evaluating stability or generating dynamic motions such as jumping.

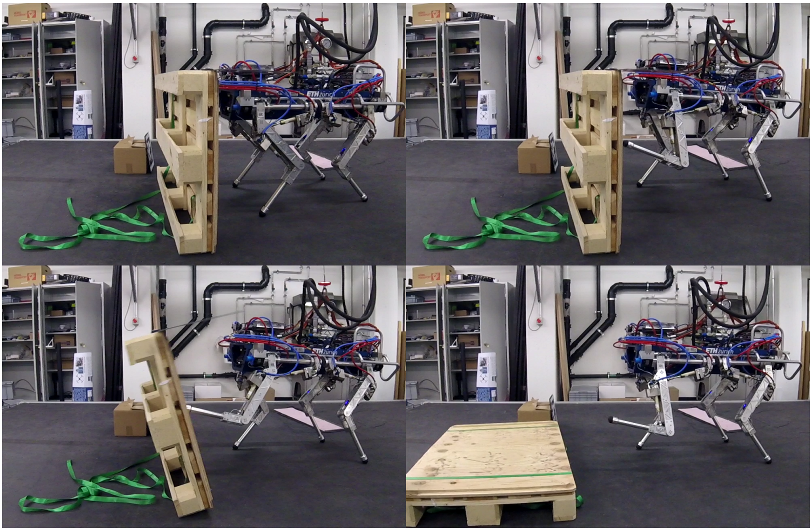


Figure 3.9: Sequence of images during execution of the rough manipulation task in hardware. Time progresses line-wise from top left to bottom right.

Also, not all kinematic plans are dynamically feasible since they may exceed torque limits or friction cones. Often motion planning through contacts is solved using a hierarchy of optimization problems [39, 101, 144, 247]. The issue is that the lower hierarchies need to respect the constraints of the higher hierarchies, often leading to heuristics which impair the optimality of the overall solution.

Contributions

In this work, we apply TO in form of Sequential Linear-Quadratic (SLQ) control [211] to the quadruped robot HyQ [205], both in simulation and on hardware. Our work is one of very few examples, where TO is used to plan dynamic motions through contact changes on a whole-body 3D model and where these trajectories are shown to work on hardware. Compared to previous work on TO through contacts [48, 156, 188, 217], these hardware results help to understand the capabilities and limitations of the approach on physical systems. Furthermore, this is possibly the first time that a gait-free whole-body TO approach is shown on quadruped hardware. In contrast to state-of-the-art approaches on quadrupeds [35], neither contact switching times nor contact events nor contact points are defined a priory. Instead they are a direct outcome of the optimization. This allows us to generate a diverse set of motions and gaits using a single approach. By using an efficient formulation based on Differential Dynamic Programming and our high-performance solver, optimization times for these tasks usually do not exceed a minute, even for complex trajectories, outperforming state-of-the-art approaches [35, 188]. Thus, the TO can be run online and adapt to the given situation. Hence, this work is also one of the earliest examples of whole body TO run alongside a control and estimation pipeline for a legged robotic system. Many TO approaches [48, 156, 188] simply ”play-back“ trajectories for visualization rather than testing them in a control framework. However, only by executing trajectories in the loop with estimators and controllers, we can see physical defects, argue about stability and verify our model assumptions.

II.2 Trajectory Optimization

Optimal Control Problem

In this work, we consider a general non-linear system

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{3.8}$$

where $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ denote state and input trajectories respectively. Our TO approach tries to find the optimal state and control trajectories by solving a finite-horizon optimal control problem which minimizes a given cost function

$$J(\boldsymbol{x}, \boldsymbol{u}) = h\left(\boldsymbol{x}(t_f)\right) + \int_{t=0}^{t_f} l\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right) dt \tag{3.9}$$

To solve this optimal control problem, we apply Sequential Linear Quadratic Control [211], which optimizes a linear, time-varying feedback and feedforward controller of the form $\boldsymbol{u}(\boldsymbol{x}, t) = \boldsymbol{u}_{ff}(t) + \mathcal{K}(t)\boldsymbol{x}(t)$ where $\mathcal{K}(t)$ is the time-varying control gain and $u_{ff}(t)$ the feedforward control action.

Sequential Linear Quadratic Control

Sequential Linear Quadratic Control is an iterative optimal-control algorithm. SLQ first rolls out the system dynamics. Then, the non-linear system dynamics are linearized around the trajectory and a quadratic approximation of the cost function is computed. The resulting Linear-Quadratic Optimal Control problem is solved backwards. Since the solution to the Linear-Quadratic Optimal Control problem can be computed in closed form with Ricatti-like equations, SLQ is very efficient. Algorithm 1 summarizes the algorithm. SLQ computes both, a feedforward control action as well as a time-varying linear-quadratic feedback controller. Handling input constraints in SLQ is possible and we do so by saturating the control input [222]. So far SLQ type algorithms were known to be unable to handle state or state-input constraints. However, recent research conducted after the herein presented work, shows that SLQ is able to handle state and state-input constraints without breaking its linear time complexity by using soft-constraints [166] or by modifying the algorithm [71]. While not yet used in this work, these approaches extend the applicability of SLQ.

Cost Function

While SLQ can handle non-quadratic cost functions, pure quadratic cost functions increase convergence and are often sufficient to describe complex tasks. Therefore, we assume our cost function to be of quadratic form

$$J = \bar{\boldsymbol{x}}(t_f)^T \boldsymbol{H} \bar{\boldsymbol{x}}(t_f) + \int_{t=0}^{t_f} \bar{\boldsymbol{x}}(t)^T \boldsymbol{Q} \bar{\boldsymbol{x}}(t) + \bar{\boldsymbol{u}}(t)^T \boldsymbol{R} \bar{\boldsymbol{u}}(t) \; dt \tag{3.10}$$

where $\bar{x}(t)$ and $\bar{u}(t)$ represent deviations of state and input from a desired state and desired input respectively. These references are fixed, time-invariant setpoints, i.e. there are no state or input trajectories. $\boldsymbol{H}$, $\boldsymbol{Q}$ and $\boldsymbol{R}$ are the weightings for final and intermediate state as well as control input cost respectively. In some instances, we add intermediate state waypoints to the cost function. These temporal costs are weighted by the waypoint cost matrix $\boldsymbol{W}(x, t)$ and defined as

$$W(\boldsymbol{x}, t) = \sum_{n=0}^{N} \hat{\boldsymbol{x}}_n(t)^T \boldsymbol{W}_n \hat{\boldsymbol{x}}_n(t) \sqrt{\frac{\rho_n}{2\pi}} \exp\left(-\frac{\rho_n}{2}(t - t_n)^2\right) \tag{3.11}$$

where ρ_p defines the "temporal spread“ of the cost and t_n defines the time at which the waypoint is active. The waypoint penalizes the deviation from a fixed, time-invariant, desired state $\boldsymbol{x}_{n,d}$ via the difference $\hat{\boldsymbol{x}}_n(t) = \boldsymbol{x}(t) - \boldsymbol{x}_{n,d}$. $\boldsymbol{W}_n$ defines the weighting, which is a purely diagonal matrix. We limit the waypoint usage to two explicit cases: Firstly, when a task cannot possibly be described by a pure quadratic cost function and secondly to increase robustness during hardware experiments. All quadruped gaits shown in Section II.5 are discovered without the use of waypoints.

II.3 System Model and Robot Description

For the experiments in this paper, we use the hydraulically actuated, quadrupedal robot HyQ [205]. Each of the four legs on HyQ has three degrees of freedom, namely hip abduction/adduction (HAA), hip flexion/extension (HFE) and knee flexion/extension (KFE). Each joint is driven by a hydraulic actuator. Joint torques and positions are measured via load cells and encoders respectively. A hydraulic force control loop is closed at joint level [21].

Rigid Body Dynamics

While it is a simplification, torque tracking performance on HyQ is sufficient for modeling the robot as a perfectly torque controlled system. Thus, we assume HyQ

```
Algorithm 1: SLQ Algorithm
Input: System dynamics: $\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right)$
Input: Cost function: $J = h\left(\boldsymbol{x}(t_f)\right) + \int\limits_{t=0}^{t_f} l\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right)$
Input: Initial state and control law: $\boldsymbol{x}(0), \mathbf{u}(\boldsymbol{x}, t)$
repeat
    $\boldsymbol{x}(0 \ldots t_f) \leftarrow \int_0^{t_f} f(\boldsymbol{x}(t), \boldsymbol{u}(\boldsymbol{x}, t))$ // Rollout dynamics
    $\boldsymbol{u}_a(0 \ldots t_f) \leftarrow \boldsymbol{u}(\boldsymbol{x}(0 \ldots t), 0 \ldots t)$ // Record controller
    for $t = 0$ to $t_f - 1$ do
        Linearize the system dynamics
            $\boldsymbol{A}(t) \leftarrow \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}|_{\boldsymbol{x}(t), \boldsymbol{u}(t)}$
            $\boldsymbol{B}(t) \leftarrow \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}|_{\boldsymbol{x}(t), \boldsymbol{u}(t)}$
        Quadratize cost function
            $q(t) \leftarrow J(t)$, $\boldsymbol{q}(t) \leftarrow \frac{\partial \boldsymbol{J}}{\partial \boldsymbol{x}}|_{\boldsymbol{x}(t), \boldsymbol{u}(t)}$, $\boldsymbol{Q}(t) \leftarrow \frac{\partial^2 \boldsymbol{J}}{\partial \boldsymbol{x}^2}|_{\boldsymbol{x}(t), \boldsymbol{u}(t)}$
            $\boldsymbol{P}(t) \leftarrow \frac{\partial^2 \boldsymbol{J}}{\partial \boldsymbol{x} \partial \boldsymbol{u}}|_{\boldsymbol{x}(t), \boldsymbol{u}(t)}$
            $\boldsymbol{r}(t) \leftarrow \frac{\partial \boldsymbol{J}}{\partial \boldsymbol{u}}|_{\boldsymbol{x}(t), \boldsymbol{u}(t)}$, $\boldsymbol{R}(t) \leftarrow \frac{\partial^2 \boldsymbol{J}}{\partial \boldsymbol{u}^2}|_{\boldsymbol{x}(t), \boldsymbol{u}(t)}$
    for $t = t_f - 1$ to $0$ do
        Solve the Riccati-like difference equations:
            $\boldsymbol{P}(t) \leftarrow \boldsymbol{Q}(t) + \boldsymbol{A}^T(t)\boldsymbol{P}(t+1)\boldsymbol{A}(t) +$
                    $\boldsymbol{K}^T(t)\boldsymbol{H}\boldsymbol{K}(t) + \boldsymbol{K}^T(t)\boldsymbol{G} + \boldsymbol{G}^T\boldsymbol{K}(t)$
            $\boldsymbol{p}(t) \leftarrow \boldsymbol{q} + \boldsymbol{A}^T(t)\boldsymbol{p}(t+1) + \boldsymbol{K}^T(t)\boldsymbol{H}\boldsymbol{l}(t) + \boldsymbol{K}^T(t)\boldsymbol{g} + \boldsymbol{G}^T\boldsymbol{l}(t)$
            $\boldsymbol{H} \leftarrow \boldsymbol{R}(t) + \boldsymbol{B}^T(t)\boldsymbol{P}(t+1)\boldsymbol{B}(t)$
            $\boldsymbol{G} \leftarrow \boldsymbol{B}^T(t)\boldsymbol{P}(t+1)\boldsymbol{A}(t)$
            $\boldsymbol{g} \leftarrow \boldsymbol{r}(t) + \boldsymbol{B}^T(t)\boldsymbol{p}(t+1)$
            $\boldsymbol{K}(t) \leftarrow -\boldsymbol{H}^{-1}\boldsymbol{G}$ //feedback update
            $\boldsymbol{l}(t) \leftarrow -\boldsymbol{H}^{-1}\boldsymbol{g}$ //feedforward increment
    $\alpha \leftarrow 1$ // Initialize line search
    repeat
        Line search
        1. Update the control:
            $\mathbf{u}(\boldsymbol{x}, t) \leftarrow \boldsymbol{u}_a(t) + \alpha \boldsymbol{l}(t) + \boldsymbol{K}(t)\boldsymbol{x}(t)$
        2. Forward simulate the system dynamics:
            $\boldsymbol{x}(0 \ldots t_f) \leftarrow \int_0^{t_f} f(\boldsymbol{x}(t), \boldsymbol{u}(\boldsymbol{x}, t))$
        3. Compute new cost:
            $J = h\left(\boldsymbol{x}(t_f)\right) + \int_{t=0}^{t_f} l\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right) dt$
        4. decrease $\alpha$ by a constant $\alpha_d$:
            $\alpha = \alpha / \alpha_d$
    until found lower cost or number of maximum line search steps reached
until maximum number of iterations or converged ($\boldsymbol{l}(t) < l_t$)
```

to behave like a rigid body system defined as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{J}_c^T \lambda(\boldsymbol{q},\dot{\boldsymbol{q}}) + \boldsymbol{S}^T \tau \tag{3.12}$$

where $\boldsymbol{M}$ denotes the inertia matrix, $\boldsymbol{C}$ the centripetal and Coriolis forces and $\boldsymbol{G}$ gravity terms. $\boldsymbol{q}$ is the state vector containing the 6 DoF base state as well as joint positions and velocities. External and contact forces λ act on the system via the contact Jacobian $\boldsymbol{J}_c$. Torques created by the actuation system τ are mapped to the actuated joints via the selection matrix $\boldsymbol{S}$. To formulate our dynamics according to Equation (3.8), we define our state as

$$\boldsymbol{x} = [{}_{\mathcal{W}}\boldsymbol{q} \;\; {}_{\mathcal{L}}\dot{\boldsymbol{q}}]^T = [{}_{\mathcal{W}}\boldsymbol{q}_B \;\; {}_{\mathcal{W}}\boldsymbol{x}_B \;\; \boldsymbol{q}_J \;\; {}_{\mathcal{L}}\omega_B \;\; {}_{\mathcal{L}}\dot{\boldsymbol{x}}_B \;\; \dot{\boldsymbol{q}}_J]^T \tag{3.13}$$

where ${}_{\mathcal{W}}\boldsymbol{q}_B$ and ${}_{\mathcal{W}}\boldsymbol{x}_B$ define base orientation and position respectively, which are expressed in a global inertial "world" frame $\mathcal{W}$. The base orientation is expressed in Euler angles (roll-pitch-yaw). Base angular and linear velocity are denoted as ω_B and $\dot{\boldsymbol{x}}_B$ respectively and both quantities are expressed in a local body frame $\mathcal{L}$. Joint angles and velocities are expressed as $\boldsymbol{q}_J$ and $\dot{\boldsymbol{q}}_J$ respectively. Expressing base pose and twist in different frames allows for more intuitive tuning of the cost function weights. Using Equations (3.12) and (3.13) our system dynamics in Equation (3.8) become

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \begin{bmatrix} {}_{\mathcal{W}}\dot{\boldsymbol{q}} \\ {}_{\mathcal{L}}\ddot{\boldsymbol{q}} \end{bmatrix} \qquad\qquad (3.14) \\ &= \begin{bmatrix} \boldsymbol{R}_{\mathcal{W}\mathcal{L}} \; {}_{\mathcal{L}}\dot{\boldsymbol{q}} \\ \boldsymbol{M}^{-1}(\boldsymbol{q})(\boldsymbol{S}^T\tau + \boldsymbol{J}_c\lambda(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) - \boldsymbol{G}(\boldsymbol{q})) \end{bmatrix} \end{aligned}$$

where $\boldsymbol{R}_{\mathcal{W}\mathcal{L}}$ defines the rotation between the local body frame $\mathcal{L}$ and the inertial "world" frame $\mathcal{W}$. While dropped for readability, $\boldsymbol{R}_{\mathcal{W}\mathcal{L}}$ is a function of ${}_{\mathcal{W}}\boldsymbol{q}$ which needs to be considered during linearization. For SLQ we need to linearize the system given in Equation (3.14). For the derivatives of the upper row with respect to the state $\boldsymbol{x}$ as well as all derivatives with respect to τ we compute analytical derivatives. For the derivatives of the lower row in Equation (3.14) with respect to the state $\boldsymbol{x}$, we use numerical differentiation.

Contact Model

Choosing or designing an appropriate contact model is critical for the performance of TO. For TO it is beneficial to use a smooth contact model which provides good gradients of the dynamics. Yet, a soft model can lead to unphysical effects such as ground penetration or sliding contacts. As a trade-off we are using a non-linear spring-damper contact model extending the model proposed in [211]. We consider two contact models: one collinear and one orthogonal to the surface normal. The orthogonal contact model is defined as

$$\lambda_N = \begin{cases} 0 & p_n \leq 0 \\ (k_n + d_n\dot{p}_n)\frac{p_n^2}{2\alpha_c}\boldsymbol{n}_s & 0 < p_n < \alpha_c \\ (k_n + d_n\dot{p}_n)(p_n - \frac{\alpha_c}{2})\boldsymbol{n}_s & p_n \geq \alpha_c \end{cases} \tag{3.15}$$

Table 3.1: Typical Values for Contact Model Parameters

α_c	k_n	d_n	k_t	d_t
0.01	8000...90000	2000...50000	0...5000000	2000...5000

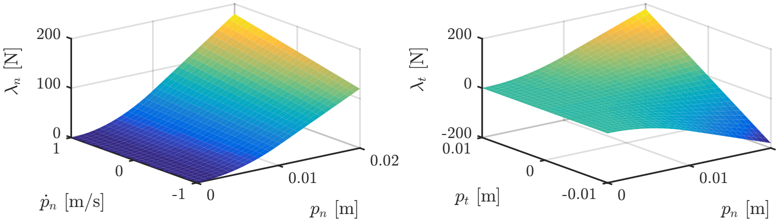


Figure 3.10: Visualization of the contact model. The left plot shows the contact force in surface normal direction as a function of penetration and its derivative. The right plot shows the force in tangential direction as a function of penetration and tangential displacement.

while the tangential model is defined as

$$\lambda_t = \begin{cases} 0 & p_n \leq 0 \\ (k_t p_t \boldsymbol{n}_d + d_t \dot{p}_t \boldsymbol{n}_v)\frac{p_n^2}{2\alpha_c} & 0 < p_n < \alpha_c \\ (k_t p_t \boldsymbol{n}_d + d_n \dot{p}_n \boldsymbol{n}_v)(p_n - \frac{\alpha_c}{2}) & p_n \geq \alpha_c \end{cases} \tag{3.16}$$

Both models include proportional and derivative terms which can be interpreted as springs with stiffnesses k_n, k_t and dampers with damping ratios d_n, d_t respectively. The range of parameter values used for the contact model are given in Table 3.1. For the normal direction the spring displacement p_n is defined as the ground penetration along the surface normal $\boldsymbol{n}_s$. In tangential direction, the offset p_t is computed between the current contact location and the location where the contact has been established. In case of the tangential model, the force vector λ_t is composed of the spring force in tangential displacement direction $\boldsymbol{n}_d$ and a damping force in displacement velocity direction $\boldsymbol{n}_v$. Both, the normal and tangential contact models are visualized in Figure 3.10. To avoid discontinuities during contact changes, the models are non-linear towards zero ground penetration, i.e. for $p_n < \alpha$, where α_c is a smoothing coefficient.

Friction and friction limits are considered by the contact model via friction cones. Thus, they are not included as constraints but are a part of the dynamical system. This allows the TO algorithm to reason about possible slippage and contact force saturation. In this work, we assume that static and dynamic friction coefficients are the same. Therefore, the friction cone can be expressed as a contact force saturation $\boldsymbol{F}_{t,sat} = max(||\boldsymbol{F}_t||, \mu F_n)\boldsymbol{n}_t$, where $\boldsymbol{F}_t$ and F_n are contact forces parallel

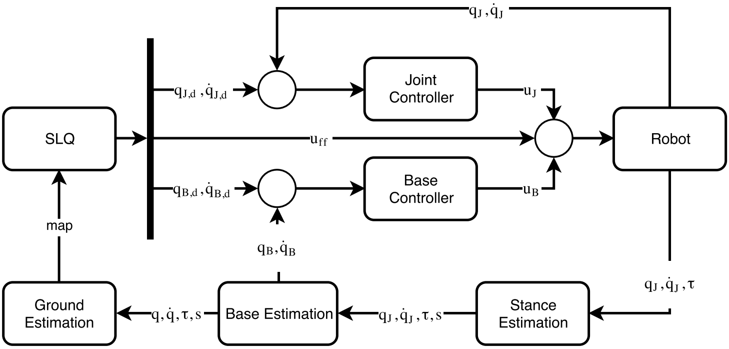


Figure 3.11: Overview of the control and estimation pipeline. Base, stance and ground estimators provide information about the location and contact configuration of the robot. Joint and base controllers stabilize the robot.

and orthogonal to the surface. The normal vector $\boldsymbol{n}_t$ defines the direction of the (unsaturated) tangential contact force $\boldsymbol{F}_t$. The friction coefficient is denoted by μ. Saturating the contact forces results in discontinuities in the overall dynamics. However, due to numerical differentiation, gradient information is recovered and line search helps to mitigate the issue in practice. One could consider replacing the hard limit by a (conservative) soft limit. While the presented contact model works well for obtaining trajectories, there are limitations when it comes to the robustness of the obtained trajectories. While the friction cone is considered, violations of it, i.e. sliding contacts, are not penalized. However, such slippage can cause issues during execution. Additionally, the previously mentioned trade-off between softness for gradients and physical accuracy remains. These and other limitations are discussed in Subsection II.6.

II.4 State Estimation and Tracking Control

Our TO is fully integrated into our estimation and control framework, shown in Figure 3.11. This framework consists of base state and ground estimators, the SLQ solver and tracking controllers. The base estimator and controller run at 250 Hz, while the joint controller operates at 1 kHz.

State Estimation

SLQ assumes full state-feedback. We directly measure joint positions using encoders and compute joint velocities using numerical differentiation. These measurements are fused with IMU data to obtain a base state estimate [19]. While in the TO problem, we assume a soft contact model, our base controller uses discrete contact state information. We obtain the contact state by estimating ground reaction forces from joint torques and compare them to a fixed threshold. We then fit a ground plane to the last contact points of all feet, which provides an estimate of the elevation and contact surface normal. This plane is then also the reference for the soft contact model in the TO. While this is a shortcut to ground estimation or elevation mapping, the presented approach is not limited to co-planar contacts. However, to maintain good gradient information the ground height and normal should be continuous and differentiable.

Tracking Controller

SLQ does not only optimize feed-forward control action but also a feedback controller. While we have successfully applied theses feedback gains to robotic hardware [165], we are not using the optimized feedback gains in this work. The gains are computed for a linearized model and thus significantly depend on the linearization point. During execution, the robot's state will deviate from this nominal point and can thus lie outside of the region of attraction of the controller. Therefore, we use a combination of a joint and base state controller instead. A base controller allows us to directly track the base state and tune feedback gains on these states intuitively. Yet, for swing legs, we still require a joint controller. Additionally, a joint controller on stance legs increases robustness. Hence, we use PD controllers on both, the base and all joint states. The base PD task space controller can be described as

$$\boldsymbol{F}_{cog} = \boldsymbol{P}_x({}_{\mathcal{L}}\boldsymbol{x}^*_{base} - {}_{\mathcal{L}}\boldsymbol{x}_{base}) + \boldsymbol{D}_x({}_{\mathcal{L}}\dot{\boldsymbol{x}}^*_{base} - {}_{\mathcal{L}}\dot{\boldsymbol{x}}_{base}) \tag{3.17}$$

which regulates errors between desired (${}_{\mathcal{L}}\boldsymbol{x}^*_{base}$, ${}_{\mathcal{L}}\dot{\boldsymbol{x}}^*_{base}$) and actual (${}_{\mathcal{L}}\boldsymbol{x}_{base}$, ${}_{\mathcal{L}}\dot{\boldsymbol{x}}_{base}$) base state. The desired body wrench $\boldsymbol{F}_{cog}$ is applied to the robot by converting it to forces at the feet $\boldsymbol{\lambda}_c$ and then mapping them to the joint torques through $\tau_{fb} = \mathbf{J}_c^T \lambda_c$. These torques are then added to the feedforward control action obtained from SLQ. Since it is a model based approach. the SLQ control action already includes torques that counteract gravity and thus, gravity compensation does not need to be added in Equation (3.17).

II.5 Experiments

To validate the approach, we describe different choices of cost functions and show the resulting gaits and motions. For all tasks executed on hardware we use hard input constraints. Not all tasks are executable on hardware for which a thorough discussion is provided in Subsection II.6.

For each task, the cost function is shown as a color code below the heading. The colors indicate the individual, relative weightings of the diagonal entries of each weighting matrix, ranging from lowest (green) to highest (red) on a logarithmic scale. No color means that the respective value is zero and all off-diagonal elements are zero as well. The input cost matrix $\boldsymbol{R}$ is set to identity in all experiments. All tasks are initialized with a simple stance controller and no contact sequence or timings are given. Snapshots of all tasks are shown in Figure 3.12. The optimized

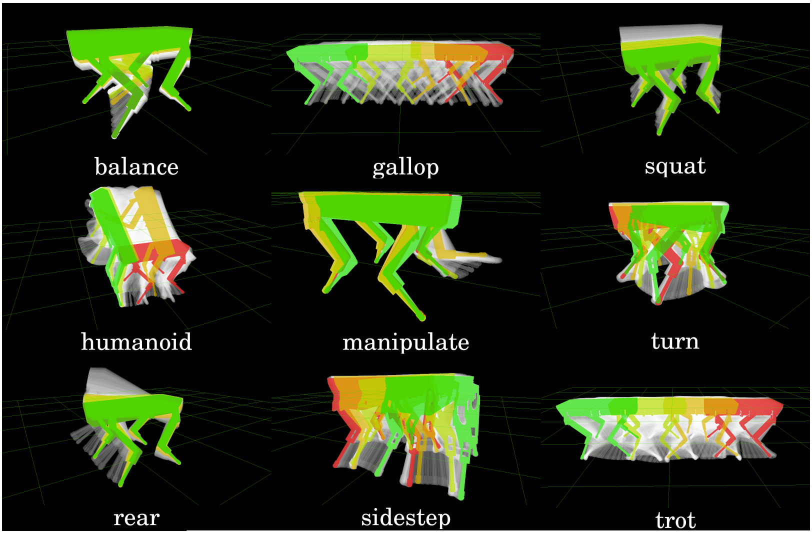
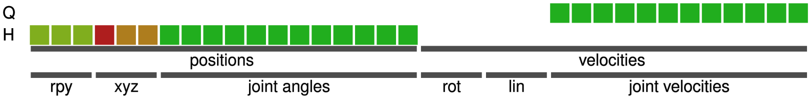


Figure 3.12: Time series of the optimized trajectories for each task. Poses are shown as a color gradient over time ranging from red (initial pose) to green (final pose). Intermediate poses are indicated in transparent white. All displayed motions contain contact switches during dynamic maneuvers. These contact switches result from optimization and are not pre-specified.

trajectory and hardware experiments are shown in the video. All cost functions, solver parameters and the robot model are provided as supplementary material[2].

Galloping



The first gait we demonstrate is galloping. The galloping gait is a direct outcome of setting the final cost terms to penalize the deviation from a desired final pose which is 2 m in front of the robot. Additionally, we add some regularization on the base and leg motion to prevent excessive motions of the body or the limbs. As the results in Figure 3.13 illustrates, we obtain a gallop motion with 9 steps which

---

[2] `https://bitbucket.org/neunertm/hyq_slq_config`

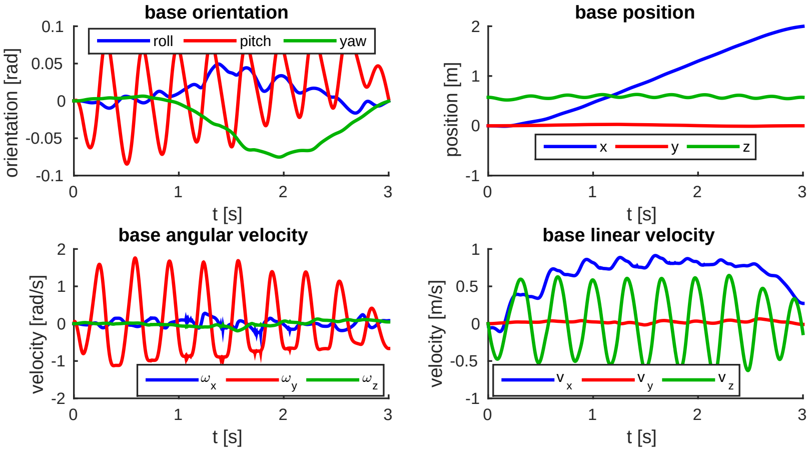
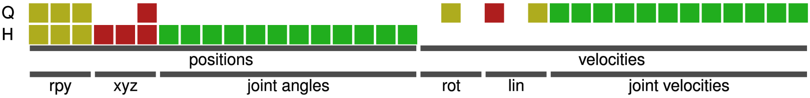


Figure 3.13: Plots of base pose and base twist during galloping as optimized by SLQ. The robot takes in total 9 galloping steps. The desired final position at $x = 2.0$ m is reached with good accuracy.

includes acceleration and deceleration. Finally, the robot reaches a desired position at $x = 2.0$ m. As expected, we see significant pitch motion of the upper body. From Figure 3.14 we can tell that mostly the hind legs are used for acceleration.

Trotting



One hypothesis for the previous task resulting in a galloping behavior is the short time horizon and no penalty on the orientation. If we increase the time horizon to 8 s and penalize the base motion, i.e. we give HyQ more time and encourage smoother base motions, we see that the optimization prefers trotting over galloping. The trot consists of four steps per diagonal leg pair with almost constant stride length. By setting the desired position to the side instead of to the front, the resulting trajectory is a sidestepping motion. If only a desired yaw angle is set, the robot turns on the spot. In all cases the diagonal leg pairs are moved together.

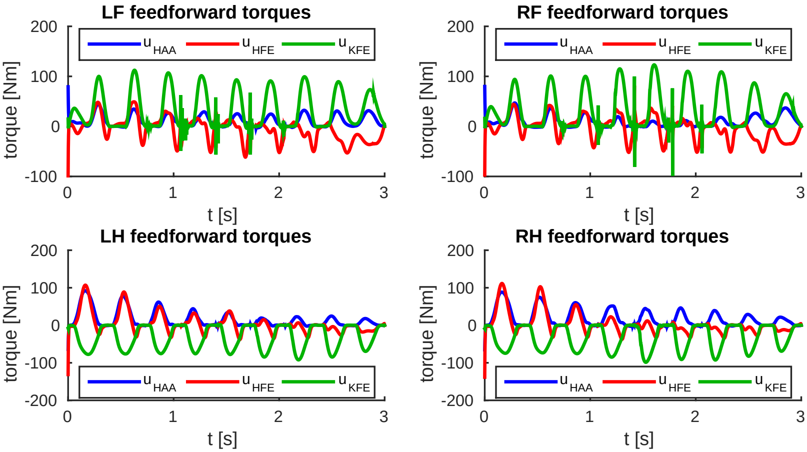
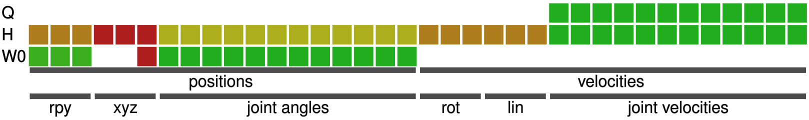


Figure 3.14: Torque profiles of the different joints during a galloping motion. As the lower plots show, the hind legs and especially the HFE joints, contribute greatly to the acceleration. The front leg torques seem to contribute fairly evenly to the galloping motion throughout the trajectory.

Squat Jump



Next, we test if our TO approach can leverage and reason about the dynamics of our system. We do so by adding an intermediate waypoint cost term for the base pose at 0.2 m above the initial base height. Since reaching there by extending the legs exceeds the kinematic limits of the robot, a jump is required. By adding a penalty on the deviation from default joint positions to the waypoint, we encourage a larger ground clearance at the apex. After running our optimization, we obtain a near symmetric squat jump. The overall optimization spans the entire motion, e.g. preparation for lift-off from default pose, the lift-off itself, going to default pose in the air, landing and returning to the default pose. The apex waypoint is localized in time but contact switches and timings are optimized. Figure 3.15 shows the optimized and executed squat jump trajectory. Throughout the task, the base stays level. The desired apex height of 0.2 m is reached with millimeter accuracy (measured 0.192 m). Torques are shown in Figure 3.16 which also illustrate lift-off

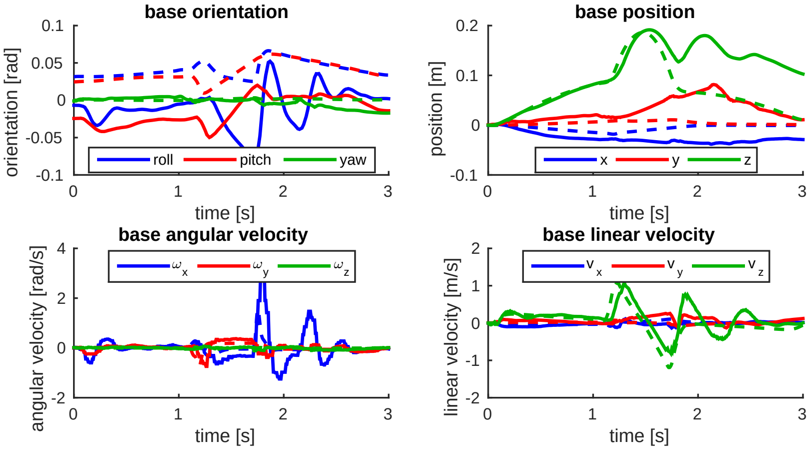


Figure 3.15: Plots of optimized (dashed) and executed (solid) base state during a squat jump on hardware. The robot reaches the desired/planned apex height. Due to insufficient damping, there is a small rebound motion after landing.

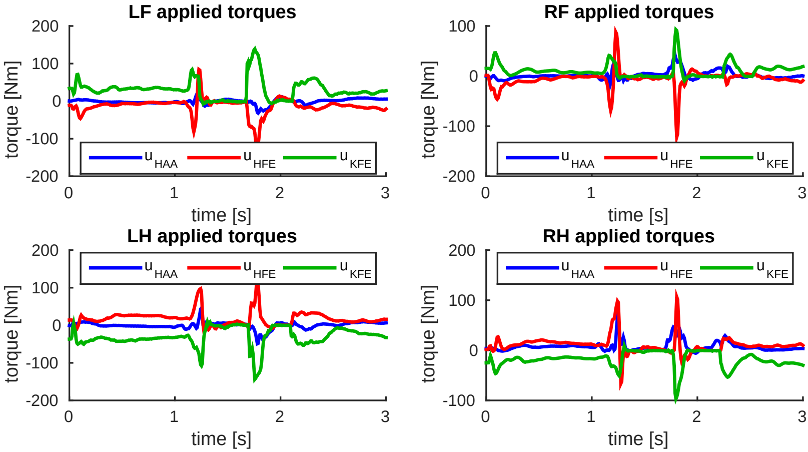
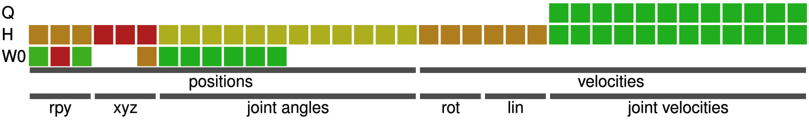


Figure 3.16: Plots of the applied torques during a squat jump on hardware. There are two distinct torque spikes produced during take-off and landing.

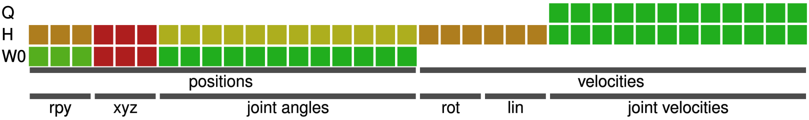
and touch-down times. In the left image of Figure 3.17, we can further see that the robot's legs are in their default position in mid-air as specified in the cost function.

Rearing



For the next task, we are using a similar cost function as for the squat jump. Instead of penalizing the deviation from a neutral base pose, we penalize deviations from a 30° pitched base orientation and we lower the desired apex height to 0.7 m. The final trajectory is a rearing motion, where HyQ lifts off with the front legs, reaches the apex position and finally returns to full contact as well as its default pose.

Diagonal Balance



In order to demonstrate that SLQ can also find statically unstable trajectories, we are demonstrating a diagonal balance task. Here, we use a waypoint term in our cost function again. This term penalizes the orientation and height of the base. Furthermore, it encourages the robot to pull up its legs by bending HFE and KFE of the left front and right hind leg. The final trajectory shows the expected balancing behavior. Again a screenshot at apex is shown in Figure 3.12 and the videoshows the full motion. Interestingly, while we are using a single intermediate term with a single time point and absolutely symmetric costs, the lift-off and touch-down of the swing legs is not synchronous but the front left leg lifts-off later and touches down earlier. This asymmetry most likely stems from the asymmetric location of the Center of Gravity and asymmetric inertia of the robot's main body.

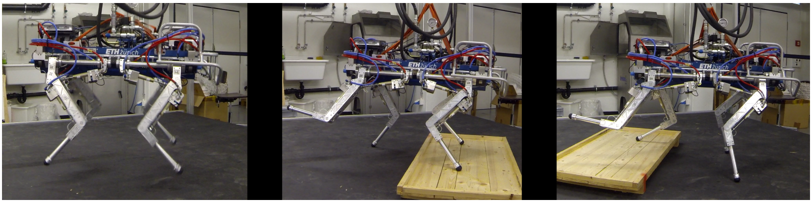
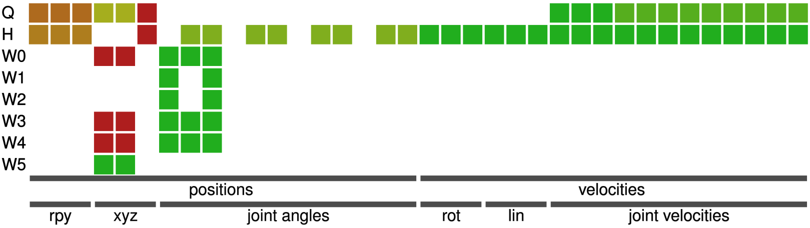


Figure 3.17: Hardware tests for the squat jump (left) and the manipulation under disturbance (center and right). As the algorithm is run online, the motion is optimized to the terrain, e.g. leading to different hind leg joint angles.

Manipulate



As a last task we test a "rough" manipulation motion where HyQ pushes over an obstacle with its front left leg. This task involves submotions such as shifting the robot's CoG in the support polygon of the three stance legs, executing the push motion and shifting the CoG back. While in classic approaches these motions would possibly be all hand coded, they directly result from a single cost function in our TO approach. While the resulting trajectory works perfectly fine in simulation, it lacks robustness on hardware. Since our TO approach is deterministic and we penalize control input, the algorithm tries to minimize the shift of the CoG, leading to a risky trajectory. For visual purposes only, we add additional intermediate waypoints for the front left leg. While we could easily add the pallet push contact to our optimization, we leave it unmodelled on purpose such that it becomes a disturbance to our controller verifying its robustness. Figure 3.12 shows a sequence of images of the optimized motion while Figure 3.9 shows a sequence of images taken during execution. One of the advantages of TO and running it online is its capabilities to adjust to different initial conditions. We vary the initial conditions by placing HyQ's feet over a wooden platform in different configurations as shown in Figure 3.17. All these tests are included in the video.In all motions, CoG shift and leg lift-off overlap in time, i.e. the result is a fluid, dynamic motion. Independent of the initial inclination, the robot levels its base pose during execution. Also, depending on the initial configuration, lift-off timings of the front left leg vary significantly to obtain

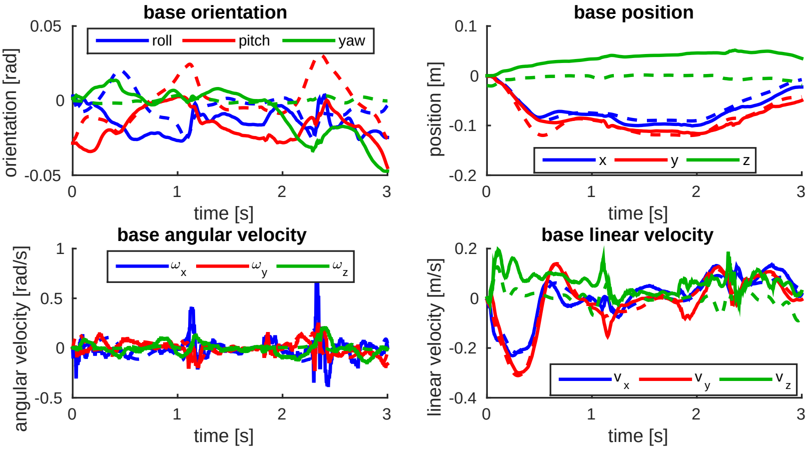
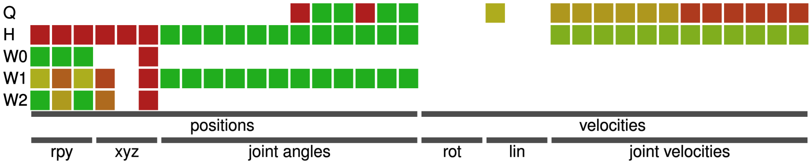


Figure 3.18: Base pose and twist trajectories (solid) and their respective references (dashed) for the rough manipulating task on hardware. The plots show that the planned and executed base trajectories only slightly deviate.

an optimal motion. This underlines the importance of automatically discovering contact timings. For the experiment in Figure 3.9, the base pose/twist tracking is shown in Figure 3.18.

Humanoid Walk



While exceeding the capabilities of our hardware, we evaluate a humanoid walking task where HyQ gets up on its hind legs, moves to a target point in front and then balances there. In contrast to a humanoid with ankles, HyQ only has point feet, increasing under-actuation and the difficulty of the task. The first cost function waypoint, widely spread in time, penalizes deviations in base orientation and height. This ensures that HyQ stays upright during the entire task after getting up. The stand up motion is encouraged by the second waypoint penalizing base orientation, height and changes in forward position. We add a third waypoint one second before the end of the time horizon, defining the target pose and orientation. While the

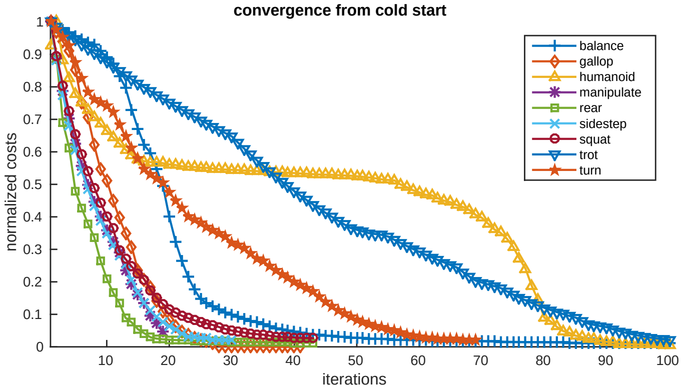


Figure 3.19: Convergence rates for different tasks, which seem to be influenced by the length and complexity of each task. The fastest convergence is observed in the rearing task. Trotting converges the slowest.

last waypoint and the final cost specify the same base pose, we separate them in time to demonstrate that HyQ can stay upright and stabilize in place for a short time. Before getting up HyQ pulls its hind left leg in, moving the contact point closer below the center of gravity. Also, it uses the front left leg (“left arm”) to get up, resulting in a very natural, coordinated, asymmetric motion. After getting into a two-leg standing phase, forward motion is initiated by a short symmetric hopping but quickly changes to a walking pattern. Such non-trivial motions are hard to obtain from fixed timing methods [166].

II.6 Discussion

Runtime and Convergence

When running TO online, runtime and convergence become a major concern since these measures define how long the robot “thinks” before executing a task. Especially in dynamic environments, we want to keep the optimization procedures short. In this section, we will look at both the number of iterations for each task as well as the runtime of each iteration. This gives us an indicator of the complexity of a task and tells us how far we are from running our approach in Model Predictive Control (MPC) fashion.

First, we measure convergence rates. To obtain comparable results, we initialize all tasks with a stance controller and normalize the costs with the initial cost of each task. The results are shown in Figure 3.19. The curves suggest that there is a relation between the complexity of a task and its convergence rate. The trotting is a

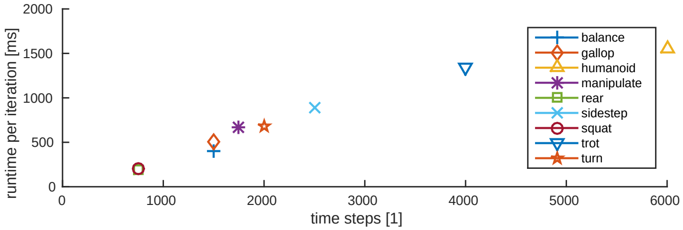


Figure 3.20: Runtime per iteration for different tasks. From a theoretical point of view, the relation between number of time steps and the runtime per iteration should be linear. This plot nicely supports this hypothesis.

complex behavior with a long time horizon which might explain slow convergence. In contrast, the rearing task is relatively simple and converges quickly. When it comes to runtime, SLQ has a major advantage over other TO approaches: The complexity scales linearly with discretization steps and thus with time horizon. Figure 3.20 shows the runtime as a function of time steps, underlining this linear relation. The timings are measured on a quadcore laptop computer and averaged over all iterations for each task as indicated in Figure 3.19. To achieve these runtimes, we use a custom multi-threaded solver and optimized code for the system dynamics computation generated by RobCoGen [79]. Given the runtimes and the required iterations, we achieve convergence times of less than a minute for most tasks.

Robustness and Model Accuracy

There are various reasons why some of the motions and especially the gaits cannot yet be executed on hardware. First, some tasks, such as the humanoid walking, exceed the physical limitations of the system. But since they illustrate the capabilities of the approach we still include them. Second, our TO is a deterministic approach without a notion of robustness. In some tasks robustness can be increased by cost function tuning. However, it remains an issue for all deterministic TO approaches since such methods cannot determine how sensitive a trajectory is to disturbance or model errors. E.g. the found trotting behavior would be more robust with an increased stepping frequency. Third, some tasks show slippage behavior leading to risky behaviors, potentially resulting in accidents. This slippage can have two sources: the friction cone and contact model softness. In the current form, there is no penalty for exceeding the friction cone. Therefore, sliding is not penalized or avoided, even when setting conservative limits. Additionally, the inherent trade-off in the contact model parameters between good gradients and physicality remains. Lastly, our control framework cannot modify step timings or locations to stabilize gaits efficiently. While heuristics like capture points methods exists, we think that running the approach in receding horizon fashion could address this issue in a more principled way.

II.7 Conclusion and Outlook

We have presented a fully dynamic, whole-body TO framework able to create motions which involve multiple contact changes. The approach does not require any priors or initial guesses on contact points, sequences or timings. We demonstrate the capabilities on various tasks including squat jumps, rearing, balancing and rough manipulation. Furthermore, our TO is able to discover gaits such as galloping, trotting and two legged walking. Hardware results show that optimized trajectories can be transferred to physical systems. Despite the versatility of our approach, we obtain an optimized trajectory in less than one minute without warm starting. However, there are also some shortcomings. The solution space is huge, i.e. we can apply our TO to a broad variety of tasks. While this generality is a strength, it also requires to “choose” a solution by modifying cost function weights or adding additional terms. While this is a more or less intuitive approach, one would wish to further reduce the number of open parameters. Another drawback is that we can currently not transfer some motions onto hardware. We believe that running the approach as a model predictive controller will mitigate some of these issues. The presented timings suggest that - when warm starting and using shorter time horizons - SLQ is fast enough to be run with a receding horizon, even for complex systems such as HyQ. In previous work [165], we have already shown such an approach for stabilization, rapid replanning and disturbance rejection.

Paper III: “Efficient Whole-Body Trajectory Optimization Using Contact Constraint Relaxation”

Authors

Michael Neunert, Farbod Farshidian, Jonas Buchli

Abstract

In this work we present a Trajectory Optimization framework for whole-body motion planning for floating base robots. We demonstrate how the proposed approach can optimize whole body trajectories for most common gaits found in bipeds and quadrupeds. The underlying optimal control problem is solved efficiently using Sequential Linear Quadratic control. In contrast to most previous methods, the proposed approach is fast while still using a full dynamic model. Additionally, the approach is contact model free. Instead contact forces are added to the Optimal Control formulation as additional control inputs and contact constraints are handled using a relaxation approach. In our experiments we demonstrate that, despite this relaxation, our solver resolves constraint violations in only a few iterations. Hardware experiments underline the transferability from simulation to hardware.

Published in:
Proceedings of the IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS), pp. 43–48, 2016

III.1 Introduction

Trajectory Optimization (TO) is a highly popular approach for solving complex, dynamic motion planning problems in robotics. It promises to replace manually designed planning and control frameworks. In TO non-linear optimization or Optimal Control are used to design a trajectory that minimizes a high level performance criterion, e.g. a cost function. Also in legged robotics, TO is gaining a lot of attention. However, due to the high number of degrees of freedom of modern legged robots and their need for establishing and breaking contacts with the environment, these problems become high-dimensional and non-convex. Thus, different approaches, both for the formulation of the problem as well as for solving it, have been proposed.

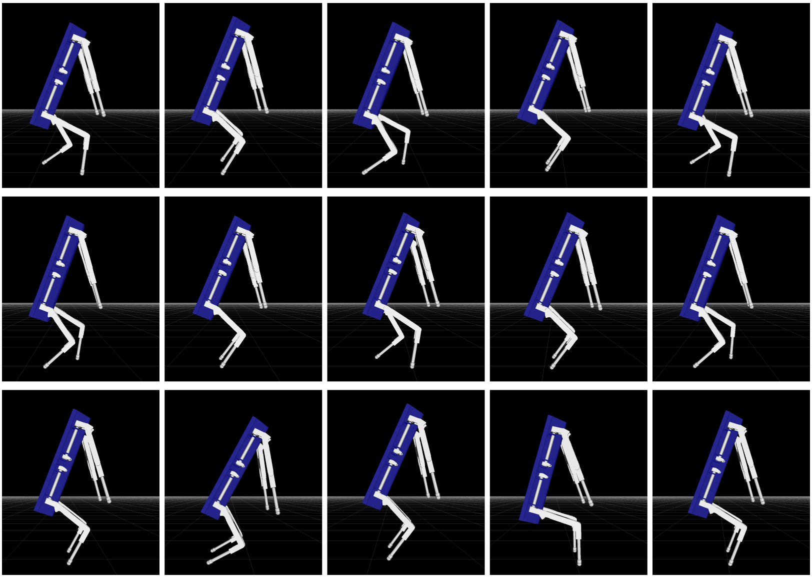


Figure 3.21: Time series of the optimized trajectories for the humanoid gaits. The picture series progresses from left to right and the robot is moving to the right. From top to bottom the gaits are: walk, run and hop.

Related Work

Coros et al. [247] propose an optimization approach on a hierarchy of simplified models, including an inverted pendulum model, to solve whole-body motion planning tasks on a humanoid robot. Their approach shows impressive results and also optimizes contact locations and timings. However, it also relies on heuristics and optimality cannot be guaranteed. Another gait free approach that relies on an inverted pendulum model is presented by Mordatch et al. [155]. A different way of reducing the model complexity is using centroidal dynamics, e.g. as used in [46] and [126], leading to great hardware results. Another approach using a simplified model without pre-specifying contact sequences is proposed by Mordatch et al. in [156]. However, the proposed method additionally relaxes the entire optimization problem and only enforces dynamic and kinematic consistency as soft constraints, leading to severe violations of these quantities. A whole-body trajectory optimization approach is presented by Posa et al. [188]. In this work, the authors apply direct transcription and the resulting non-linear programming is solved using Sequential Quadratic Programming (SQP). While not assuming a given contact sequence, the approach is only demonstrated on lower dimensional, planar systems and might not scale to robots with many degrees of freedom. Other direct transcription approaches using SQP are presented in [189] and [181]. Both approaches follow the idea of projecting their decision variables to the tangent of the constraint manifold. In both cases, results are presented for full models of complex robots, i.e. a humanoid and a quadruped respectively. Therefore, solving the resulting non-linear program is computationally very costly. The same drawback also applies to the Multiple shooting approach in [122] applied to the humanoid robot HRP-2. An even more complex robot model that includes actuator dynamics is used by Gehring et al. [35]. Also here, the high model dimensionality leads to long computation times. Recently, researchers have tried to tackle Trajectory Optimization using Optimal Control algorithms. Popular for their efficiency are Differential Dynamic Programming (DDP) [147] and closely related algorithms such as iterative Linear Quadratic Gaussian (iLQG) [222] and Sequential Linear Quadratic (SLQ) [211] control. Impressive results computed close to real-time have been obtained from such solvers [217]. In recent work [168] we have demonstrated that such an approach can also automatically discover gaits and the resulting trajectories can be applied to hardware. However, in such methods there is a trade-off to be made between using a physically accurate, stiff contact model or softening it for better gradients and faster convergence. Additionally, these approaches require very careful cost function engineering to obtain reasonable solutions. To overcome the limitations of handling constraints in such methods, we have derived a continuous–time SLQ variant which allows for including state–input constraints as hard limits [71]. Furthermore, this work provides an algorithmic foundation for gradually enforcing state constraints. While we have applied this approach to a centroidal model of a quadruped in simulation, we have not yet tested it on a full multi–rigid body model or on hardware.

Contribution

In this work, we propose how to efficiently solve a TO problem over contact switches using SLQ. We present a contact constraint relaxation scheme by including contact forces as control input variables and gradually enforcing contact constraints via the cost function. While such approaches are said to have poor numeric properties and might lead to constraint violation, we demonstrate how our method reliably optimizes constraint satisfactory trajectories. We illustrate this for common gait types of bipeds and a quadrupeds. Due to the relaxation and an efficient Sequential Linear Quadratic control solver, the non-linear TO problem is solved in a few seconds for a full non-linear dynamic model. This outperforms many state-of-the-art TO approaches. By using a single quadratic cost across different gaits, we show that the proposed formulation does not require extensive tuning. While gait patterns need to be pre-specified, we still optimize for contact locations without relying on heuristics or model reductions. Lastly, this work is a rare example where optimized gaits obtained from whole-body TO are directly applied to hardware.

III.2 Trajectory Optimization

Optimal Control Problem

In our Trajectory Optimization problem, we assume general non-linear system dynamics

$$\boldsymbol{x}(t+1) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{3.18}$$

where $x(t)$ and $u(t)$ represent state and control input respectively. For this system, we are trying to find a time-varying feedforward and feedback controller of the following form

$$\boldsymbol{u}(\boldsymbol{x}, t) = \boldsymbol{u}_{ff}(t) + \mathcal{K}(t)\boldsymbol{x} \tag{3.19}$$

where $u_{ff}(t)$ denotes a time-varying feedforward action and $\mathcal{K}(t)$ a time-varying feedback controller. Given the system dynamics, we obtain such a controller by solving the finite time-horizon optimal control problem

$$J(\boldsymbol{x}, \boldsymbol{u}) = \Phi\left(\boldsymbol{x}(t_f)\right) + \sum_{t=0}^{t_f-1} L\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right) \tag{3.20}$$

where $\Phi(\cdot)$ describes the final evaluated at the end of the time horizon t_f and $L(\cdot)$ describes the intermediate cost.

Sequential Linear Quadratic Control

In this work, we are solving the non-linear Optimal Control problem stated above by applying Sequential Linear Quadratic (SLQ) control [211]. Given an initial guess, SLQ forward simulates the non-linear system dynamics. Afterwards, a linear approximation of the dynamics and a quadratic approximation of the cost function is computed around the obtained trajectory. This results in a series of Linear Quadratic control problems which are then solved with a backward sweep using Ricatti-like equations. This backwards sweep results in a feedforward control

Algorithm 3 SLQ Algorithm

```
Given
- System dynamics as in Eq. 3.18, Cost function as in Eq. 3.21
- Initial stable control law, $\mathbf{u}(\boldsymbol{x}, t)$
repeat
    - Forward simulate the system dynamics
    - Linearize the system dynamics along the trajectory:
    $\delta\boldsymbol{x}(t+1) = \boldsymbol{A}(t)\delta\boldsymbol{x}(t) + \boldsymbol{B}(t)\delta\boldsymbol{u}(t)$
    - Quadratize cost function along the trajectory $\boldsymbol{\tau}$:
    $\tilde{J} \approx p(t) + \delta\boldsymbol{x}^\top(t_f)\boldsymbol{p}(t_f) + \frac{1}{2}\delta\boldsymbol{x}^\top(t_f)\boldsymbol{P}(t_f)\delta\boldsymbol{x}(t_f)$
        $+ \sum_{t=0}^{t_f-1} q(t) + \delta\boldsymbol{x}^\top\boldsymbol{q}(t) + \delta\boldsymbol{u}^\top\boldsymbol{r}(t)$
        $+\frac{1}{2}\delta\boldsymbol{x}^\top\boldsymbol{Q}(t)\delta\boldsymbol{x} + \frac{1}{2}\delta\boldsymbol{u}^\top\boldsymbol{R}(t)\delta\boldsymbol{u}$
    - Backwards solve the Riccati-like difference equations:
        $\boldsymbol{P}(t) = \boldsymbol{Q}(t) + \boldsymbol{A}^\top(t)\boldsymbol{P}(t+1)\boldsymbol{A}(t) +$
                $\boldsymbol{K}^\top(t)\boldsymbol{H}\boldsymbol{K}(t) + \boldsymbol{K}^\top(t)\boldsymbol{G} + \boldsymbol{G}^\top\boldsymbol{K}(t)$
        $\boldsymbol{p}(t) = \boldsymbol{q} + \boldsymbol{A}^\top(t)\boldsymbol{p}(t+1) + \boldsymbol{K}^\top(t)\boldsymbol{H}\boldsymbol{l}(t) + \boldsymbol{K}^\top(t)\boldsymbol{g} + \boldsymbol{G}^\top\boldsymbol{l}(t)$
        $\boldsymbol{H} = \boldsymbol{R}(t) + \boldsymbol{B}^\top(t)\boldsymbol{P}(t+1)\boldsymbol{B}(t)$
        $\boldsymbol{G} = \boldsymbol{B}^\top(t)\boldsymbol{P}(t+1)\boldsymbol{A}(t)$,      $\boldsymbol{g} = \boldsymbol{r}(t) + \boldsymbol{B}^\top(t)\boldsymbol{p}(t+1)$
        $\boldsymbol{K}(t) = -\boldsymbol{H}^{-1}\boldsymbol{G}$,      $\boldsymbol{l}(t) = -\boldsymbol{H}^{-1}\boldsymbol{g}$
    - Update the control:
    $\mathbf{u}(\boldsymbol{x}, t) = \boldsymbol{u}_n(t) + \alpha\boldsymbol{l}(t) + \boldsymbol{K}(t)\left(\boldsymbol{x}(t) - \boldsymbol{x}_n(t)\right)$
    - Line search over $\alpha$
    - Increase constraint penalty: $\gamma \leftarrow \gamma_0\,\gamma$
until maximum number of iterations or converged ($\boldsymbol{l}(t) < l_t$)
```

update and a new feedback controller. Solving Ricatti-like equations makes SLQ very efficient and results in linear complexity with respect to the time horizon. One drawback of SLQ is that it cannot handle hard state constraints. However, using a constraint relaxation described in Subsection III.2 they can be included as soft constraints in the cost function. Our soft-constrained SLQ version is summarized in Algorithm 3. As we will show later, despite using such soft constraints, the algorithm still finds constraint satisfactory trajectories. While handling hard input constraints in SLQ-type algorithms is possible, as shown in [71] and [218], we apply constraint relaxation on input constraints as well to promote convergence speed.

Cost Function and Constraint Relaxation

In this work, we assume a quadratic cost function with an added constraint violation penalty

$$\begin{aligned} J =& \bar{\boldsymbol{x}}(t_f)^\top\boldsymbol{Q}_f\bar{\boldsymbol{x}}(t_f) + \sum\nolimits_{t=0}^{t_f-1}\boldsymbol{x}(t)^\top\boldsymbol{Q}\boldsymbol{x}(t) + \boldsymbol{u}(t)^\top\boldsymbol{R}\boldsymbol{u}(t) \\ &+ \gamma_{(i)}\sum\nolimits_{t=0}^{t_f} C_c(\boldsymbol{x}, \boldsymbol{u}, t), \end{aligned} \tag{3.21}$$

where $\bar{\boldsymbol{x}}(t_f) = \boldsymbol{x}_f(t_f) - \boldsymbol{x}_{f,d}$ is the state deviation from desired final state. $\boldsymbol{Q}_f$, $\boldsymbol{Q}$ and $\boldsymbol{R}$ denote weighting matrices for final cost, intermediate cost and input cost respectively. The quadratic elements of the cost function encode the task of the robot and are able to regularize states if required. Additionally, we add a constraint violation cost term $C_c(\boldsymbol{x}, \boldsymbol{u}, t)$ which allows for handling both state and input constraints using relaxation. The constraints considered in this work and the resulting soft constraint function are described in Subsection III.3. In early iterations of the algorithm, constraints are relaxed to allow the solver to find a good initial guess that fulfills the task but may violate constraints. Over the course of the iterations, the constraints are then enforced by increasing the constraint violation cost gain $\gamma_{(i)}$ after each iteration. We increase the gain exponentially: $\gamma_{(i)} = \gamma_0^i$, where i denotes the iteration number and γ_0 a user defined value. In our examples, we choose $\gamma_0 = 3$ as an initial gain.

III.3 System Model and Contact Constraints

To verify our approach, we use our hydraulically actuated, quadrupedal robot HyQ [205], modelled using rigid body dynamics. This four legged robot has three actuated joints per leg which are all torque controlled.

Rigid Body Dynamics

Due to a high performance onboard joint torque controller, we assume HyQ to be a rigid body system with perfect torque inputs, described as

$$\dot{\boldsymbol{q}}(t+1) = \boldsymbol{M}^{-1}\left(\boldsymbol{J}_c^\top \boldsymbol{\lambda} + \boldsymbol{S}^\top \boldsymbol{\tau} - \boldsymbol{h}\right)\Delta t \tag{3.22}$$

where $\boldsymbol{M}$ is the joint space inertia matrix, $\boldsymbol{h}$ are the Coriolis, centrifugal and gravity terms and $\boldsymbol{q}$ is the state vector containing base and joint state. Contact forces $\boldsymbol{\lambda}$ act on the system via the Jacobian $\boldsymbol{J}_c$, whereas actuator forces $\boldsymbol{\tau}$ get mapped to the system by the input selection matrix $\boldsymbol{S}$.

In order to satisfy our formulation of a non-linear system in Equation 3.18, we define our state in the optimization as

$$\boldsymbol{x} = [{}_{\mathcal{W}}\boldsymbol{q} \; {}_{\mathcal{L}}\dot{\boldsymbol{q}}]^\top = [{}_{\mathcal{W}}\boldsymbol{q}_B \; {}_{\mathcal{W}}\boldsymbol{x}_B \; \boldsymbol{q}_J \; {}_{\mathcal{L}}\omega_B \; {}_{\mathcal{L}}\dot{\boldsymbol{x}}_B \; \dot{\boldsymbol{q}}_J]^\top \tag{3.23}$$

where ${}_{\mathcal{W}}\boldsymbol{q}_B$ and ${}_{\mathcal{W}}\boldsymbol{x}_B$ represent the pose of the trunk expressed in a global inertial "world" frame $\mathcal{W}$. We use Euler angles (roll-pitch-yaw) to represent the base orientation. ω_B and $\dot{\boldsymbol{x}_B}$ jointly represent the base twist, i.e. angular and linear velocity, expressed in a local body frame $\mathcal{L}$. $\boldsymbol{q}_J$ and $\dot{\boldsymbol{q}}_J$ denote joint angles and velocities respectively. Combining Equations 3.18, 3.22 and 3.23 our system dynamics can be expressed as

$$\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + \begin{bmatrix} \boldsymbol{R}_{\mathcal{WL}} \; {}_{\mathcal{L}}\dot{\boldsymbol{q}}(t) \\ \boldsymbol{M}^{-1}(\boldsymbol{S}^\top \boldsymbol{\tau} + \boldsymbol{J}_c^\top \boldsymbol{\lambda} - \boldsymbol{h}) \end{bmatrix} \Delta t \tag{3.24}$$

where $\boldsymbol{R}_{\mathcal{WL}}$ rotates a vector expressed in the local body frame $\mathcal{L}$ to the inertial "world" frame $\mathcal{W}$.

Implicit Contact Model

If we were to use an explicit contact model, the contact force vector $\boldsymbol{\lambda}$ in Equations 3.22 and 3.24 would become a function of the rigid body position $\boldsymbol{q}$ and velocity $\dot{\boldsymbol{q}}$ vectors. The issue with such a contact model is that it needs to be stiff to reflect reality but still sufficiently soft to help convergence of any optimal control or optimization problem. In order to avoid this conflict, we do not define an explicit contact model. Instead, we add the contact force vector $\boldsymbol{\lambda}$ to our control input vector $\boldsymbol{u}(t) = [\boldsymbol{\tau}\ \boldsymbol{\lambda}]^\top$. Furthermore, the contact constraints (see Subsection III.3) are added to the problem to ensure physicality.

Using this implicit contact model has both benefits and drawbacks. One major benefit is that it allows the Optimal Control solver to directly influence the contact forces which relaxes the otherwise very stiff problem. This allows for longer integration steps and faster convergence rates. However, it also makes the Optimal Control problem higher dimensional. This issue is partially mitigated since we do not require to compute a contact model. State-of-the-art simulators often solve Linear Complementary Problems (LCP) in their contact models which require iterative solvers. Here, we combine the contact model solving and trajectory optimization in a single Optimal Control problem.

One might be inclined to use an explicit contact model and increase its stiffness over the course of iterations in order to get the best of both worlds. However, such an approach is difficult to apply to an SLQ-type approach. In each iteration, SLQ relies on a rollout of the dynamics given a controller obtained in the last iteration. If we increase the contact model stiffness between iterations, the rollout in the following iteration might be unstable and will not provide good linearization points for the control update.

Contact Constraints

We assume a rigid body system is in contact with its environment if the distance between the system and the environment is zero, i.e. there is no gap between both but also penetration is inadmissible. Only when in contact, external forces can act between the system and the environment. These two requirements describe a rigid contact model which can be expressed by a single complementary constraint

$$d_{ee}(\boldsymbol{q})\boldsymbol{\lambda}_{ee} = 0 \tag{3.25}$$

where $\boldsymbol{\lambda}_{ee}$ is the contact force acting at a specific end-effector and $d_{ee}(\boldsymbol{q})$ is the distance of the end-effector to the environment as a function of the rigid body state vector $\boldsymbol{q}$ introduced in Equation 3.22. While this is a very general formulation, it can make the TO problem highly non-convex. To mitigate this issue, we assume a given contact sequence, i.e. a binary contact state $c_{ee}(t)$ for each end-effector. This contact state function evaluates to 1 if an end-effector is in contact and 0 if it is not in contact. This allows us to split the complementary constraint into its two original parts, a distance constraint and a force constraint

$$c_{ee}(t)d_{ee}(\boldsymbol{q}) = 0, \quad (1 - c_{ee}(t))\boldsymbol{\lambda}_{ee} = 0. \tag{3.26}$$

Furthermore, under the assumption that no slippage occurs, the point $\boldsymbol{p}_{ee}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ on a rigid body system which is in contact with the environment is constrained to have zero velocity

$$c_{ee}(t)\dot{\boldsymbol{p}}_{ee}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = 0. \tag{3.27}$$

This constraint separation converts the state-input complementary constraint (Eq. 3.25) into pure state and input constraints (Eq 3.26). Given our constraint relaxation formulation introduced in Subsection III.2, we define the constraint violation term of our cost function (Eq. 3.21) as the sum of squared violations over all end-effectors

$$\begin{aligned} \boldsymbol{C}_c(\boldsymbol{x}, \boldsymbol{u}, t) = \sum\nolimits_{ee=0}^{N_{ee}} (1 - c_{ee}(t))(\beta_3 \boldsymbol{\lambda}_{ee}^{\top} \boldsymbol{\lambda}_{ee} + \beta_4 d_{s,ee}^2(t)) \\ + c_{ee}(t)(\beta_1 \dot{\boldsymbol{p}}_{ee}^{\top} \dot{\boldsymbol{p}}_{ee} + \beta_2 d_{ee}^2) \end{aligned} \tag{3.28}$$

where N_{ee} is the number of end-effectors and $\beta_{1...4}$ are constant weighting factors. Additionally to the constraint, we add a desired swing leg height $d_{s,ee}$ to the cost function.

III.4 Experiments

To evaluate the proposed approach, we optimize trajectories for the common dynamic gait patterns found in bipedal and quadrupedal locomotion. In both cases, we use the model of our robot HyQ described in Section III.3. While by design, HyQ is a quadruped, we use the same model for optimizing humanoid gaits. While in reality, the robot is not strong enough for bipedal locomotion, it allows us to compare bipedal and quadrupedal locomotion. For each of the two systems, we only use one single cost function with fixed, purely diagonal weighting matrices to optimize all gaits. Furthermore, we demonstrate the transferability of the optimized motion on several trotting tasks executed on hardware. The cost function weights used for hardware experiments are the same as in simulation. To set the goal for the robot to reach, we specify the desired final body pose in the cost. We cold start all tasks with zero control input resulting in a vertical fall. The SLQ and integration step size is et to $\Delta t = 2ms$. The obtained results are summarized below and shown in the video attachment.

Humanoid

First, we demonstrate humanoid gaits in simulation. We rely on the three most common gaits also found in humans, namely walking, running and hopping. Picture series of all three optimized gaits are shown in Figure 3.21.

Walk First we show a humanoid walking gait with 50% contact duty cycle and without flight or double support phase. While the contact sequence is pre-specified, the contact locations are not. This allows the optimizer to find a trajectory where the robot first steps back with its left leg in order to create forward thrust during

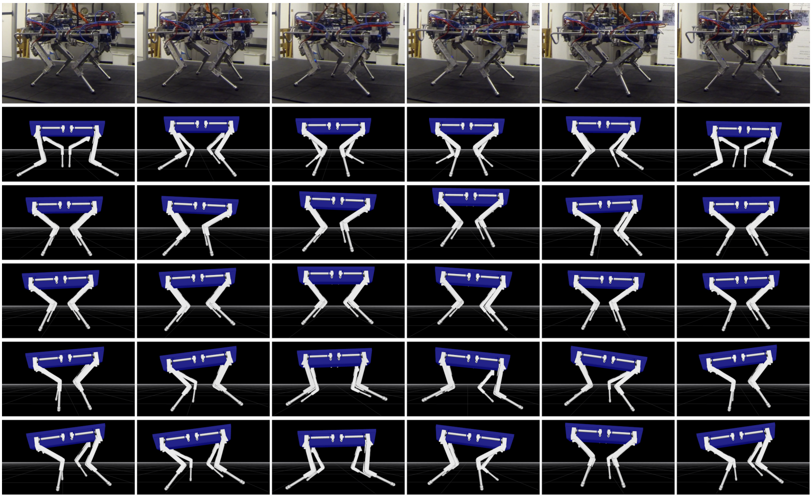


Figure 3.22: Time series of the optimized trajectories for the quadruped gaits. The picture series progresses from left to right and the robot moves to the right in the image plane. From top to bottom the gaits are: trot (hardware), flying trot (simulation), hop, bound, rotary gallop and transverse gallop.

the stance phase. The subsequent footstep locations are fairly equidistant until the robot returns to its initial configuration at the end of the trajectory. Since our humanoid model does not have ankles to produce torques, we can observe rhythmic, lateral tilting of the base. Since HyQ's “arms” (front legs) are lightweight and we penalize motions of the corresponding joints, the optimization keeps them close to their initial configuration during the motion. If more arm motion is desired, the cost function weights can be adjusted accordingly.

Run For the humanoid running gait, we reduce the contact duty cycle of the walk to 40%, leading to two flight phases of 10%, resulting in a total flight time of 20% of the cycle. As a result, the gait is more dynamic and the robot shows a small jumping motion between support phases. We still observe the left leg stepping back to initialize the gait.

Hop The last gait sequence we demonstrate on a humanoid is a symmetric hopping gait. Here, the contact duty cyle is set to 50% for both legs. In the optimized trajectory, the robot first hops back again to generate forward thrust. Afterwards, it executes two longer hops and then a shorter one to return to its resting posture at the goal position.

Quadruped

For the quadruped case, we demonstrate variations of four basic gait pattern types, resulting in six examples in simulation in total. Figure 3.22 shows a picture series for five gaits: trot, hop, bound, rotary gallop and transverse gallop. While the galloping gait exceeds the hardware limitations, we perform several hardware tests of the trotting gait.

Trot and Running Trot As a first example on the quadruped, we show a trot where diagonal legs are moved simultaneously. As with the humanoid walk we first demonstrate a gait pattern with no overlap, i.e. without flight phases or full support phases. The resulting trajectory is a trot that starts and ends in a four leg stance. From Figure 3.23 we can see that control effort is mainly required during the stance phase. The strongest torques can be seen in the Knee Flexion-Extension joint. Figure 3.25 shows the optimized contact forces. The largest contact forces occur during touchdown and just before lift-off. This is directly correlated with the joint torques in Figure 3.23. As expected, the contact forces in z-direction dominate, which corresponds to the vertical axis in world coordinates. However, most importantly, the contact force plots clearly demonstrate that forces are only applied during contact phases. This underlines that the contact relaxation method is successful in avoiding contact constraint violations. We also demonstrate a trot in simulation where we set the goal position to the front-left of the initial position and set the desired final yaw angle to 90 degrees. However, the cost function weights remains the same. As a result, HyQ trots in a curved motion. By reducing the contact duty cycle of the trot to 35%, we can also obtain a running trot with two flight phases.

To show that the optimized trajectories are physical and are directly applicable to the robot, we perform trotting tests on hardware. We test two variations of the trot: One with 50% duty cycle, i.e. no overlap of stance or flight phases and one with 60% duty cycle, i.e. a very brief four leg support phase. In both cases, we directly apply the optimized input trajectory as a feedforward torque command to the actuator. Additionally, we add a joint space PD controller to track joint positions. For reasons discussed in [168], we are not using the feedback control gains provided by SLQ. Figure 3.24 compares planned and measured base trajectories. While a base controller could enhance performance, our full model based approach allows us for successfully executing the trajectories for several seconds without base feedback.

Hop As a next task, we demonstrate symmetric hopping where the contact sequence is the same for all four legs. The resulting trajectory shows an almost constant forward velocity. This greatly reduces the involved control effort. If we decrease control effort weightings, we observe that the base almost comes to a complete stop during stance phases.

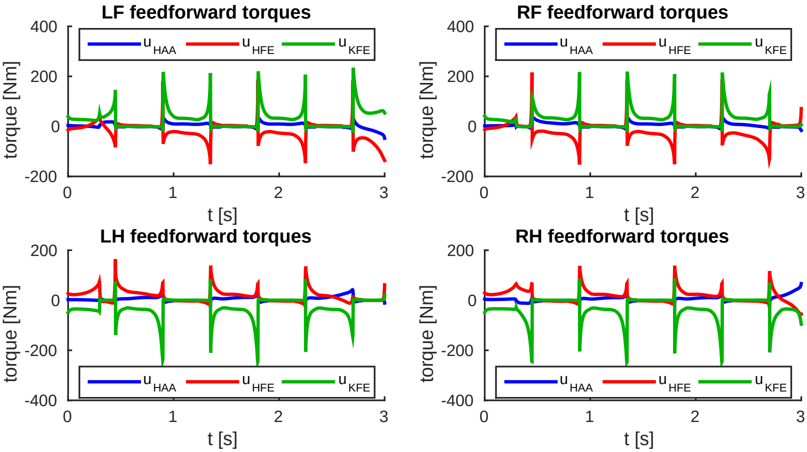


Figure 3.23: Feedforward joint torques Hip-Abduction-Adduction (HAA), Hip-Flexion-Extension (HFE) and Knee-Flexion-Extension (KFE) for all four legs of a quadruped during a trot. During the trot, the highest joint torques can be observed in the KFE joint.

Bound Another quadruped gait is bounding. Here the legs move in pairs, grouped in front and hind legs. Between two subsequent stance phases, there is a short flight phase. We observe a rhythmic pitching motion in the base. Interestingly, the optimized trajectory shows almost equidistant jumps except for the last jump, which is executed in place at the goal position. A possible reason could be that it is favorable to move with higher forward velocity given the dynamics and our cost function weightings. Reducing the time horizon by one gait cycle could potentially get rid of the final extra jump.

Rotary and Transverse Gallop As a final test, we demonstrate rotary and transverse galloping. Both tasks differ only in the gait pattern of the left front leg. These gaits allow the robot to travel a large distance using long stride lengths. When compared to a trot, these long stride lengths result in relatively slow joint velocities given the speed of the base. Comparing the results with galloping animals, we see significant similarities. One similarity is that the base is pitched downwards around the time both front legs are in stance and pitched upwards when the hind legs are in stance.

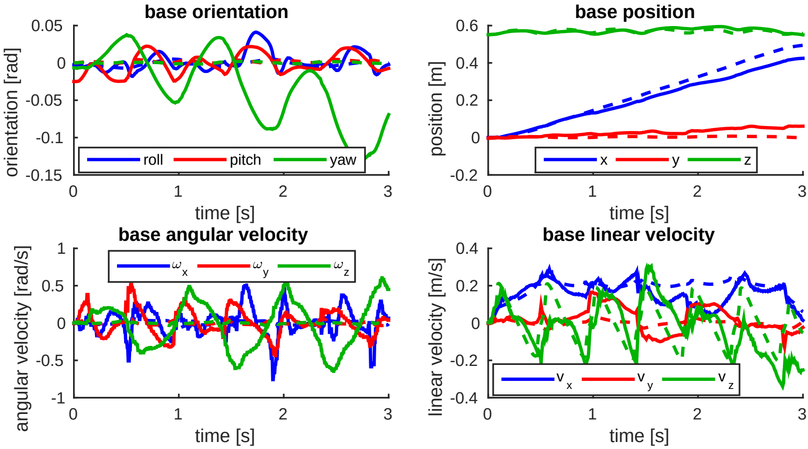


Figure 3.24: Comparison between planned (dashed) and measured (solid) base state trajectories for a trot executed on hardware. Using the full body model for optimizing the trajectories allows for executing them without base state feedback for several seconds.

Constraint Satisfaction

Since we are using soft constraints, we have to ensure that the obtained motions do not violate these constraints and result in unphysical trajectories. In the first iteration, we do not enable our soft constraints but allow the algorithm to find a good initial guess. Hence, after the first iteration the robot to “flies” to the goal position. For the second iteration we then enable the soft constraints. In the subsequent iterations, the total costs of the found solution stay flat or slightly increase since the algorithm is primarily trying to fix the constraint violation. So instead of looking at the cost function decay, we analyze the constraint violation cost. We focus on the quadruped results, since they involves more constraints to be satisfied. The results for the humanoid model are comparable. Figure 3.26 illustrate the constraint violations in input and end-effector position/velocity for all quadruped tasks as the Mean Squared Error (MSE). For the kinematic error, this is the sum of both the distance and velocity violation accumulated over all four legs. Similarly, the contact force constraint violation is also summed up over all four legs. As shown in the graphs, the algorithm finds solutions with constraint violations below levels of 10^{-3} in less than 18 iterations. To put these numbers in perspective, an unmodelled trunk weight of 0.01 kg would create an MSE in contact forces of 0.01. Kinematic violations are below the system’s control accuracy. Also hardware tests show that constraint violations are negligible.

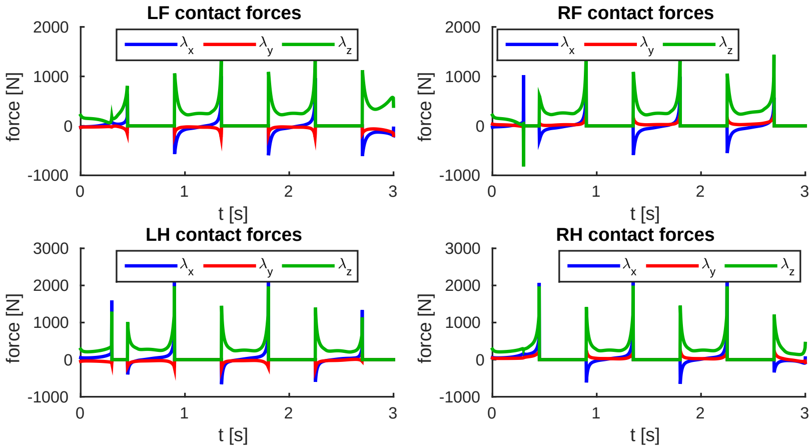


Figure 3.25: Contact forces during a trot of a quadrupedal robot. Corresponding to Figure 3.23 contact forces are high when joint torques are high. The z-direction corresponding to the vertical axis in world coordinates, is dominating. No contact forces are exerted on legs in swing phase.

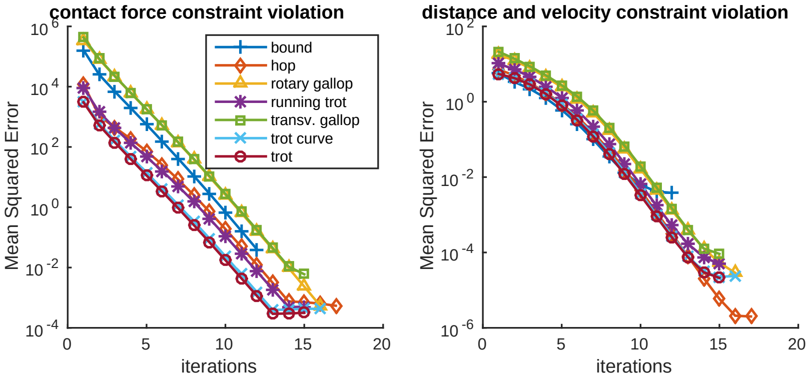


Figure 3.26: MSE of the constraint violation for the quadruped gaits. The left plot shows the sum of contact force constraint violations. The right plot shows the sum of the end-effector constraint violations. In both cases the violation monotonically decreases to usually below levels of 10^{-3}.

Runtime

Theory suggests that the runtime of SLQ scales linear with the time horizon. In previous work [168] we have demonstrated that this also holds true for our SLQ implementation. However, the runtime-per-iteration differs. On one hand, our control input dimensionality is increased compared to the previous approach and our cost function is more complex. On the other hand, we can take larger integration steps and we do not need to compute a contact model. In order to check whether the approach is fast enough to do online planning or even re-optimizing the trajectory during execution in Model Predictive Control (MPC) fashion, we measure the runtime of each iteration. The profiling of the algorithm is done for the quadruped case since it is the higher dimensional model. Given a time horizon of 3 seconds and a sampling frequency of 500 Hz, our solver requires around 800 ms per iteration on an Intel Core i7-4810MQ processor. As expected, this is slightly worse than the runtime presented in our previous work. However, given that we can take larger integration steps, the presented approach outperforms our previous one in runtime-per-iteration given the same time horizon.

As shown in the previous subsection, the presented examples converge to negligible constraint violation in 10 to 18 iterations from cold start. Thus, the overall computation time until convergence usually lies in the range of 8 to 15 seconds for trajectories of 3 seconds. In our examples, we cold start, use small integration steps and long time horizons. Considering that in an MPC approach we would use shorter time horizons and a very good initial guess is available from the previous iteration, we could possibly run the approach in MPC fashion with the current implementation. Additionally, part of the system dynamics linearization is using numerical differentiation which could be replaced by analytical derivatives. Also, hardware results show that the trajectories are robust, permitting low MPC update rates.

III.5 Discussion, Conclusion and Outlook

In this work, we have presented how an efficient Trajectory Optimization algorithm can be combined with constraint relaxation to solve complex motion planning tasks under switching contacts in a few seconds. Despite this relaxation, constraint violations are robustly reduced to a negligible amount. Since we are not using an explicit contact model, the problem appears to be more convex resulting in fast convergence. Compared to our previous work [168], the problem is also less sensitive to cost function parameters. However, this comes at the expense of having to pre-specify the contact sequence which can be both an advantage or a disadvantage. Hardware results demonstrate that the trajectories can be directly applied to the physical system. So far we are not considering contact friction, joint torque or kinematic limits. These could also be included as relaxed constraints or as shown in [71]. By including pre-specified contact locations as constraints, we are planning to apply this approach to rough terrain locomotion, possibly even in MPC fashion.

Paper IV: "Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds"

Authors

Michael Neunert, Markus Stäuble, Markus Giftthaler, Camine D. Bellicoso, Jan Carius, Christian Gehring, Marco Hutter, Jonas Buchli

Abstract

In this work we present a whole-body Nonlinear Model Predictive Control approach for Rigid Body Systems subject to contacts. We use a full dynamic system model which also includes explicit contact dynamics. Therefore, contact locations, sequences and timings are not prespecified but optimized by the solver. Yet, thorough numerical and software engineering allows for running the nonlinear Optimal Control solver at rates up to 190 Hz on a quadruped for a time horizon of half a second. This outperforms the state of the art by at least one order of magnitude. Hardware experiments in form of periodic and non-periodic tasks are applied to two quadrupeds with different actuation systems. The obtained results underline the performance, transferability and robustness of the approach.

Published in:
ArXiv preprint arXiv:1712.02889 [cs.RO], pp. , 2017

Notes:
To be published in Robotics and Automation Letters (RA-L) 2018 and to be presented at the IEEE International Conference on Robotics and Automation (ICRA) 2018

IV.1 Introduction

In this paper, we present a whole-body Nonlinear Model Predictive Control (NMPC) approach for Rigid Body Dynamics (RBD) systems subject to contacts. By using an explicit, Auto-Differentiable contact model as part of the dynamic system, the approach is able to reason about contacts and optimize through them efficiently. Contact timings, sequences or locations are not pre-specified, but an outcome of the optimization. Thanks to a highly-efficient, unconstrained nonlinear optimal control solver, we are able to successfully apply the method to two different quadruped platforms. We verify the performance and versatility of our approach by testing a multitude of tasks including periodic gait patterns as well as highly dynamic motions, such as squat jumps.

Related Work

Motion planning and control for legged, and especially quadruped locomotion is often tackled with multi-stage planning and control frameworks [35, 113, 143, 236]. Such frameworks often consist of one or multiple planner stages that use a simplified model and a reactive tracking controller. This results in the dilemma that the planner does not always produce feasible, i.e. dynamically consistent, plans or needs to plan conservatively. The tracking controller on the other hand does not have enough control authority to e.g. modify foothold positions or contact timings, but blindly tries to track the planners reference. In this field, centroidal dynamics approaches [46, 71, 102, 127, 128] become increasingly popular as they capture the core dynamics of the problem. However, many of these approaches plan or optimize contact forces that are not guaranteed to be realizable.

In recent years, there has been an increasing number of whole-body optimization and optimal control based approaches [143, 154, 181, 188, 217]. While these approaches are very complete, their runtimes are still a few orders of magnitudes away from running in receding horizon or MPC fashion. There are also some whole-body NMPC approaches verified in simulation [61]. However, without hardware validation, it remains an open question if and how well these approaches can be applied to real robots. While whole-body, contact invariant NMPC has been demonstrated on hardware before [123], the presented motions were rather slow or even quasi static, underlined by the fact that the authors do not apply the torque output to the robot but instead only use the position and velocity trajectories as references for the joint controllers. Additionally, contact switching was not dynamic and artificially enforced. Also stability during execution was explicitly encoded in the task.

Contributions

In this work, we demonstrate whole-body, contact invariant nonlinear MPC for highly dynamic motions that require explicit reasoning about the full dynamics of the system and the contacts. Furthermore, to the best of our knowledge, we demonstrate that such a framework can be applied to both single motions as well as periodic gaits on hardware for the first time. We show that the approach transfers

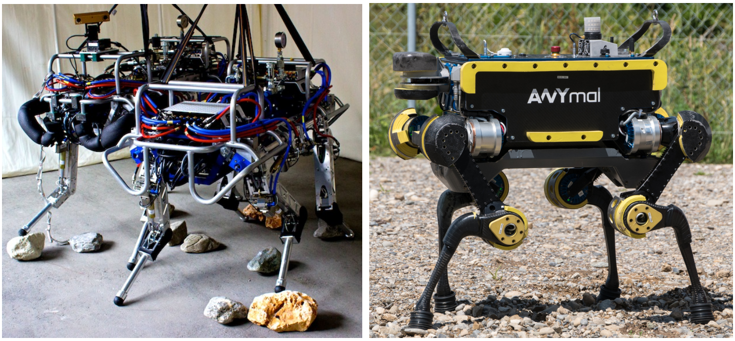


Figure 3.27: The quadrupeds HyQ-blue (left front) and ANYmal (right), which served as test platforms for our MPC experiments.

between platforms, and apply the same framework to the two quadrupeds HyQ and ANYmal (Figure 3.27). We also demonstrate the robustness and replanning capabilities of the approach by adding significant disturbances during execution. We summarize our solver framework, which uses Auto-Differentiation and code generation to achieve high computational performance exceeding the current state of the art in robotics NMPC applications by at least one order of magnitude. In contrast to many previous approaches, our solver is also available as open-source software [87]. Furthermore, we also publish the cost function weights used during experiments to ensure reproducibility.

In previous work [168], we demonstrated the versatility of optimizing whole-body motions through contacts. However, trajectories were only optimized once before execution. Also, especially interesting tasks such as periodic gaits could not be transferred to hardware due to model mismatches and lack of robustness of the plans. In this work, we demonstrate that an NMPC approach which continuously re-optimizes the state and control trajectories at high frequency, results in robust performance and copes with model mismatches. Running NMPC on real hardware poses severe restrictions in terms of computation time and software integration. In this work, we describe how we overcome these issues and improve our solver to achieve a speedup of several orders of magnitude.

Structure of this paper

We organize the paper as follows. In Section IV.2, we define how we formulate the NMPC problem for Rigid Body Dynamics Systems. In Section IV.3 we describe our approach of solving the problem. Afterwards, in Sections IV.4 and IV.5, we describe the implementation from a software point of view as well as the integration on hardware. In Section IV.6, we then present our experiments and discuss the results and findings. Finally, in Section IV.7 we summarize the paper and provide an outlook to future work.

IV.2 NMPC for Rigid Body Systems

The NMPC approach used in this work recurrently solves finite-horizon Optimal Control Problems with cost functions of the form

$$J(\boldsymbol{x}(t), \boldsymbol{u}(t)) = h\left(\boldsymbol{x}(t_f)\right) + \int_{t=0}^{t_f} L\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t\right) dt \tag{3.29}$$

and nonlinear system dynamics

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{3.30}$$

with state and control trajectories $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$.

System Modelling

Analogously to [168], we consider general Rigid Body Dynamics of the form

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{J}_c^\top \boldsymbol{\lambda}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{S}^\top \boldsymbol{\tau} \tag{3.31}$$

where $\boldsymbol{M}_{18\times18}$ is the inertia matrix, $\boldsymbol{C}_{18\times1}$ captures Coriolis and centripetal forces and $\boldsymbol{G}_{18\times1}$ represents the gravity terms. We further assume the system is actuated via joint forces/torques $\boldsymbol{\tau}_{12\times1}$ where the selection matrix $\boldsymbol{S}_{18\times12}$ maps these forces to the actuated degrees of freedom. Additionally, external forces $\boldsymbol{\lambda}_{12\times1}$ act on the system via the contact Jacobian $(\boldsymbol{J}_c)_{18\times12}$. The joint positions/angles and their velocities are denoted by $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ respectively. In the case of a floating base robot, these quantities also contain the base pose (i.e. its orientation and position) as well as the base twist (i.e. its linear and angular velocity). These quantities can also be thought of as an unactuated 6 DoF joint between an inertial and the base frame. We use an Euler-Angle parametrization of the 3D orientation.

We define the state of our system as follows

$$\boldsymbol{x} = [{}_{\mathcal{W}}\boldsymbol{q}^\top \ {}_{\mathcal{L}}\dot{\boldsymbol{q}}^\top]^\top = [{}_{\mathcal{W}}\boldsymbol{q}_B^\top \ {}_{\mathcal{W}}\boldsymbol{x}_B^\top \ \boldsymbol{q}_J^\top \ {}_{\mathcal{L}}\boldsymbol{\omega}_B^\top \ {}_{\mathcal{L}}\dot{\boldsymbol{x}}_B^\top \ \dot{\boldsymbol{q}}_J^\top]^\top \tag{3.32}$$

where the pose is expressed in an inertial "world" frame $\mathcal{W}$ and the twist is expressed in a local body frame $\mathcal{B}$. Due to the different reference frames, the twist is not a pure time derivative of the pose, but requires an additional coordinate transform $\boldsymbol{T}_{\mathcal{W}\mathcal{L}}$ which is composed of a pure rotation matrix for linear velocities as well as a slightly more complex mapping matrix for angular velocities. This leads to the overall system dynamics

$$\begin{aligned}\dot{\boldsymbol{x}}(t) &= \begin{bmatrix}{}_{\mathcal{W}}\dot{\boldsymbol{q}}^\top & {}_{\mathcal{L}}\ddot{\boldsymbol{q}}^\top\end{bmatrix}^\top \qquad (3.33)\\ &= \begin{bmatrix}\boldsymbol{T}_{\mathcal{W}\mathcal{L}} \ {}_{\mathcal{L}}\dot{\boldsymbol{q}} \\ \boldsymbol{M}^{-1}(\boldsymbol{q})(\boldsymbol{S}^\top\boldsymbol{\tau} + \boldsymbol{J}_c\boldsymbol{\lambda}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{G}(\boldsymbol{q}))\end{bmatrix}\end{aligned}$$

Contact Model

In order to avoid pre-specifying contact sequences, locations or timings, we need to enable the NMPC solver to reason about contacts. Some approaches resort to adding complementarity constraints to enforce contacts (e.g. [188]). However, these constraints do not satisfy the Linear Independence Constraint Qualification (LICQ) [196]. Almost all off-the-shelf Nonlinear Optimal Control or Nonlinear Programming solvers assume LICQ and, therefore, cannot handle these problems [31]. In contrast, we add the contact physics to our dynamic model using an explicit contact model. As a result, the contact forces become an explicit function of the robot's state: $\boldsymbol{\lambda}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{g}(\boldsymbol{x}(t))$.

Our contact model consists of a combination of linear springs and dampers perpendicular and parallel to the contact surface. For each end-effector we compute the contact model in the specialized contact frame C as follows:

$$\begin{aligned} {}_C\boldsymbol{\lambda}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = &- k \exp(\alpha_k \; {}_C\boldsymbol{p}_z(\boldsymbol{q})) \\ &- d \; \mathrm{sig}(\alpha_d \; {}_C\boldsymbol{p}_z(\boldsymbol{q})) \; {}_C\dot{\boldsymbol{p}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \end{aligned} \tag{3.34}$$

with d and k being damper and spring parameters. In order to achieve smooth derivatives, we multiply the damper-term with a sigmoid function of the normal component $\boldsymbol{p}_z(\boldsymbol{q})$ of the contact surface penetration $\boldsymbol{p}(\boldsymbol{q})$. Both the exponential and the sigmoid function serve as smoothing elements. Their 'sharpness' is controlled by α_k and α_d. Finally, the contact forces are transformed into the robot body frame by

$${}_B\boldsymbol{\lambda}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{R}_{WB}(\boldsymbol{q})_C\boldsymbol{\lambda}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \tag{3.35}$$

and subsequently passed to the forward dynamics where they act on the corresponding link. Our particular choice of the contact model smoothing supports the gradient-based solver by ensuring that contact forces never completely vanish. Therefore, the solver can reason about contacts even before they are established. While this is slightly nonphysical, we will later see that this does not hinder good performance on hardware. We emphasize that there exists physically more accurate explicit contact models [9] and implicit contact models as popular in physics engines. The latter however rely on optimization based solvers which cannot be differentiated well. In contrast, our simplified model captures the governing effects accurately enough and allows for computing derivatives efficiently by using Auto-Diff which is key to solving the NMPC problem fast enough [85].

IV.3 NMPC Approach

Informally speaking, NMPC is achieved through solving the Nonlinear Optimal Control (NLOC) problem at sufficiently high rates. Popular approaches to nonlinear optimal control are Single Shooting, Multiple Shooting or Direct Collocation [53], which discretize the continuous time optimal control problem and transcribe it into a Nonlinear Program (NLP). These NLPs are often solved using general off-the-shelf

NLP solvers as presented in [181, 189]. However, such an approach does not fully exploit the sparsity structure inherent to optimal control problems and often results in poor algorithm runtimes, which are not fast enough for MPC applications.

To overcome this issue, we formulate our problem as an unconstrained optimal control problem, and resort to an optimized, custom solver, that implements a family of iterative Gauss-Newton NLOC algorithms [84]. The solver employs a first-order method that locally approximates the NLOC problem as a Linear Quadratic Optimal Control (LQOC) problem using a Gauss-Newton Hessian approximation. The LQOC is solved by a Riccati-based solver which has linear complexity in the time horizon. This makes the approach efficient for larger time horizons. Our solver can be considered a generalization of the well-known iLQR [222] and SLQ [211] algorithms and covers both Single and Multiple Shooting. It designs time-varying state-feedback controllers of the form

$$\boldsymbol{u}_n(\boldsymbol{x}) = \boldsymbol{u}_n^{ff} + \boldsymbol{K}_n(\boldsymbol{x}_n - \boldsymbol{x}_n^{\mathrm{ref}}) \tag{3.36}$$

where $\boldsymbol{u}_n^{ff}$ is the feedforward control action and $\boldsymbol{K}_n$ a linear feedback controller regulating deviations of the state $\boldsymbol{x}_n$ from the reference trajectory $\boldsymbol{x}_n^{\mathrm{ref}}$. For most experiments in this paper, we use the iLQR algorithm. We furthermore compare it to the more efficient Gauss-Newton Multiple Shooting (GNMS) approach which can act as a direct replacement. Both algorithms use the same approach of formulating and solving a local LQOC problem, however their MPC formulations varies.

A summary of the iLQR-NMPC algorithm, is given in Algorithm 4. It shows two forward integration steps during the algorithm. One directly after retrieving the state measurement to get the nominal trajectory, and a second one during the line search after updating the controller. For our application, the latter is important to obtain a new reference trajectory for the feedback controller to track. In contrast, the GNMS-NMPC algorithm, which is summarized in Algorithm 5, designs a state reference trajectory simultaneously with the new control policy. Furthermore, it allows to separate the algorithm into a *feedback* and a *preparation* phase [52], which helps to minimize the latency between state-measurement and control policy update. For a detailed overview about the GNMS algorithm, the reader is referred to [84]. An open-source reference implementation is available in [87].

IV.4 Software Implementation

Running NMPC for a high dimensional system in real-time remains a challenge despite the powerful consumer PCs available today. While the development of processors with faster clock speed has stalled in recent years, processing power instead foremost grows due to higher computation core counts as well as vectorization. However, both parallel execution and vectorization cannot be leveraged automatically by standard compilers. Also, many computational routines such as integrating a differential equation over time, are naturally sequential operations that cannot be parallelized easily. In this subsection we describe how we optimize the NMPC solver from a numerical point of view and leverage the processor architecture to reduce the computational burden.

Algorithm 4 Discrete-time iLQR-MPC Algorithm

```
Given
- cost function (3.29) and system dynamics (3.30).
- receding MPC time horizon $N$.
- stable initial control policy $\boldsymbol{u}_n(\boldsymbol{x})$ of form (3.36)
Repeat Online:
- get state measurement $\boldsymbol{x}_{\text{meas}}$.
- forward integrate system dynamics (3.30) with $\boldsymbol{x}_0 = \boldsymbol{x}_{\text{meas}}$ to obtain
  state trajectories $\boldsymbol{X} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, control trajectories
  $\boldsymbol{U} = \{\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{N-1}\}$ and corresponding sensitivities $\boldsymbol{A}_n$, $\boldsymbol{B}_n$.
- quadratize cost function (3.29) around $\boldsymbol{X}$ and $\boldsymbol{U}$
- solve LQOC problem using a Riccati backward sweep
- retrieve control policy $\boldsymbol{u}_n^+(\boldsymbol{x})$ of form (3.36)
- line search over the control increment $(\boldsymbol{u}_n(\boldsymbol{x})^+ - \boldsymbol{u}_n(\boldsymbol{x}))$
  and update $\boldsymbol{X}^+$ by means of a forward simulation of the nonlinear
  dynamics (3.30) with $\boldsymbol{x}_0 = \boldsymbol{x}_{\text{meas}}$
- send policy $\boldsymbol{u}_n^+(\boldsymbol{x})$ and $\boldsymbol{X}^+$ to the robot tracking controller
- update $\boldsymbol{u}_n(\boldsymbol{x}) \leftarrow \boldsymbol{u}_n^+(\boldsymbol{x})$
```

Algorithm 5 Discrete-time GNMS-NMPC Algorithm

```
Given
- cost function (3.29) and system dynamics (3.30).
- receding MPC time horizon $N$.
- initial state and control trajectories $\boldsymbol{X} = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$,
  $\boldsymbol{U} = \{\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{N-1}\}$
Repeat Online:
Feedback phase
- get state measurement $\boldsymbol{x}_{\text{meas}}$.
- forward integrate system dynamics (3.30) with $\boldsymbol{x}_0 = \boldsymbol{x}_{\text{meas}}$ on the first
  multiple-shooting interval, obtain sensitivities $\boldsymbol{A}_0$, $\boldsymbol{B}_0$.
- quadratize cost function (3.29) around $\boldsymbol{X}$ and $\boldsymbol{U}$ for first control stage
- solve LQOC problem using a Riccati backward sweep
- retrieve updated control policy $\boldsymbol{u}_n^+(\boldsymbol{x})$ and updated trajectories $\boldsymbol{U}^+$, $\boldsymbol{X}^+$.
- send policy $\boldsymbol{u}_n^+(\boldsymbol{x})$ and $\boldsymbol{X}^+$ to the robot tracking controller
Preparation phase
- update: $\boldsymbol{u}_n(\boldsymbol{x}) \leftarrow \boldsymbol{u}_n^+(\boldsymbol{x})$, $\boldsymbol{X} \leftarrow \boldsymbol{X}^+$, $\boldsymbol{U} \leftarrow \boldsymbol{U}^+$
- forward integrate system dynamics (3.30) for the multiple-shooting
  intervals 1 to $N$, obtain sensitivities $\boldsymbol{A}_1, \ldots \boldsymbol{A}_{N-1}$, $\boldsymbol{B}_1, \ldots, \boldsymbol{B}_{N-1}$.
- quadratize cost function (3.29) around $\boldsymbol{X}$, $\boldsymbol{U}$ for multiple-shooting intervals 1 to $N$.
```

Modelling Framework

Our NMPC controller relies heavily on evaluating Rigid Body Dynamics and Kinematics. Therefore, we use RobCoGen [79], an efficient code generation framework for modelling Rigid Body Dynamics. In [85] we augmented RobCoGen to be compatible with Auto-Differentiation as implemented in our Control Toolbox (CT) [87]. Furthermore, we use CT's contact model and kinematics that wrap around RobCoGen to provide contact force mappings. The resulting framework is lean compared to sophisticated physics engines and produces fast to evaluate, hard realtime capable code.

Integration and Sensitivity Computation

Our system dynamics include a contact model that needs to be chosen stiff enough to approximate the real physics of contact well. Naturally, this leads to a numerical stiffness which requires us to take small integration time-steps. Therefore, instead of using standard explicit integrators such as Euler or Runge-Kutta schemes, we use symplectic (or semi-implicit) integrators, which are also popular in physics engines [40]. A symplectic integrator alternates between integrating positions and velocities, i.e. the updated positions are used to compute the velocity update and vice versa. Therefore, the computational complexity is similar to an explicit scheme, but the numerical stability is increased. In contrast to implicit integrators, we do not have to compute Jacobians explicitly and do not have to solve an internal optimization problem. The symplectic integrator allows us to increase the integration step size by a factor of four compared to explicit schemes [168].

Our LQOC solver requires us to compute sensitivities along the trajectory, i.e. partial derivatives of the integrated state with respect to the start state and control action of the step. These sensitivities can be understood as matrices $\boldsymbol{A}_n$ and $\boldsymbol{B}_n$ in a local linear approximation of the form $\boldsymbol{x}_{n+1} = \boldsymbol{A}_n \boldsymbol{x}_n + \boldsymbol{B}_n \boldsymbol{u}_n + \boldsymbol{c}_n$. While many existing approaches compute the sensitivities numerically [123], it is slow and can lead to severe numerical problems hindering convergence [50]. Therefore, we compute them exactly by integrating a corresponding sensitivity ODE. A description of the integration scheme for sensitivities of symplectic integrators can be found in [190]. The linearization of our dynamic system is performed with Auto-Diff and code-generation described in detail in [85], which provides the same accuracy as analytic derivatives but outperforms them in terms of computational speed.

Multithreading and Vectorization

Another important factor for obtaining best performance is multi-threading. Some parts of our algorithm are inherently sequential, for example solving the LQOC problem backwards in time, or the forward simulation of the system when employing iLQR. However, using a multiple-shooting approach allows us to parallelize the forward simulation over the individual multiple-shooting intervals. In this case, computation time decreases linearly with the number of cores. In contrast, iLQR requires to compute a single, continuous forward simulation and thus does not benefit from a multi-core processor in this step. The cost and sensitivity computation, which can be distributed among all available cores, is parallelizable for all our algorithm variants.

Another optimization possibility is to use the processor's vectorization capabilities, which are Single Instruction Multiple Data (SIMD) implementations, especially useful for mathematical operations on vectors and matrices. In a previous implementation [168], we had already used SSE [75] instructions. In this implementation, we switched to AVX [75] instructions. Since our code mostly consists of matrix and vector manipulations and register sizes of AVX are doubled over SSE, we obtained an additional speedup of almost a factor of two. This number is also supported by the release notes of the Eigen library [93] used for all computations. With the release of AVX512 doubling register sizes yet again, we expect another speedup of about factor of two.

IV.5 Hardware Integration

Platform Descriptions

For hardware experiments we use two different quadruped robots, that strongly vary in size, weight and actuation principles. Therefore, the experiments underline the generality of the approach and show that no platform specific modifications are required. HyQ is an 80 kg, hydraulically actuated quadruped, while ANYmal weighs around 34 kg and uses electric, series-elastic actuators. Both robots are briefly described in the following.

HyQ HyQ [205] is a fully torque controlled, hydraulic quadruped built by the Italian Institute of Technology. It features three joints per leg, namely Hip Abduction Adduction (HAA), Hip Flexion Extension (HFE) and Knee Flexion Extension (KFE). All joints are equipped with absolute and relative encoders, the joint torques are measured by load cells.

ANYmal While being more compact, ANYmal [107] features the same joint configuration as HyQ. Furthermore, it is fully torque controlled via series-elastic actuators. Joint encoders before and after the elastic element measure its deflection which is used to compute the internal joint torque.

Tracking Controller

The control and tracking framework mostly corresponds to the one shown in Figure 3.28 also used in [168]. When running our NMPC framework, we directly apply the optimized torques as feedforward signal to the actuators. However, in our framework, there is an average delay of about 10-15 ms between state measurement and execution of the corresponding, optimized policy on the robot. This delay prevents from robustly controlling positions and velocities of the swing

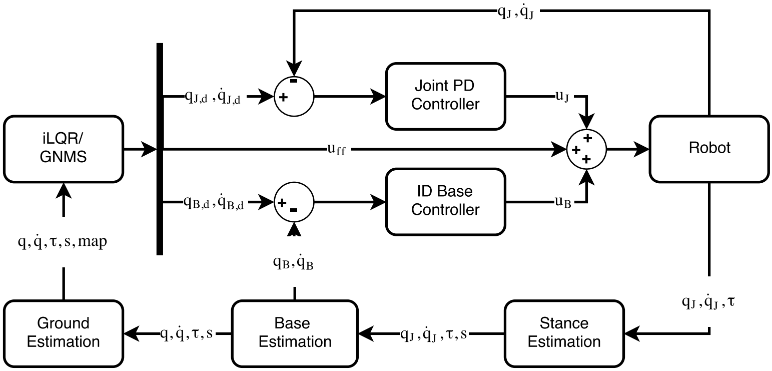


Figure 3.28: Structure of the estimation and control approach for hardware execution of the NMPC controller. Estimators estimate ground height and base state information. The optimized control input obtained from the NMPC solver is then augmented with the output of two tracking controllers.

legs. Therefore, we add a PD control loop around the optimized joint trajectories that improves position and velocity tracking. Lastly, we also add a virtual model controller [191] using inverse dynamics (without gravity compensation) which only contributes a few percent of the overall control signal. Therefore, the presented tasks also work without the base controller but show slightly improved performance with the controller enabled.

State Estimation

Both quadrupeds run a state estimator that fuses measurements from an Inertial Measurement Unit (IMU) and the leg encoders to estimate the base pose and twist of the robot. The state estimator is described in [19]. Joint position measurements are directly obtained from both robots and are then numerically differentiated to obtain joint velocities. Since we use an explicit contact model, an estimation of the ground is required. Here we check the stance of each foot and fit a plane through all stance legs. This ground estimator could possibly be replaced in the future by a mapping approach that delivers differentiable height information.

Computing Setup

In our setup, we use a dedicated computer that runs the NMPC control loop. The NMPC-node receives the current state of the robot via the Robot Operating System (ROS) from the midlevel control computer that executes the tracking controller described in subsection IV.5. Due to different software and hardware architectures, this tracking controller runs at 250 Hz on HyQ and at 400 Hz on ANYmal. The midlevel controller then sends desired torque, position and velocity setpoints to a lowlevel torque controller. In return, it receives current state measurements and computes the base and ground state estimate. The lowlevel controller runs a torque tracking and a position PD controller at 1 kHz on HyQ and 2.5 kHz on ANYmal. This controller is implemented on embedded hard real-time systems on both robots.

IV.6 Results

The performance of our algorithms is assessed on both quadrupeds. We test a periodic trotting gait on both robots and disturb them during the tests. Furthermore, we execute additional dynamic motions on ANYmal. For all experiments, the cost functions weights are visualized in figure 3.29 and provided as supplementary material.

In all experiments, we employ a time horizon of 500 ms, a control discretization of 4 ms and an integration rate of 1 ms for the aforementioned symplectic Euler scheme. In this work we employ the iLQR and GNMS solvers from [84], and use the solution from the previous MPC iteration as a warm start. We run a so called "real-time iteration scheme" [53], where we apply the optimized trajectory after a single iteration. Note that running only a single solver iteration before updating the state measurement results in better overall performance than running multiple iterations and letting the solver converge. All results are obtained with the same settings, solver and model. The different behaviors are thus simply a result of the choice of cost function and weights, offering a great versatility.

Hardware Experiments HyQ

Trotting As a first test, we investigate a periodic gait pattern, encouraged over periodically activated costs penalizing joint angle deviations from a desired swing leg apex configuration. In previous work [168], we have demonstrated that our approach can also discover a trotting gait without swing leg costs. However, by adding such costs, we can influence the gait frequency and encourage trotting while staying in place. Since we only penalize the apex height of the swing leg, exact contact timings are not specified but can be adjusted by the algorithm. Additionally, since the swing phase is not a constraint, it can be violated by the algorithm if helpful for the overall performance. This can be observed during execution. If the robot base is strongly pushed to one side, the feet on that side are not lifted but remain on the ground to first stabilize the base. This means the algorithm naturally modifies the gait pattern and contact timings to optimize performance. Another observation is that, due to a feedforward dominant controller that plans ahead of time instead of simply reacting aggressively to disturbances, we obtain a compliant controller. HyQ can be perturbed significantly both on the base and the legs without reacting stiffly. Even placing planks under single feet does not deteriorate performance. Figure 3.30 shows the results of a trotting experiment on HyQ. The plot shows that the base orientation and position deviations from the

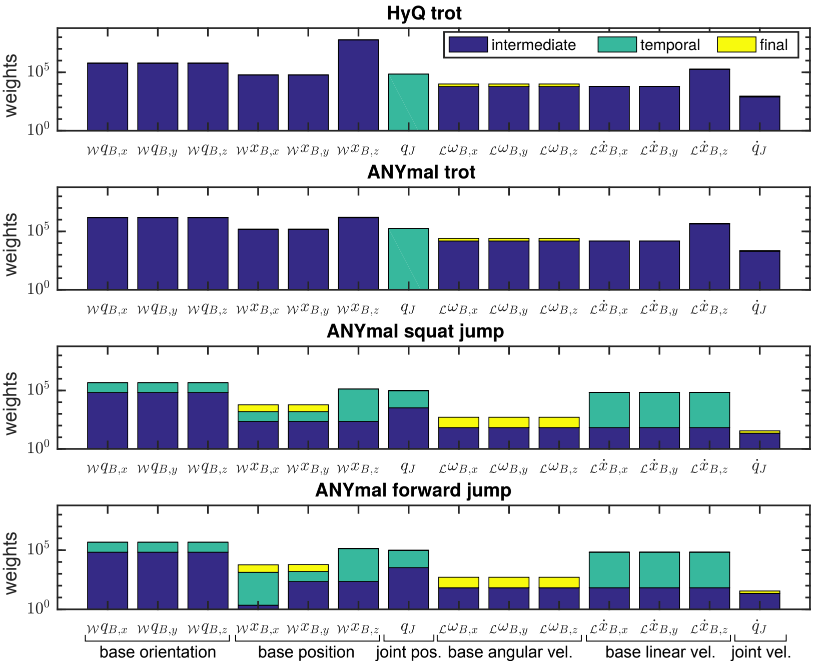


Figure 3.29: Cost function weights for the different hardware experiments plotted on a logarithmic scale. All weighting matrices are diagonal and thus the weights illustrated correspond to the diagonal entries. Weights on joint positions q_J and velocities $\dot{q}_J$ are the same for all legs. The temporal costs in the trotting experiment affect swing leg pairs and are activated periodically. For the squat and forward jump temporal costs affect all legs equally and are activated once per jump.

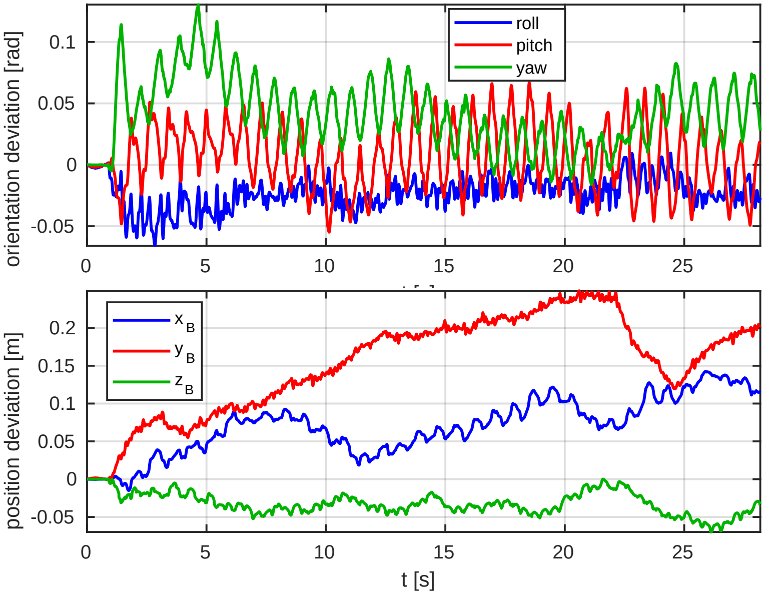


Figure 3.30: Plots of the deviation of the orientation and position offset during trotting experiments on HyQ. In the cost function, we penalize orientation stronger than position which shows in the plots. Compared to ANYmal the magnitude of the deviations is slightly larger. However, this is in part also due to the different sizes of both robots.

set point are regulated and remain small. Also, we add a strong cost penalty on the base orientation to improve stability. The resulting overall controller is stable and can robustly handle aforementioned disturbances.

Hardware Experiments ANYmal

Trotting We repeat the same trotting experiments with adjusted weights (due to different mass/inertias) on ANYmal. Also here we observe that the controller is robust to disturbances. By adjusting the desired position and orientation of the base with a joystick, we navigate the robot around. Figure 3.31 show measurements of the robot base during the trotting experiment. These plots illustrate how the MPC controller deals with disturbances. The robot is executing a periodic trot motion in place. At $t = 2.8s$ we placed a wooden plank below one of ANYmal's feet. For the duration of the disturbance the MPC controller escapes the periodic

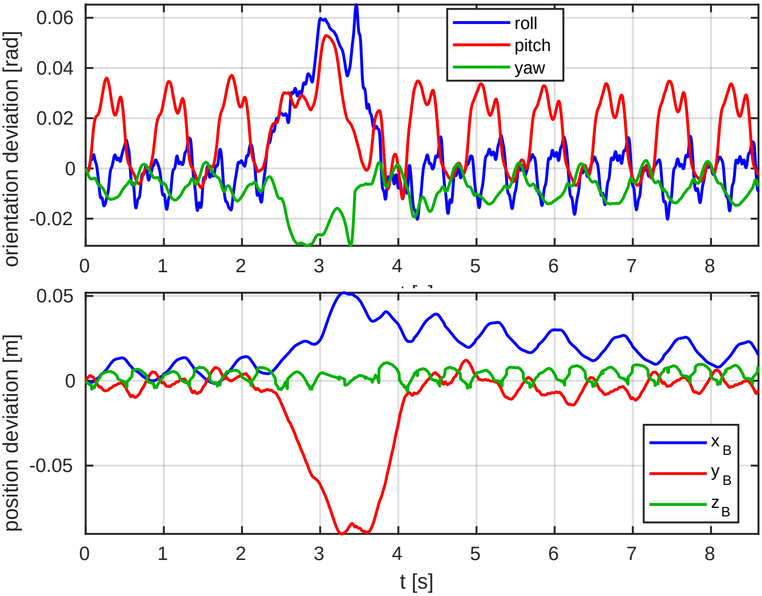


Figure 3.31: Base orientation (top) and position (bottom) of ANYmal during a trot. At t = 2.8s we placed a wooden stick below one foot which acted as heavy disturbance on the system. The controller is able to return to a periodic motion after the disturbance is removed.

trot and finds different solutions. To mitigate the disturbance the controller shifts and rotates the base and when the disturbance is gone it returns to the periodic trot. Compared to HyQ the deviations of the base orientation and position are much smaller. One of the reasons for this is certainly the size and weight differences between these two robots.

Squat Jumps To underline that the approach can leverage the full system dynamics, we also perform a squat jump. Here, amongst the usual running and final cost, a temporal cost encourages an upward base velocity at a certain time point. However, the lift-off time and especially the landing time are not pre-specified. Also, while it is not surprising that all legs lift-off at the same time, we do not explicitly enforce it. To test repeated jumps, we activate the vertical velocity costs periodically. While the robot does not always land perfectly, the MPC controller optimizes a trajectory from the current state and tries to get back as close as possible to the nominal

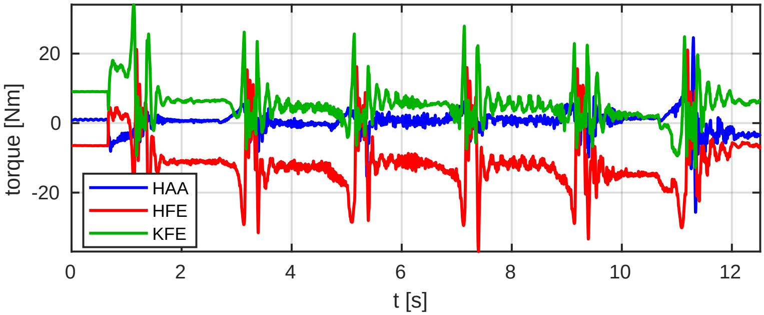


Figure 3.32: Joint torques of ANYmal during a repeated squat jump. The torques stay well below the physical torque limit of 40 Nm. Furthermore, the torques drop quite abruptly after take off and then peak during the landing phase.

state. This experiment underlines the robustness of the approach, as several squat jumps can be executed without resetting the robot to the nominal configuration. Figures 3.33 and 3.32 show the measurements of ANYmal during the squat jump experiments. We enforce jumps every 2 seconds, starting at $t = 1s$. The desired take off velocity was 1.0 m/s in vertical direction (z axis) leading to an apex height of 3 cm. The measurements show that the robot slightly overshoots the velocity setpoint by 0.1 m/s during execution. On the torque level the robot stayed well below the admissible torque level of ANYmal (40 Nm) without imposing additional constraints. The robot drifts in x and y direction between consecutive squat jumps. The reason for this is that we do not impose large cost penalties in x and y positions for stability reasons. One interesting observation is that the base height is lowered and the knees are bent before jumping off to accumulate kinetic energy, which reduces the maximum torque used for a given height.

Forward Jump If we add a forward velocity component to the squat jump, we obtain a forward jump behavior. Again, lift-off time and landing time are not pre-specifed. Results of the forward jump can be found in the submitted video.

Timings

In this section, we present timings for different solver setups. For all timings and experiments, the NMPC solver is run on an Intel Core i7 4790 quadcore PC with 3.6 GHz.

Figure 3.34 shows the timings obtained from the trotting experiments on ANYmal. We measured the rate of incoming trajectories that the tracking controller received. It can be seen that our solver runs at around 80 Hz for when using the iLQR-algorithm and at 175 Hz when using Gauss-Newton Multiple-Shooting. The higher

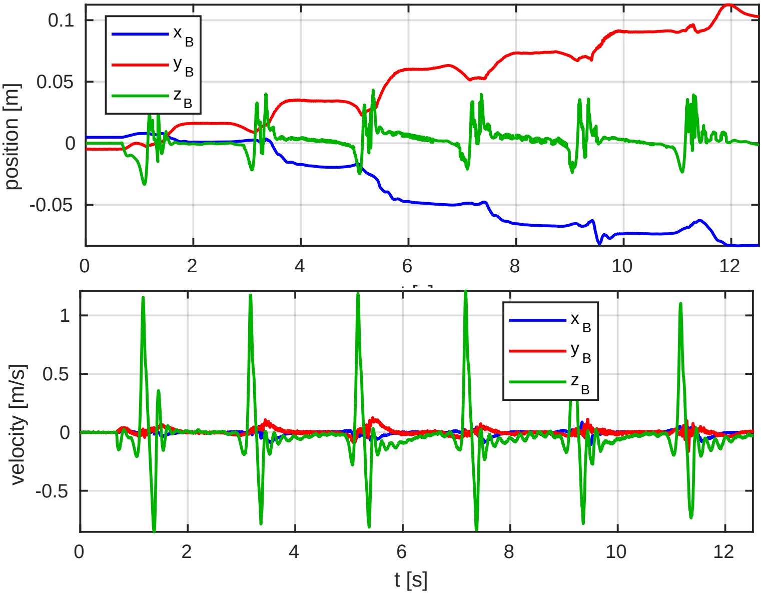


Figure 3.33: Base position and linear velocity of ANYmal during a repeated squat jump. We mostly penalize base orientation and linear velocities to obtain straight jump motions. As a result, the controller achieves a constant apex height but drifts slightly in x and y directions. We observe that ANYmal first slightly bents its knees and lowers the base before jumping off to accumulate energy, a behaviour that would not result from a pure feedback controller.

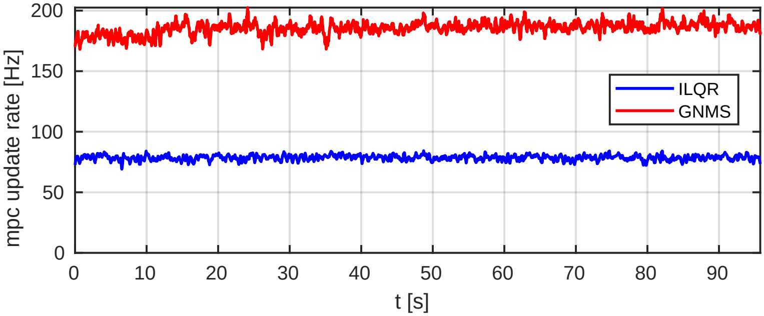


Figure 3.34: MPC update rate as recorded during two trotting experiments on ANYmal. While iLQR achieves update rates of around 80 Hz, GNMS reaches almost 190 Hz. While the higher update rate does not lead to significantly better performance, it leaves more headroom for extending the time horizon.

frequency of GNMS-MPC results from three main factors: first, the parallel forward simulation, second, in contrast to iLQR, GNMS does not require a line-search for this particular problem, and third, the GNMS algorithm allows for using a higher control discretization (6 ms). While GNMS offers a higher update rate, it is not very noticeable in performance such that iLQR performs similarly well. This is illustrated in Figure 3.35 which compares the costs of the trajectories obtained from both algorithms during a trot task. However, we notice that below 30 Hz update rate, the performance on hardware starts to degrade significantly. Therefore, GNMS leaves more headroom for more complex systems or longer time horizons.

At 80 Hz, a single computation takes less than 12 ms. Compared to [123], which is using the same algorithm but requires around 50 ms per iteration, our solver is about 300% faster for the same time horizon and similar number of degrees of freedom while having a much finer control discretization (4 ms instead of 20 ms) and only using a third of the CPU cores. Assuming the same number of cores and the same discretization, our solver would run about 10-15 times faster than the one presented in [123]. This results from a more efficient solver implementation, optimized vectorization, faster computation of the dynamics due to a simpler contact model as well as using Auto-Diff code-generation. As a result, delays are significantly reduced, which is crucial for robustness.

IV.7 Summary and Outlook

The presented work shows the first application of whole-body NMPC through contacts for dynamic motions. Furthermore, this is the first time that such an approach is applied to hardware for periodic gait patterns and tasks with dynamic contact switches. We demonstrate that NMPC can be run at rates that exceed

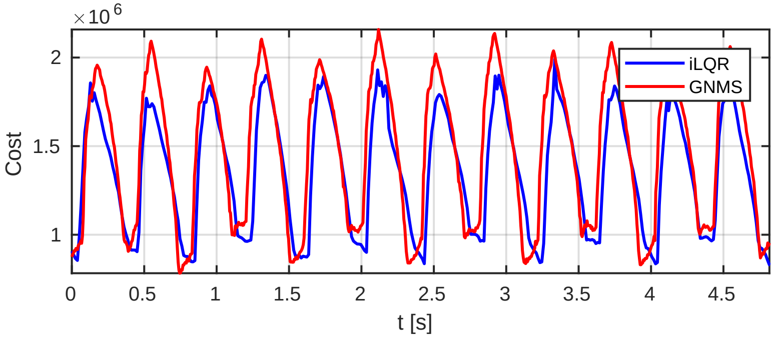


Figure 3.35: Total cost of the iLQR and GNMS algorithms during the trot experiment. Both algorithms result in a similar cost after optimization.

the state of the art by an order of magnitude by using an efficient implementation combined with state-of-the-art software engineering techniques such as Auto-Diff, symplectic integration as well as vectorization. By applying our method to two different robots, we show that our integration with state estimation and controllers allows us to deploy the approach to different robots with different actuation principles without major adjustments. The tasks themselves underline the benefits of running an NMPC controller that can reason about contacts. When looking e.g. at the trotting task, we see disturbance-dependent gait pattern changes, base stabilization and small reactive side stepping. If implemented with a classical approach the resulting planning and control pipeline would possible consist of several modules and layers. Using NMPC they emerge naturally from a single algorithm. While cost function tuning remains a manual task to achieve optimal performance, new tasks can be simply tuned in a few minutes. This way, we avoid designing a control and planning framework from scratch when the task at hand changes, which is still often the case in robotics.

While these first results look promising, there is still a significant amount of future work to be considered, both from an algorithmic as well as from an implementation point of view. On the algorithmic side, we plan to leverage the speed advantage and better warm starting capabilities of GNMS to extend the time horizon of the NMPC controller. As part of this work, it would be worthwhile to see how a larger time horizon influences performance and robustness. We expect that a longer time horizon could show more elaborate disturbance rejection and recovery behavior since it offers more flexibility and predictive capabilities to the solver. Furthermore, while most tasks by design stayed within the physical limitations of the platforms, GNMS would allow us to handle constraints such as torque limitations explicitly as as inequality constraints. Given that the higher update rate of GNMS-MPC did not lead to an increase in performance, we believe that our method can cope with the extra computational complexity when dealing with constraints, especially since the algorithm's complexity remains linear in time. Also, we could resort to soft-constraints rather than hard constraints [166]. From a practical perspective, we wish to include some model adaptation or disturbance estimation to our implementation that robustifies our approach even further. Additionally, this could help to bring new motions to hardware. Finally, the NMPC controller could also be used to track and stabilize motions generated from a high level foothold and motion planner to handle non-convex terrains.

Acknowledgement

We wish to thank the entire SYSCOP group at IMTEK at the University of Freiburg, especially Prof. Moritz Diehl and Dimitris Kouzoupis for their valuable feedback and Gianluca Frison for his support for interfacing with the HPIPM solver. Furthermore, our gratitude goes to Marko Bjelonic and Péter Fankhauser who helped with the experiments.

Chapter 4

Part B: Applying Numerical Optimal Control to Mobile Robots

Paper V: "Automatic Differentiation of Rigid Body Dynamics for Optimal Control and Estimation"

Authors

Markus Giftthaler[†], Michael Neunert[†], Markus Stäuble, Marco Frigerio, Claudio Semini and Jonas Buchli ([†]these authors contributed equally to this work)

Abstract

Many algorithms for control, optimization and estimation in robotics depend on derivatives of the underlying system dynamics, e.g. to compute linearizations or gradient directions. However, we show that when dealing with Rigid Body Dynamics, these derivatives are difficult to derive analytically and to implement efficiently. To overcome this issue, we extend the modelling tool 'RobCoGen' to be compatible with Automatic Differentiation. Additionally, we propose how to automatically obtain the derivatives and generate highly efficient source code. We highlight the flexibility and performance of the approach in two application examples. First, we show a Trajectory Optimization example for the quadrupedal robot HyQ, which employs auto-differentiation on the dynamics including a contact model. Second, we present a hardware experiment in which a 6 DoF robotic arm avoids a randomly moving obstacle in a go-to task by fast, dynamic re-planning. This paper is an extended version of [167].

Published in:
Advanced Robotics, pp. 1225–1237, 2017

Award:
The original conference paper has been awarded with the *Best Paper Award* at SIMPAR 2016 and thus the authors received an invitation to submit an extended version to the Advanced Robotics SIMPAR special issue.

Notes:
This paper is an extended version of: *Michael Neunert, Markus Giftthaler, Marco Frigerio, Claudio Semini, Jonas Buchli. Fast Derivatives of Rigid Body Dynamics for Control, Optimization and Estimation. In Proceedings of the IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), 2016, pp. 91–97*

V.1 Introduction

Most robotic systems consist of multiple rigid links connected via joints. This includes entire systems such as robotic arms, legged robots or exoskeletons, but also components like robotic hands or integrated actuators. We can mathematically model these systems using Rigid Body Dynamics (RBD). The latter results in non-linear differential equations for which computing a closed-form solution is intractable. Therefore, many state-of-the-art approaches in control, estimation, optimization and planning rely on iterative, gradient-based algorithms which employ linear approximations of the given system equations. These algorithms include optimal controllers such as Linear Quadratic Regulators (LQR), optimal estimation approaches such as Kalman Filters and Batch Estimation, as well as parametric design optimization and Trajectory Optimization (TO), e.g. Direct Transcription, Direct Multiple Shooting and Differential Dynamic Programming.

To obtain the required derivatives for these algorithms, there are several options. Often, analytic expressions 'manually-derived on paper' are considered the ideal solution since they are accurate and fast to compute. However, as shown later, the expressions obtained from deriving RBD derivatives manually are fairly complex and difficult to optimize. Therefore, they often lead to poor runtimes. Additionally, the manual process is error prone. To avoid this step, symbolic toolboxes such as Matlab [145], Mathematica [240] or Maxima [146] can be employed. These tools apply known calculus rules in order to symbolically determine the derivatives. While this approach is viable in theory, in practice the derivative expressions easily get too large to be suitable for a computationally efficient implementation.

As a shortcut to the previously mentioned approaches, Numeric Differentiation (Num-Diff) is frequently used. In this method, the input to the function to be differentiated is perturbed in each input dimension to obtain an approximation of the derivative using finite differences. However, these methods are prone to numeric errors and they are computationally costly when the input dimension is high.

As another option, Automatic Differentiation can be used. Automatic Differentiation (Auto-Diff) is a programming approach for obtaining the derivatives from source code instructions. Auto-Diff builds an expression graph of the original function which is later differentiated using the chain rule and known 'atomic' derivatives. Since Auto-Diff operates on expressions rather than numerical values, it provides the same accuracy as analytical derivatives. Yet, it still offers the comfort of obtaining the derivatives in an automated fashion. Additionally, Auto-Diff derivatives usually have lower complexity than unoptimized analytical derivatives.

Automatic Differentiation Tools

Over the years, several Automatic Differentiation tools have been developed. One implementation approach is Source Code Transform, i.e. parsing the (uncompiled) source code to build the expression graph and subsequently generating code for its derivatives. However, for advanced programming languages such as C++, this is challenging to implement. Hence, where supported by the programming language, Auto-Diff tools often rely on operator overloading. In this technique, the function to be differentiated is called with a special Auto-Diff scalar type instead of standard numeric types. This scalar type has overloaded operators that ‘record’ the operations performed on it. The Auto-Diff tool then builds the expression graph from the recordings. Popular libraries in C++ that employ operator overloading are e.g. Adept [103], Adol-C [231], CppAD [12] and FADBAD++ [14]. While all these tools are based on the same approach, there are significant differences both in performance as well as in functionality [38, 103]. The performance difference usually stems from the implementation of the expression graph, i.e. how fast it can be differentiated and evaluated. In terms of functionality, especially higher order derivatives make a key difference. They are difficult to implement and thus not supported by many tools.

In this work, we are not proposing a new Auto-Diff tool but rather discuss how the existing tools can be leveraged when working with Rigid Body Dynamics and underline their benefits for efficient robotics control and optimization.

Contributions

Auto-Diff is already widely used in the mathematical optimization community - but it is only slowly gaining popularity in the robotics community. In this work, we illustrate the potential of using Auto-Diff for robotics. We demonstrate that it outperforms analytical derivatives in terms of computational complexity while preventing significant overhead in obtaining them. To employ Auto-Diff on Rigid Body Dynamics, we extend our open source modelling tool ‘RobCoGen’ [79] to be Auto-Diff compatible. The resulting framework allows for obtaining optimized derivatives of well-known RBD quantities and algorithms as used by most optimization, estimation and optimal control based algorithms.

Furthermore, rather than directly applying Auto-Diff at runtime, we perform an Auto-Diff code-generation step for the derivatives. In this step, the dynamic expression graph is converted to pure C code. This eliminates the overhead of the expression graph, which we demonstrate is crucial for good performance. Furthermore, the resulting C code is real-time capable and can be run on micro-controllers, in multi-threaded applications or even on GPUs. This allows for directly using it in hard real-time control loops and embedded platforms but also for massive sampling in data driven methods. Since RobCoGen is an open source library, we provide the community with a tested, efficient and easy to use tool for modelling, analyzing and controlling Rigid Body Systems.

This paper is an extended version of [167]. In this work, we provide additional timings and technical details on the implementation. Furthermore, we showcase two practical examples how our approach can lead to a performance leap in robotic control problems.

First, we demonstrate how Auto-Diff can be used when modelling a Rigid Body System subject to contacts. We show that Auto-Diff can produce efficient derivatives of the system dynamics of a legged robot including a contact model. We demonstrate that this leads to a significant speed-up in optimal control applications.

Our second example extends our previous example [167] on Auto-Diff for TO using Direct Multiple Shooting (DMS). While the original example showed a planning approach for collision-free trajectories for a robotic arm, we extend it to continuous online replanning. Robots often operate in uncertain, dynamically changing environments or in the presence of disturbances, requiring updates of the optimal control policy at runtime. We present a hardware experiment with a 6 DoF robot arm avoiding an unpredictably moving obstacle during a positioning task. Thanks to the low algorithm run-times achieved through the Auto-Diff generated derivative code, the re-planning can be run in closed-loop with a robotic vision system.

Related Work

Existing libraries that implement auto differentiable Rigid Body Dynamics are rare. Drake [219] supports Auto-Diff for gradient computations of dynamics. However, it relies on Eigen's [58] Auto-Diff module which cannot provide higher order derivatives and does not support code-generation for derivatives. Also, the code is not optimized for speed. Instead, it relies on dynamic data structures, which introduces significant overhead and limits the usability of the library for hard real-time, multi-threaded and embedded applications. MBSlib [239] also provides Auto-Diff support and does so in a more rigorous manner. Yet, it also relies on dynamic data structures, limiting efficiency and potential embedded and control applications. It is unclear whether MBSlib is compatible with any existing Auto-Diff code-generation framework.

There are also efforts of deriving simplified analytic derivatives of RBD [81]. However, the resulting expressions have to be implemented manually and do not necessarily fully exploit efficient RBD algorithms, leading to more complex expressions with runtime overhead. A thorough discussion on this issue is presented in Section V.2. A comparison between Symbolic and Automatic Differentiation for Rigid Body Kinematics is conducted in [56]. However, the authors do not perform a code-generation step for Auto-Diff which, as we will see later, significantly improves performance.

There is considerable research on how to use Auto-Diff to model and simulate Rigid Body systems, e.g. [57], [92], [36]. However, this work is focused on solving the RBD differential equations in order to obtain equations of motions rather than explicitly computing their derivatives. Similar approaches are also used to perform a sensitivity analysis or parametric design optimizations, as e.g. in [17]. However, this research field is only partially concerned with explicit derivatives of kinematics and dynamics algorithms. Additionally, metrics such as complexity and runtime, which are crucial for online control and estimation, are not the primary focus of this community.

In the context of our application examples, the optimal control toolkits 'ACADO' [105] and 'CasADi' [6] deserve special mentioning. The former allows users to specify optimal control problems with a symbolic syntax and solves them. The latter is a symbolic framework for automatic differentiation, which also supports code generation. Both toolkits are targeted at real-time algorithms for optimization based control and optimization. However, these tools are general purpose tools that require modelling dynamical systems using specialized syntax. Thus, they lack the infrastructure to efficiently model Rigid Body systems and automatically compute relevant RBD quantities such as contact Jacobians, joint space inertia matrices, transforms or forward / inverse dynamics and kinematics. Instead of adding such an infrastructure to a general purpose tool, we decided to augment ‘RobCoGen’, a proven tool for modelling RBD with Auto-Diff support. We then complement it with our own optimal control problem solver suite.

V.2 Rigid Body Dynamics and Kinematics

When dealing with Rigid Body Systems, we are usually interested in one or multiple of the following quantities (or elements of those): forward/inverse dynamics, forward/inverse kinematics, the transforms between joints/links and end-effector/contact Jacobians. In robotics, Rigid Body Dynamics are often expressed as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{J}_c^T(\boldsymbol{q})\lambda + \boldsymbol{S}^T\tau \tag{4.1}$$

where $\boldsymbol{q}$ represents the rigid body state in generalized coordinates, $\boldsymbol{M}$ is the Joint Space Inertia Matrix, $\boldsymbol{C}$ are the Coriolis and centripetal terms, $\boldsymbol{G}$ is the gravity term, $\boldsymbol{J}_c$ is the contact Jacobian, λ are external forces/torques, $\boldsymbol{S}$ is the selection matrix and τ represents the joint forces/torques. $\boldsymbol{S}$ maps input forces/torques to joints and is used to model underactuated systems. In case of a fully actuated system with directly driven joints, $\boldsymbol{S}$ is identity. For readability, we drop the dependency of these quantities on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ in the following.

Forward Dynamics and its Derivatives

If we want to know how a system reacts to given joint torques and external forces, we can compute the forward dynamics expressed as

$$fd(\boldsymbol{q}, \dot{\boldsymbol{q}}, \tau) = \ddot{\boldsymbol{q}} = \boldsymbol{M}^{-1}(\boldsymbol{J}_c^T\lambda + \boldsymbol{S}^T\tau - \boldsymbol{C} - \boldsymbol{G}) \ . \tag{4.2}$$

Featherstone [72] shows that computing $\boldsymbol{M}$ (without inverting it) already has a worst-case complexity $\mathcal{O}(n^3)$ where n is the number of rigid bodies, which makes naively evaluating Equation (4.2) very expensive. Therefore, he proposes several algorithms for factorizing $\boldsymbol{M}$ and its inversion. Additionally, he also propses the Articulated Rigid Body Algorithm, which computes the entire forward dynamics, Equation (4.2), and only has complexity $\mathcal{O}(n)$.

To get the derivatives of (4.2), we can naively apply the chain rule:

$$\begin{aligned}\frac{\partial(fd)}{\partial \boldsymbol{q}} &= \frac{d\boldsymbol{M}^{-1}}{d\boldsymbol{q}}\left(\boldsymbol{J}_c^T\lambda + \boldsymbol{S}^T\tau - \boldsymbol{C} - \boldsymbol{G}\right) \\ &\quad + \boldsymbol{M}^{-1}\left(\frac{d\boldsymbol{J}_c^T\lambda}{d\boldsymbol{q}} - \frac{d\boldsymbol{C}}{d\boldsymbol{q}} - \frac{d\boldsymbol{G}}{d\boldsymbol{q}}\right) \end{aligned} \tag{4.3}$$

$$\frac{\partial(fd)}{\partial \dot{\boldsymbol{q}}} = \boldsymbol{M}^{-1}\frac{d\boldsymbol{C}}{d\dot{\boldsymbol{q}}} \tag{4.4}$$

$$\frac{\partial(fd)}{\partial \tau} = \boldsymbol{M}^{-1}\boldsymbol{S}^T \tag{4.5}$$

If we were to further simplify the expression we could use the following identities

$$\frac{d\boldsymbol{M}^{-1}}{d\boldsymbol{q}} = \boldsymbol{M}^{-1}\frac{d\boldsymbol{M}}{d\boldsymbol{q}}\boldsymbol{M}^{-1} \tag{4.6}$$

$$\frac{d\boldsymbol{M}}{d\boldsymbol{q}} = \sum_{n=0}^{N}\frac{\partial \boldsymbol{J}_n^T}{\partial \boldsymbol{q}}\theta_n\boldsymbol{J}_n + \boldsymbol{J}_n^T\theta_n\frac{\partial \boldsymbol{J}_n^T}{\partial \boldsymbol{q}} \tag{4.7}$$

where n is the index of a rigid body, θ_n is its fixed inertia matrix and J_n is its state-dependent Jacobian. The identity in Equation (4.7) has been derived in [81] and similar identities are presented for $\frac{d\boldsymbol{C}}{d\boldsymbol{q}}$, $\frac{d\boldsymbol{C}}{d\dot{\boldsymbol{q}}}$ and $\frac{d\boldsymbol{G}}{d\boldsymbol{q}}$. But even without looking at these additional identities, two issues with analytical derivatives become prominent: First, the expressions are fairly large and error prone to implement. Second, there is significant amount of intermediate calculations in the forward dynamics that must be handled carefully to avoid re-computation. Thus, there are three tedious, manual steps involved in implementing analytical derivatives: (i) Careful derivation of the formulas, (ii) their correct implementation and (iii) intelligent caching of intermediate results. As we show in the experiments, Auto-Diff takes care of all three steps, alleviating the user from all manual work while still providing an equally fast or even faster implementation.

Inverse Dynamics and its Derivatives

If we want to know what joint torques are required to achieve a certain acceleration at a certain state, we can compute the inverse dynamics by solving Equation (4.1) for τ. In case we actuate all joints directly, $\boldsymbol{S}$ is an identity matrix and we get

$$id(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) = \tau = \boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{C} + \boldsymbol{G} - \boldsymbol{J}_c^T\lambda\ . \tag{4.8}$$

For the inverse dynamics computation, Featherstone presents an efficient algorithm, the Recursive Newton-Euler Algorithm [72], which again has complexity $\mathcal{O}(n)$ and avoids computing $\boldsymbol{M}$ and other elements explicitly.

For the fully actuated case, the inverse dynamics derivatives are defined as

$$\frac{\partial(id)}{\partial \boldsymbol{q}} = \frac{d\boldsymbol{M}}{d\boldsymbol{q}}\ddot{\boldsymbol{q}} + \frac{d\boldsymbol{C}}{d\boldsymbol{q}} + \frac{d\boldsymbol{G}}{d\boldsymbol{q}} - \frac{d\boldsymbol{J}_c^T}{d\boldsymbol{q}}\lambda \tag{4.9}$$

$$\frac{\partial id}{\partial \dot{\boldsymbol{q}}} = \frac{d\boldsymbol{C}}{d\dot{\boldsymbol{q}}} \tag{4.10}$$

$$\frac{\partial id}{\partial \ddot{\boldsymbol{q}}} = \boldsymbol{M} \tag{4.11}$$

If we were to analytically simplify these equations further, we could again use the identity in Equation (4.7) and other identities presented in [81].

In case of an under-actuated robot, such as floating base robots, we can obtain the inverse dynamics by using an (inertia-weighted) pseudo-inverse of $\boldsymbol{S}$ to solve Equation (4.1) for τ as described in [172]. This is only one possible choice for computing floating base inverse dynamics [200]. However, it will serve as an example and the following discussion extends to the other choices as well. The inertia-weighted pseudo-inverse is defined as $\bar{\boldsymbol{S}} = \boldsymbol{M}^{-1}\boldsymbol{S}^T(\boldsymbol{S}\boldsymbol{M}^{-1}\boldsymbol{S}^T)^{-1}$ and leads to the inverse dynamics expression

$$id(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) = \tau = (\boldsymbol{S}\boldsymbol{M}^{-1}\boldsymbol{S}^T)^{-1}\ddot{\boldsymbol{q}} + \bar{\boldsymbol{S}}^T(\boldsymbol{C} + \boldsymbol{G} - \boldsymbol{J}_c^T\lambda) \quad . \tag{4.12}$$

The derivatives of the inverse dynamics then become

$$\begin{aligned} \frac{\partial(id)}{\partial \boldsymbol{q}} &= \frac{d(\boldsymbol{S}\boldsymbol{M}^{-1}\boldsymbol{S}^T)^{-1}}{d\boldsymbol{q}}\ddot{\boldsymbol{q}} + \frac{d\bar{\boldsymbol{S}}^T}{d\boldsymbol{q}}(\boldsymbol{C} + \boldsymbol{G} - \boldsymbol{J}_c^T\lambda) \\ &\quad + \bar{\boldsymbol{S}}^T\left(\frac{d\boldsymbol{C}}{d\boldsymbol{q}} + \frac{d\boldsymbol{G}}{d\boldsymbol{q}} - \frac{d\boldsymbol{J}_c^T}{d\boldsymbol{q}}\right) \end{aligned} \tag{4.13}$$

$$\frac{\partial(id)}{\partial \dot{\boldsymbol{q}}} = \bar{\boldsymbol{S}}^T\frac{d\boldsymbol{C}}{d\dot{\boldsymbol{q}}} \tag{4.14}$$

$$\frac{\partial(id)}{\partial \ddot{\boldsymbol{q}}} = (\boldsymbol{S}\boldsymbol{M}^{-1}\boldsymbol{S}^T)^{-1} \quad . \tag{4.15}$$

Similar to the forward dynamics case, we see that the derivative expressions become large, and their implementation is similarly error prone. Additionally, without careful optimization, we might accidentally introduce significant overhead in the computation, e.g. by not caching intermediate results.

Higher-Order Derivatives

In Subsections V.2 and V.2 we presented the first-order derivatives of the forward and inverse dynamics. However, for some optimal control algorithms such as Differential Dynamic Programming (DDP), we might need second order derivatives of the dynamics in case we want to include them in the system dynamics and/or the cost functions. If we want to further analytically differentiate the derivatives, we have to apply the chain rule to them, resulting in even larger expressions. At this point it becomes highly questionable if analytic derivatives should be implemented manually. Since symbolic computation engines also apply the chain rule and only have limited simplification capabilities, these approaches will face the same issues for second order derivatives. This might also be part of the reason why second order optimal control algorithms, such as DDP, are only rarely applied to more complex robotic systems. Instead, algorithms that approximate the second order derivatives are usually preferred [211]. When using Auto-Diff, we can easily obtain second order derivatives and hence do not face this limitation.

V.3 Automatic Differentiation with RobCoGen

The Robotics Code Generator

The Robotics Code Generator (RobCoGen) is a computer program that, given the description of an articulated robot, generates optimized code for its kinematics and dynamics [79]. A simple file format is available to provide the description (model) of the robot. Currently, RobCoGen generates both C++ and Matlab code, implements coordinate transforms, geometric Jacobians and state-of-the-art algorithms for forward and inverse dynamics, as described in [72].

Derivatives for RobCoGen

RobCoGen does not natively support any code generation for derivatives of rigid body kinematics or dynamics. However, it can be easily extended to generate C++ code suitable for Automatic Differentiation in order to leverage other tools specifically designed for that purpose, such as CppAD. The changes required to support Auto-Diff essentially reduce to generating code templated on the scalar type (rather than using standard `float` or `double`). This simple change enables the use of a variety of Auto-Diff tools based on operator overloading, yet does not prevent the regular usage of the generated code.

It is important to note that RobCoGen uses code-generation to create robot specific RBD code. This is not to be confused with Auto-Diff codegen, which uses RBD code to compute a derivative function, which is then translated into source code. In this work, we apply Auto-Diff codegen to the output of RobCoGen. Thus, there are two sequential, entirely unrelated code-generation steps involved in this work. First, RobCoGen generates the dynamics and kinematics functions. Then Auto-Diff codegen uses those to create the derivative source code. We will always indicate which type of generated code we are referring to, i.e. whether we refer to the dynamics/kinematics generated by RobCoGen or to the derivatives generated using Auto-Diff codegen[1].

---

[1] While the implementation of both code generation steps differs significantly, the goal is the same. In both cases, specialized code is generated instead of using general dynamic data structures. This eliminates the overhead of transversing such data structures, checking for dimensions, sizes and branches. Furthermore, static data structures are easier to implement

Since RobCoGen implements the most efficient algorithms for dynamics (e.g. the Articulated Rigid Body algorithm for forward dynamics), and since the lowest achievable complexity of Auto-Diff derivatives is proportional to the complexity of the original function [91], it follows that, at least theoretically, we can obtain the most efficient derivatives from RobCoGen as well.

Implementation in RobCoGen

Since the interoperability with Auto-Diff tools is ongoing development within the RobCoGen project, we implemented a proof of concept of the approach by modifying existing RobCoGen generated code for the quadruped robot HyQ [205] and a 6 DoF robotic arm.

Our goal is to make the entire RobCoGen generated code auto-differentiable. Thus, we add a scalar template to all algorithms which compute one of the following quantities:

- Forward dynamics: Articulated Rigid Body Algorithm
- Inverse dynamics: Recursive Newton-Euler Algorithm
- Homogeneous-coordinate transforms
- Coordinate transform for spatial vectors
- Jacobians
- Rigid Body quantities such as $\boldsymbol{M}$, $\boldsymbol{C}$ and $\boldsymbol{G}$

We further template all input and parameter types, including state, joint torques, external forces, all inertia parameters etc. Therefore, also uncommon derivatives, e.g. derivatives of inverse dynamics with respect to inertia parameters, can be obtained. This feature can be useful for design optimization or parameter estimation applications. Also, Auto-Diff derivatives of essential Rigid Body quantities such as the Joint Space Inertia Matrix $\boldsymbol{M}$ can be computed. This allows to use Auto-Diff to generate custom derivatives for special applications, e.g. when using Pseudo-Inverse based inverse dynamics as shown in Subsection V.2. In such a case, the derivatives in Equations (4.13) - (4.15), can still be obtained from Equation (4.12) using Auto-Diff. The user only has to ensure that the computation of the Pseudo-Inverse is auto-differentiable as well. Furthermore, the user can also generate derivatives of transforms and Jacobians, e.g. for task space control or kinematic planning.

By templating the entire library, the user can specify which methods or quantities to differentiate. Therefore, the differentiation can be tailored to a specific use case, e.g. by computing only required parts of a Jacobian or applying the 'Cheap Gradient Principle' [91]. Another benefit is that not all derivatives are generated but only the ones required. To demonstrate how to select and generate derivatives, we provide the reference implementation described below.

---

on embedded systems and to use in hard real-time applications.

Reference Implementation of Automatic Differentiation

In order to validate the performance and accuracy of the Auto-Diff compatible version of RobCoGen, we provide a reference implementation. For this implementation, we use CppAD as our Auto-Diff tool. CppAD is very mature, well documented, supports higher derivatives and provides an efficient implementation. Still, evaluating the expression graph for computing derivatives comes at an overhead. Therefore, we also employ CppADCodeGen [132], which can generate bare C code for derivatives using CppAD.

Since CppAD operates on scalars rather than full matrices, the generated derivative code is only using scalar expressions and cannot leverage advanced compiler optimizations such as vectorization. While this leads to a slight decrease in performance, the generated code is dependency-free and can easily be run on embedded systems or GPUs. Additionally, memory for intermediate results of the derivative computation can be allocated statically. Thus, the generated code can be run in hard real-time control loops. Despite the fact that the generated code is static, it can be parametrized. Therefore, RBD parameters such as mass, Inertia or Center of Mass can be changed at runtime if required. Thus, the generated code is specialized to a certain morphology rather than a specific robot or a set of dynamic properties.

V.4 Results

To evaluate the performance of our Auto-Diff approach, we first run standalone synthetic tests on the derivatives in terms of complexity and accuracy. These tests are performed on a model of the underactuated quadruped HyQ [205], which has a floating base and three joints per leg. We then present two application examples. The first example demonstrates Auto-Diff for motion planning on HyQ. The second one demonstrates online TO applied to a 6 degree of freedom arm and includes hardware experiments.

Accuracy, Number of Instructions and Timings

In order to compare the different derivative methods, we look at the derivative of the forward dynamics with respect to the joint forces/torques as shown in Equation (4.5). To ensure that we are computing a dense derivative, we ignore the floating base part of the equation but focus on the bottom right corner of the expression. For the comparison, the derivatives are obtained from different implementations: Single-sided numerical derivatives, Auto-Diff at runtime, Auto-Diff codegen and analytical derivatives. For the analytical derivatives we have two implementations. One computes $\boldsymbol{M}$ using Featherstone's Composite Rigid Body algorithm and later naively inverts it by using Eigen's general $\boldsymbol{L}\boldsymbol{L}^T$ solver. The second analytical implementation uses the RBD-specific $\boldsymbol{L}^T\boldsymbol{L}$ factorization of $\boldsymbol{M}^{-1}$, which exploits sparsity. Such a factorization is described in [72] and implemented by RobCoGen.

We first verify the accuracy by measuring the norm of the difference between the derivative matrices. As expected, the analytical methods and Auto-Diff agree

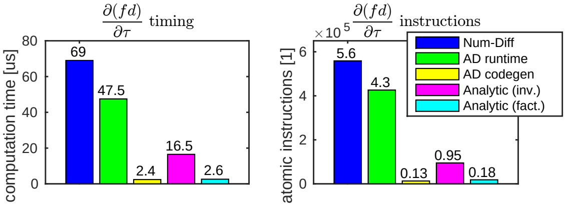


Figure 4.1: Comparison between different approaches to compute the forward dynamics derivative with respect to joint torques τ. We measure absolute computation time as well as the number of atomic instructions of the compiled code. Clearly, Num-Diff performs worst. Auto-Diff at runtime performs slightly better. However, Auto-Diff codegen and the analytic factorization perform best. The naive analytic inverse performs significantly worse than both. This underlines that naively implemented analytical derivatives can be significantly inferior to Auto-Diff.

up to machine precision ($< 10^{-13}$). Numerical differentiation however deviates by around $< 10^{-6}$ from all other implementations.

Additionally, we compare the runtime as well as the number of atomic instructions of all derivative methods. In this test, we average over 10,000 computations and enable the highest optimization level of our compiler. The results of this test are shown in Figure 4.1. This test shows two interesting findings: Firstly, there is a very significant difference between the Auto-Diff generated derivative code and the Auto-Diff computation at runtime. The generated code is much faster, which is possibly a result of removing the overhead of travelling through the expression graph and enabling the compiler to optimize the code. Secondly, we see that – despite the fact that the RobCoGen's factorization of $\boldsymbol{M}^{-1}$ exploits the sparsity and structure of the problem – it is not able to outperform the Auto-Diff generated derivative in terms of instructions and runtime. The factorization is still about 10% slower than Auto-Diff codegen. This strongly underlines the hypothesis that even carefully implemented analytical derivatives are usually equally or more expensive than those generated by Auto-Diff. On the other hand, naively implemented derivatives, even though analytic, can lead to significant overhead. This overhead, as present in the analytical inversion, is over 600% in the test above.

Finally, we test more complex expressions. We compare the derivatives of the forward dynamics as well as of the fully-actuated inverse dynamics both taken w.r.t. the state, as shown in Equations (4.3)-(4.5) and Equations (4.9)-(4.11) respectively. Also here, we are not using the naive implementations but efficient Featherstone algorithms for all derivative approaches. Additionally, we time the computation of the end-effector position $\boldsymbol{p}$ and velocity $\dot{\boldsymbol{p}}$ differentiated with respect to the full state, $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. Here, $\boldsymbol{p}$ and $\dot{\boldsymbol{p}}$ are expressed in a fixed inertial frame ('world' frame).

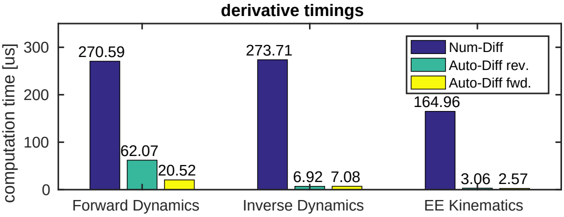


Figure 4.2: Comparison between Numeric Differentiation ('Num-Diff') and Automatic Differentiation codegen ('Auto-Diff') using reverse ('rev') and forward ('fwd') mode for the computation of three types of derivatives on HyQ. For the forward dynamics derivatives we implement Equations (4.3)-(4.5) while in the inverse dynamics derivatives we use Equations (4.9) -(4.11). Additionally, end-effector position and velocity given in Equation (4.16) differentiated with respect to the rigid body state $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ are timed. We can see that Auto-Diff codegen outperforms Num-Diff by a factor of about 10-40x.

The forward kinematics to be differentiated are given by

$$\boldsymbol{p}(\boldsymbol{q}) = \boldsymbol{T}_{ee}(\boldsymbol{q})\boldsymbol{q}, \ \dot{\boldsymbol{p}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{J}_{ee}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} \tag{4.16}$$

where $\boldsymbol{T}_{ee}$ and $\boldsymbol{J}_{ee}$ are end-effector transform and Jacobian respectively. We compute the forward kinematics derivatives for all four feet of the quadruped. Since we do not have optimized analytical derivatives available for this test, we compare the performance of Num-Diff to the one of Auto-Diff codegen. In Auto-Diferentiation, there are two internal modes of building the expression graph: Forward and reverse. While both lead to the same derivatives with the same accuracy, the graph structure might be simpler in one than in the other. Therefore, we include timings for both modes.

Results of this test are shown in Figure 4.2 which plots the average of 10,000 executions of each approach. There are several interesting observations to be made. First, Auto-Diff codegen is about 10-40x faster than Num-Diff. Secondly, Num-Diff timings for forward dynamics and inverse dynamics are of similar magnitude due to the fact that the derivatives are of similar dimensionality and similarly complex to compute. However, Auto-Diff can generate a simpler structure for inverse dynamics than for forward dynamics, resulting in significantly shorter timings. Lastly, reverse mode is supposed to be faster than forward mode for functions with (significantly) more input ('independent') than output ('dependent') variables. The ratios between input and output dimension on HyQ are 48:18 for the forward dynamics, 54:18 for the inverse dynamics and 36:24 for the forward kinematics of all legs. Despite the input dimension being larger than the output dimension for all three derivatives, we only see a slight benefit for the reverse mode in the inverse dynamics where the ratio is the largest.

Trajectory Optimization for a Quadrupedal Robot using Sequential Linear Quadratic Optimal Control

To verify that the previously shown results make a difference in pratical application, we are first looking at a TO problem for the quadrupedal robot HyQ. HyQ is a 18-DoF floating-base, underactuated robot, required to make and break contacts with the environment. Hence, planning motions for such a system is a high-dimensional problem. Such problems are often tackled using TO. Amongst others, TO problems can be transformed into Nonlinear Programs (NLPs), e.g. as shown in Subsection V.4, or formulated as Differential Dynamic Programming (DDP) problems. In this example, we show how Auto-Diff codegen can improve performance of such a DDP approach, namely Sequential Linear Quadratic (SLQ) optimal control.

Sequential Linear Quadratic Optimal Control

SLQ is an iterative optimal control algorithm which consists of three main steps. First, the system dynamics are forward integrated using the updated controller from a previous iteration or, at the first iteration, using a stable initial controller. The non-linear optimal control problem is then approximated by a linear quadratic optimal control problem by linearizing the system dynamics and quadratically approximating the cost around the forward simulated trajectory. Finally, the linear quadratic problem is solved in a backward pass using a Riccati approach. For a full description of the algorithm, we refer to [211].

SLQ Formulation for Legged Systems

In our SLQ problem, we include the entire state of the robot

$$\boldsymbol{x} = [{}_W\boldsymbol{q} \;\; {}_B\dot{\boldsymbol{q}}]^T = [{}_W\Omega_B \;\; {}_W\boldsymbol{x}_B \;\; \theta \;\; {}_B\omega_B \;\; {}_B\boldsymbol{v}_B \;\; \dot{\theta}]^T \tag{4.17}$$

where ${}_\Omega\boldsymbol{q}_B$ and ${}_W\boldsymbol{x}_B$ define base orientation and position expressed in the inertial ('world') frame. ${}_B\omega_B$ and ${}_B\boldsymbol{v}_B$ represent local angular velocity and linear acceleration expressed in a body fixed frame. Joint angles and velocities are represented by θ and $\dot{\theta}$, respectively.

The system dynamics are then defined as

$$\dot{\boldsymbol{x}} = [{}_W\dot{\boldsymbol{q}} \;\; {}_B\ddot{\boldsymbol{q}}]^T = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = [H_{WB}({}_B\dot{\boldsymbol{q}}) \;\; fd(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})]^T \tag{4.18}$$

where we set the control input equal to the joint torques $\boldsymbol{u} = \tau$. H_{WB} transforms velocities expressed in the body frame to the inertial frame. $fd(\cdot)$ represents the forward dynamics as shown in Equation (4.2).

As mentioned before, HyQ is subject to contacts with the environment creating contact constraints and forces. Contacts can be included in SLQ either as constraints [166] or using an explicit contact model [168]. Here, we choose the latter approach. Assuming a static environment, the contact forces become a function of the current state of the robot, thus $\lambda = \lambda(\boldsymbol{q}, \dot{\boldsymbol{q}})$. In this example, we employ a 'soft' contact model consisting of a non-linear spring in surface-normal direction and a non-linear damper for velocities. For each end-effector we compute the contact model in the contact frame C as follows:

$$ {}_{C}\lambda(\boldsymbol{q}, \dot{\boldsymbol{q}}) = -k \exp(\alpha_k \ {}_{C}p_z(\boldsymbol{q})) - d \ \mathrm{sig}(\alpha_d \ {}_{C}p_z(\boldsymbol{q})) \ {}_{C}\dot{\boldsymbol{p}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \tag{4.19} $$

where k and d define spring and damper constants. Since SLQ requires smooth derivatives [166, 168], we multiply the damper with the sigmoid of the normal component $p_z(\boldsymbol{q})$ of the contact surface penetration $\boldsymbol{p}(\boldsymbol{q})$ as defined in Equation (4.16). While both the exponential and the sigmoid serve as smoothing functions, their 'sharpness' is controlled with α_k and α_d. Finally, the contact force is rotated into the body frame

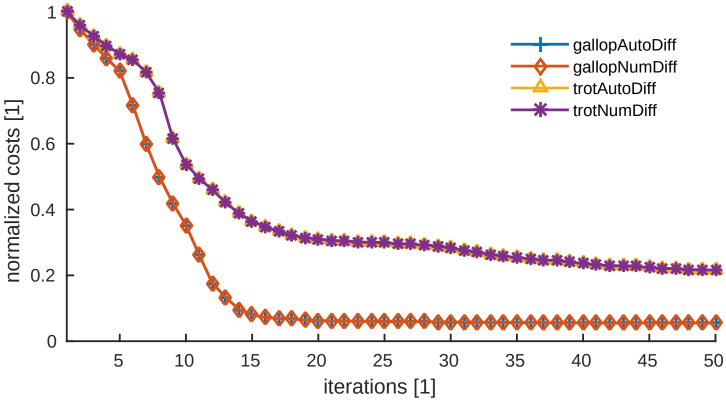
$$ {}_{B}\lambda(\boldsymbol{q}, \dot{\boldsymbol{q}}) = R_{WB}(\boldsymbol{q})_{C}\lambda(\boldsymbol{q}, \dot{\boldsymbol{q}}) \tag{4.20} $$

before it is passed to the forward dynamics. When plugging Equations (4.19)+(4.20) into the forward dynamics in Equation (4.2), we can see that its derivatives in Equation (4.3) become even more complex. Motivated by this, we apply Auto-Diff when linearizing the overall system dynamics in Equation (4.18) within SLQ.

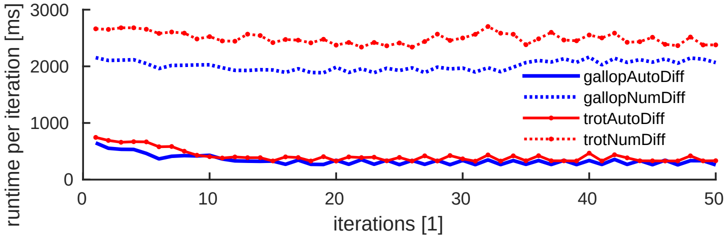
SLQ examples

To test the performance influence of Auto-Diff codegen on SLQ, we define two test tasks. In each of the tasks, we provide a cost function to SLQ that penalizes the deviation from a desired final state and regularizes deviations from a nominal state. By varying the cost function parameters, we ask for forward motions of different length. In the first example, we request a forward motion of 1.5 m while in the second example we ask for 2 m. Since the time horizon is kept constant, task 1 leads to a trotting gait while task 2 leads to a galloping gait. These gaits are automatically discovered as described in [168]. Both resulting motions are shown in the video attachment. All source code as well as the model, solver settings and cost function weights are available within our open source 'Robotics and Optimal Control Toolbox' [173].

For both tasks we compare SLQ with Auto-Diff codegen and with Num-Diff for the computation of the linear system approximation. Figure 4.3(a) shows that in both tasks Auto-Diff codegen and Num-Diff converge equally fast. While Num-Diff lacks precision, it does not impair convergence performance. However, when looking at the runtimes in Figure 4.3(b), the advantage of Auto-Diff codegen becomes obvious. Auto-Diff codegen allows for a speedup of up to 500% of the overall runtime. Note that system linearization is only one part of the SLQ algorithm. And while we do not achieve an overall speedup of around 10-40x as in the timing examples in Subsection V.4, the runtime reduction is still significant. Given the fact that SLQ requires around 15 to 50 iterations until convergence for the given tasks, this results in an absolute runtime speedup of over a minute.



(a) Learning rates for the SLQ example on HyQ. Both Num-Diff and Auto-Diff codegen exhibit the same rates when learning the same task. While not clearly visible, lines corresponding to the tasks overlap.



(b) Runtime for SLQ iterations for different tasks and derivative types. Even though the linearization is only one part of the entire SLQ algorithm, using Auto-Diff codegen speeds up the overall algorithm by up to 500%.

Figure 4.3: Speed and convergence tests for SLQ on two tasks for the quadrupedal robot HyQ. All timings were measured on an Intel Core i7 (2.8 GHz) laptop computer.

Trajectory Optimization and fast trajectory re-planning for a 6-DoF robot arm using Direct Multiple Shooting

In [167], we presented a TO example with a fixed-base robot arm avoiding a static, spherical obstacle in a positioning task. Here, we extend this example as follows: we assume that the obstacle can move in an unpredictable way – to compensate for this, we perform dynamic trajectory re-planning. Thanks to the fast, generated derivative code, we achieve re-planning frequencies that are high enough to be run in closed-loop with a visual obstacle state estimator. We show hardware experiments on an industrial robot arm and demonstrate dynamic obstacle avoidance in a go-to task.

Direct Multiple Shooting

The Direct Multiple Shooting method [23] is a widely-spread approach for numerically solving optimal control problems. An originally infinite-dimensional optimal control problem is transformed into an NLP by discretizing it into a finite set of state and control decision variables. Those are used for forward integrating the so-called 'shots', which are matched at the nodes using continuity constraints. DMS can handle inequality path constraints, e.g. task space obstacles, or control input constraints. A detailed description of the method is beyond the scope of this paper and we refer the interested reader to [53] for an overview.

We have implemented a custom DMS problem generator which hands over the resulting Nonlinear Program to off-the-shelf NLP solvers such as IPOPT [230] and SNOPT [88]. Amongst others, it achieves efficiency through exploiting the inherent block-sparse structure of the DMS constraint Jacobian. We are using a standard fourth order Runge-Kutta (RK4) integration scheme for propagating the shots, and its analytic derivative for calculating the trajectory sensitivities w.r.t. neighbouring decision variables on the fly. The sensitivities are functions of the system dynamics derivatives w.r.t. state and control at every integration sub-step in time, which can be evaluated efficiently using the Auto-Diff generated derivatives.

Robot go-to task with a randomly moving obstacle

Figure 4.4 shows a sketch of the problem setup. A fixed-base robot arm has to reach a desired joint configuration within a given time horizon T while avoiding a randomly moving spherical task space obstacle. In this experiment, we employ an ABB IRB4600 industrial robot arm with 2.55 m reach. The arm provides high-accuracy joint encoder readings (θ_m), which allow to get reasonable joint velocity estimates ($\dot{\theta}_m$) by finite-differences. The arm is controlled through an interface which allows to set joint reference positions (θ_{ref}) and velocities ($\dot{\theta}_{ref}$) at 250 Hz rate. Despite being bound to a kinematic controller, planning dynamically optimal trajectories is still an advantage in terms of energy efficiency [179] and providing smooth, feasible references.

The moving obstacle in the robot's workspace is realized using a fiducial marker on a stick which is guided by a human. The obstacle is defined as a sphere with

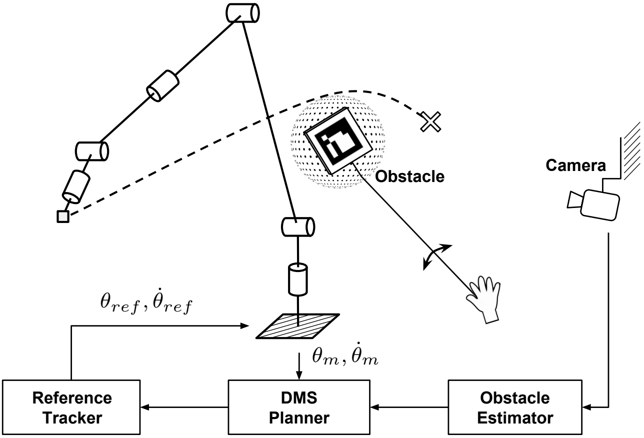

Obstacle
Camera
$\theta_{ref}, \dot{\theta}_{ref}$
$\theta_m, \dot{\theta}_m$
Reference Tracker
DMS Planner
Obstacle Estimator

Figure 4.4: The experimental setup: a fixed-base robot arm has to reach a desired joint configuration within a given time horizon T while avoiding a randomly moving obstacle.

0.5 m diameter around the marker's center. The marker is tracked by a camera system and an off-the-shelf visual state estimator provides position and -velocity estimates for the obstacle at 20 Hz rate. The trajectory of the obstacle is predicted through forward integration of its current velocity estimate. For obstacle collision avoidance, we define a grid of collision points distributed equally-spaced on the robot links. For each of those collision points, an inequality path constraint results at every node of the DMS problem. The DMS planner accesses the latest robot- and obstacle state estimates, and forwards its current optimal plan to a reference tracker which sets the joint state reference for the robot arm's internal joint controllers.

In order to deal with the changing obstacle, the planner needs to continuously re-compute optimal trajectories which drive the robot to the desired goal. In the literature, methods exist for nonlinear model predictive control or fast re-planning using DMS, for example the approach presented in [52], which updates the applied control input at every major SQP-iteration. In this work, by contrast, we run the NLP solver until a sufficient degree of convergence is met to update the whole trajectory at once. Re-planning starts immediately after the initial problem is solved. In order to ensure a smooth trajectory, the initial state for re-planning is interpolated from the last reference trajectory using the estimated re-planning time. The convergence of the NLP solver is sped up by initializing it with the previous optimal state- and input trajectories. The previous solution gets interpolated such that the nodes are again equally distributed over the remaining time horizon. The optimizer keeps re-planning as long as a feasible solution can be found and the robot arm is a minimum distance away from the target, which avoids over-parametrization towards the trajectory end.

As initial time horizon, we select $T = 5.0$ s and discretize it into 20 equally spaced shots. We choose zero-order hold interpolation of the control inputs between the nodes and propagate the system state with a constant RK4 step-size of 50 ms. The initial guess for the state decision variables consists of zero joint velocities and equidistantly spaced joint positions that are obtained by direct interpolation between the initial and terminal joint configuration. The initial guess for the control input decision variables are the steady-state joint torques computed by the Recursive Newton Euler inverse dynamics evaluated at the corresponding states.

Results

In [167], we found that IPOPT significantly outperformed SNOPT in all our simulation experiments (similar results were obtained in [182]). We shall anticipate that we found the same to hold for the re-planning case. Furthermore, we noticed that SNOPT runtimes varied significantly with the initial- and desired arm configuration and the obstacle state while the majority of all recorded IPOPT runtimes lay in a relatively narrow band. As a consequence, and for reasons of limited space, we only focus on IPOPT in the remainder of this paper.

In the following, we highlight the performance gain for DMS with re-planning when using Auto-Diff generated derivatives of Rigid Body Dynamics instead of numerical derivatives. To assess the performance, we look at the runtime and the number of iterations that IPOPT requires for different scenarios. We investigated 20 planning and re-planning scenarios with moderately varied joint positions and different obstacle movements. For showing the influence of the collision constraints on the runtime, we repeated all experiments without obstacle and provide the recorded timings as reference.

Figure 4.5 summarizes the results, where the planner was run on an Intel Xeon E5 processor. It is evident that independent of the scenario or planning stage, Auto-Diff codegen outperforms Num-Diff in runtime by a factor of usually 100% or more. During re-planning, we reach an average of 13-15 IPOPT iterations. In the multi-threaded setup with Auto-Diff codegen derivatives, this leads to re-planning times reliably lower than the update rate of the vision system (50 ms). This allows us to optimally exploit the obstacle estimator information by running the re-planner synchronously with the vision system and thus obtain optimal reactiveness and re-planning performance.

Figure 4.6 shows an image sequence from one of the hardware experiments. It gives an example of how the optimal trajectory gets updated as the obstacle is moved in the robot's workspace. A video of an example maneuver, in which the robot cycles between two joint configurations and reacts to the obstacle, is provided as video attachment.

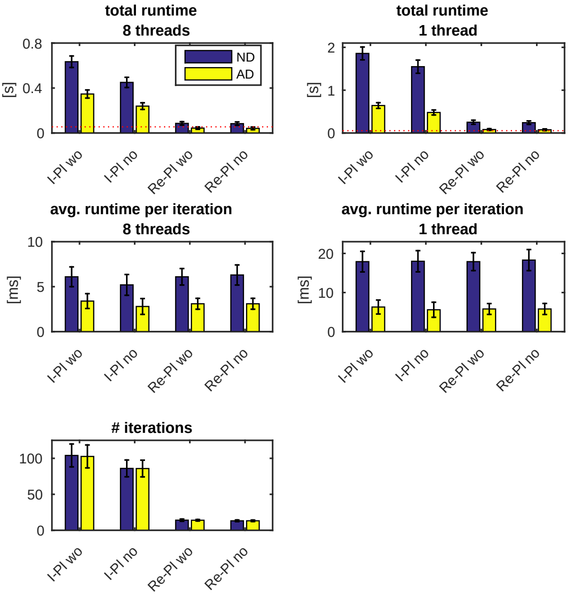
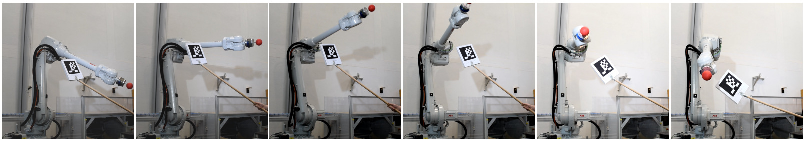
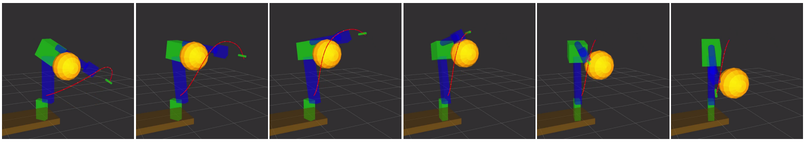


Figure 4.5: Comparison between Num-Diff ('ND', blue) and Auto-Diff codegen ('AD', yellow) in re-planning experiments. We show timings for the initial planning step ('I-Pl') and the re-planning ('Re-Pl'), for cases with obstacle ('wo') and without ('no'). These timings were recorded for different numbers of parallel threads (1/8). The plot shows average data and standard deviations from 20 experiments. The dotted, horizontal lines in the two top diagrams indicate the update time of the obstacle state estimator (50 ms). Results show that Auto-Diff codegen leads to significantly improved runtime. The relative difference is remarkable in the single-core case. Thanks to Auto-Diff codegen, it is possible to achieve re-planning times that are consistently lower than the obstacle estimator's update rate.



(a) The arm avoids a moving obstacle during a go-to task. The obstacle is defined by the fiducial marker mounted on a stick guided by a human.



(b) Visualizations of the arm, the obstacle (orange sphere) and the end-effector trajectories. The latter illustrate how the optimal plan changes over the re-planning cycles.

Figure 4.6: An image sequence of re-planning experiments on the ABB IRB4600 arm (a) and corresponding visualizations of the obstacle and the trajectories (b). All images taken equidistant in time with $\Delta t = 1$ s.

V.5 Discussion and Outlook

In this work, we propose how Auto-Diff can be efficiently applied to RBD algorithms. Based on this study we extended our open source Robotics Code Generator, enabling full Automatic Differentiation compatibility. This allows us to underline the importance of generating source code for the derivatives obtained from Auto-Diff, avoiding the usual overhead during evaluation. The resulting derivative code outperforms even carefully hand-designed and optimized analytical derivative implementations. Besides basic timings of forward and inverse dynamics derivatives, we show that Auto-Diff with generated code can significantly improve performance in control and optimization applications.

In the motion planning examples for the legged robot HyQ, Auto-Diff codegen allowed us to lower the runtime of our TO from minutes to seconds, achieving an overall speedup of 500%. By auto-differentiating the entire dynamics including the contact model, we are able to achieve fast but also accurate derivatives without manually deriving them. We believe that Auto-Diff will play an important role in running such approaches online as Model Predictive Controllers.

In our hardware experiments on a 6 DoF robot arm, Auto-Diff codegen allowed us to achieve re-planning rates fast enough to be synchronized with a typical robotic vision system. In this example, Auto-Diff provided the critical speedup to convert our optimization based motion planner to an online continuous replanning approach.

While the examples in this paper are still partially academic, they underline the potential of Auto-Diff when dealing with RBD. This motivates us to include Auto-Diff codegen in our more advanced controllers and estimators. In this work, we have already seen a significant gain from generating first order derivatives. However, our toolchain would allow us to generate second order derivatives as well. This could further speed up our control approaches which currently only approximate second order derivatives.

V.6 Source Code and Examples

The latest version of RobCoGen is available at `https://bitbucket.org/mfrigerio17/roboticscodegenerator`, where we will also release the Auto-Diff compatible version. As an example, the Auto-Diff compatible RobCoGen output for the quadruped HyQ as well as all derivative timing and accuracy tests are available at [173] and `https://bitbucket.org/adrlab/hyq_gen_ad`.

Paper VI: "Why off-the-shelf physics simulators fail in evaluating feedback controller performance - a case study for quadrupedal robots"

Authors

Michael Neunert, Thiago Boaventura, Jonas Buchli

Abstract

Simulators are essential to robotics research since they help us to understand physical properties of the robotic system as well as test control and motion planning approaches in a quick and safe manner. Yet most robotics simulators rely entirely on physics engines and do not model actuator dynamics, noise or delay. However, as we demonstrate in this work, modeling these effects is essential when evaluating the performance of closed loop controllers. In fact, neglecting them is one reason why controllers do not transfer well from simulation to hardware. While the presented study applies to most multi-link robots, our parameter choice is inspired by state-of-the-art quadrupedal robots.

Published in:
Advances in Cooperative Robotics: Proceedings of the 19th International Conference on Climbing and Walking Robots and Support Technologies for Mobile Machines (CLAWAR), World Scientific, pp. 464–472, 2016

Award:
This paper has been awarded with the *Industrial Robot Innovation Award for practical innovation in the field of robotics* at CLAWAR 2016.

VI.1 Introduction

Due to the complexity of modern robots, physics simulations have become an essential tool for robotics researchers. When looking at recent advances in simulators, great care is taken to accurately model the underlying physical behavior of rigid body dynamics [184]. Thus many simulators, both open source [124] and commercial [151, 202] rely on mature, accurate physics engines such as ODE [212] or Bullet [44]. Yet, we often experience that controllers are often not directly transferable from simulation to hardware. One reason is that robots are not pure, ideal multi-body systems. Instead their behavior is affected by actuator dynamics, sensor and system noise, delays, disturbances and other effects. While we increasingly see sensor noise models in robotics simulators and/or many provide an API/plugin architecture to include such models, actuator dynamics and delays are rarely included by default and thus often remain unmodeled. As the following study shows, these effects play an important role when assessing the performance, stability, and robustness of a closed-loop control system. While motion planning and trajectory execution usually translates well from simulation to hardware, feedback controllers and their gains usually require separate tuning on hardware. In order to understand this phenomena, we analyze the influence of actuator dynamics, noise and delays with regards to well established control theory performance and stability measures. The analysis is carried out for a conceptual one degree of freedom actuator attached to a single body, where the model parameters are inspired by actuators of state-of-the-art quadrupedal robots.

VI.2 Model Description

In order to make a transparent analysis, we study the properties of a single degree of freedom articulated rigid body model, attached to a linear or cylindrical joint. This single DoF system is assumed to be driven by a force/torque-controlled actuator, which has its own open and closed loop dynamics, and tries to track a desired force/torque, usually generated by an outer control loop. As in many state of the art systems, we use a position controller as an outer control loop. While investigating a single DoF rigid body system seems like a strong simplification, such a model is still able to capture the essential dynamics and stability limitations we consider in this paper. In addition, many state-of-the-art robots rely on low-level single input single output (SISO) joint controllers, which are usually designed for their respective single DoF. In these cases, the coupling with the overall multi-body system is usually seen and treated as a disturbance to the SISO system, and as such its influence on the closed-loop performance and stability can also be analyzed using the tools and methods presented here.

Actuator Model

In this work we consider the actuator model, depicted by a dashed box in Figure 4.7, as the closed-loop transfer function between the desired and the actual actuator

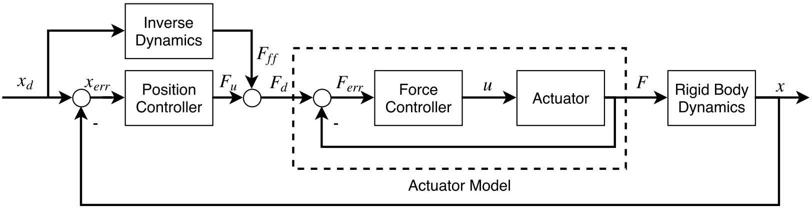


Figure 4.7: Model of a typical cascaded control structure of a single articulated rigid body. The actuator itself is driven by an inner force controller which is usually complemented by an outer position or impedance controller.

force command. In the case of an electric actuator, the model would include the controller of the motor electronics as well as the dynamics of the rotor and the gearbox. For a hydraulic actuator this would include, besides the force controller, the valve and cylinder dynamics. While not considered here, friction and other non-linear effects could be included in this model as well. For our examples, we study the behavior of actuators as found in the state-of-the-art quadrupeds HyQ [205] as well as StarlETH [106]. Based on the data presented by the respective developers, we can see that both clsed-loop dynamics can be coarsely approximated by a second order system with delay defined as $P(s) = e^{-t_d s}\frac{\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$. More precisely, based on Figures 6.11 and 7.3 in [21] as well as Figure 4.3 in [106], we approximate the parameters of the second order system for intermediate performance points as $\omega_n = 800, \zeta = 0.45$ for HyQ and $\omega_n = 110, \zeta = 0.9$ for StarlETH. Figure 4.3 in [106] shows a worst case delay of 5 ms. We decided to consider a more ideal time delay of 1 ms. Naturally, our fitted models are oversimplified and will not capture all the effects of the closed-loop force controlled system. Especially, it does not model potential internal instabilities of the force controller. However, this simple model is sufficient to identify important dynamical effects concerning control stability. In general, it is application-dependent how detailed the chosen models should be. Thus, we consider this discussion beyond the scope of this paper. The actuator model output directly acts on the rigid body dynamics. Here, we model a single DoF body with mass $m = 0.7$ kg. Based on its properties it can be imagined being a typical endeffector of a quadruped or humanoid robot. While we assume a linear joint, the findings below also qualitatively translate to rotary joints. We further assume that a position controller closes the loop around the actuator model and rigid body dynamics. It is represented as a Proportional-Derivative (PD) controller with fixed gains at $k_p = 400$ and $k_d = 20$ respectively.

Noise, Delays and other Effects

Sensor models that partially include noise, biases and delays can help to evaluate state estimation approaches. However, from a control perspective, noise can impair the control performance and depending on the sensitivity of the system it can be even amplified. Another important influence on control stability of robotic systems are delays. These delays occur due to communication between components, sampling times, computational time required by algorithms, asynchronous computation of discrete controllers or internal delays of components. In this work we assume all occurring delays are pure time delays and can be attributed to the actuator model. Based on Figure 4.3 in [106] we assume that the worst-case total internal delay accumulates to 5 ms. Apart from delay and noise, there are additional effects that can affect control stability and performance, which we intentionally neglect in this study. Amongst others, these are internal (non-linear) effects of the actuation model such as saturation, measurement quantization, discretization and sampling, aliasing, static friction, gear backlash, etc.

VI.3 Sensor-actuator effects on a robotic joint controller

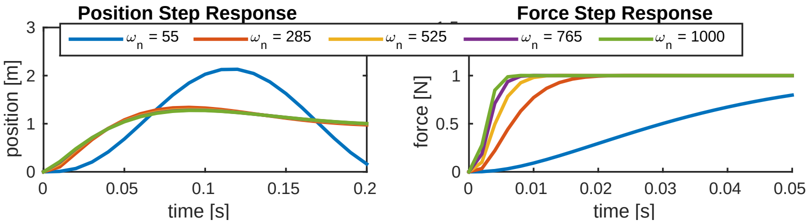


Figure 4.8: Step responses of the position control loop and the actuator dynamics for different resonance frequencies ω_n of the actuator model. Unless very slow actuator dynamics are considered, the difference in position step responses are very small, suggesting that for evaluating kinematic plans, actuator dynamics do not play a significant role.

In the following analysis we compare neglecting actuator dynamics and approximating them with a simple linear second order model as described above. To get a first impression on how actuator dynamics influence the accuracy of a simulator, we look at step responses both of our second-order actuator model with delay as well as the closed-loop position controlled system. During this analysis we vary the resonance frequency ω_n of the second-order actuator model. When looking at these step responses in Figure 4.8, we can observe that the force response significantly varies in terms of rise times for different resonance frequencies w_n. Yet, these differences appear on a relatively small time scale. When looking at the step response of our position controller, we see that for time scales comparable to the dynamics of the rigid body system, the differences between various actuator dynamics become small unless they are very slow. So if we are only interested in obtaining a good approximation of the rigid body motion as e.g. in evaluating kinematic plans, neglecting the actuator dynamics seems acceptable. Although the position step response is not deeply affected by the low-level actuation system dynamics, the stability and robustness of the closed-loop position controlled system may be significantly affected. Therefore, we will investigate how controller stability metrics are affected by the actuation dynamics. We focus on three well established control performance criteria: Gain margin, phase (delay) margin, and sensitivity.

Influence of actuator dynamics on the gain margin

The gain margin GM gives us a measure of how much we can increase our feedback gains before the system becomes unstable. It is defined as the offset of the open loop magnitude to 0 at the frequency ω_{pc} where the open loop phase crosses -180°: $GM = 1/|G(j\omega_{pc})|$. We compute the gain margin for both the closed loop system with and without actuator dynamics. The results are shown in Figure 4.9. The system without actuator dynamics is not plotted since it has infinite gain margin. This means, we could potentially have infinitely high gains without the controller becoming unstable. Of course, this fact does not hold in simulation since we observe instabilities with finite gains. This results from the discrete implementation of our controllers and system dynamics. However, this also means that we can mitigate this limitation by increasing our sampling/integration rates [216] to prevent instability of the controller. Although we can also increase sampling on real hardware or simulate the system with actuator dynamics, this will not help indefinitely in the latter cases. For instance, Figure 4.9 shows that our gain margin monotonically decreases with the dynamics of our actuator model. While the absolute numbers might not be representative, it is important to know that gain margin can drop by a factor of 3 to 5 for the chosen range. Additionally, as the right graph in Figure 4.9 shows, the gain margin is heavily influenced by delays of the actuator dynamics. Even delays in the millisecond range influence the gain margin significantly.

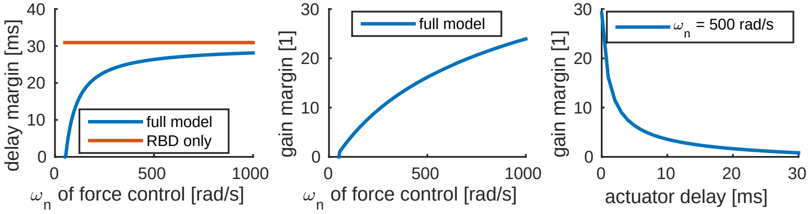


Figure 4.9: Gain and delay margin (blue) for the position loop gain for different resonance frequencies ω_n of the actuator dynamics. The reference system without actuator dynamics (red) has infinite gain margin.

Influence of actuator dynamics on the delay margin

The phase margin is a measure of how much extra phase (or pure time delay) our controller can cope with. It corresponds to the loop gain phase at the frequency that the magnitude crosses 0 db. While there might be more than one such crossing, the delay margin represents the phase margin that is closest to instability when adding delay. The delay margin is thus defined as $DM = min(180^\circ + \angle G(j\omega_{gc}))$. In Figure 4.9 we plot the delay margin for different values of our second order actuator model resonance frequency. We can observe that the delay margin asymptotically approaches the one of the model without actuator dynamics. For dynamics below 200 rad/s which we e.g. see in StarlETH, the gain margin drops significantly. Any additional delay in the system will shift the delay margin further down. This analysis reveals two interesting aspects. First, if for a given system the delay margin is already small, it is important to model delays to make sure that delays in an outer control loop (here the position controller) do not make the system unstable. Furthermore, the results also suggest that for systems with parameters in comparable range to the example, small delays in the millisecond range might lead to an unstable system. Considering that common position control loop rates often lie between 100 to 1000 Hz and occasionally run asynchronous with sensing, we can expect delays of up to 5 ms [106] and above, which can become critical for actuators with slow dynamics ($\omega_n < 200 rad/s$). Delays can become even worse when controllers are not closed on joint-level but on a whole-body-level, where complex state estimators can introduce extra delays.

Influence of actuator dynamics on the sensitivity

The sensitivity analysis tells us how variations in the process influence the behavior of the system. Using the standard definition of the sensitivity function, we can compute its peak, the nominal sensitivity peak as

$$M_s = \underset{0 \leq \omega < \infty}{max} |S(j\omega)| = \underset{0 \leq \omega < \infty}{max} \left| \frac{1}{1 + G(jw)C(jw)} \right|. \tag{4.21}$$

Since the sensitivity describes how measurement output influences the systems feedback behavior, it also tells us the influence of noise and disturbances on our system. A sensitivity greater than 1 tells us that disturbances or noise at the corresponding frequency get amplified by the system. Therefore, we focus on two aspects in the sensitivity analysis: what the nominal sensitivity peak is and at what frequency it occurs. By looking at the bode plot of the sensitivity function in Figure 4.10, we can see that the system nicely attenuates disturbances and noise in the lower extreme of the frequency spectrum. This attenuation results from the outer control loop and is not influenced by the actuator model. At the higher extreme, any disturbance is passed through by the feedback, also with no dependency on the model. Since we expect sensor noise at higher frequencies, we assume that noise rejection behaves similar with or without an actuator model. Depending on the rigid body mass and the position control parameters, the frequency at which we

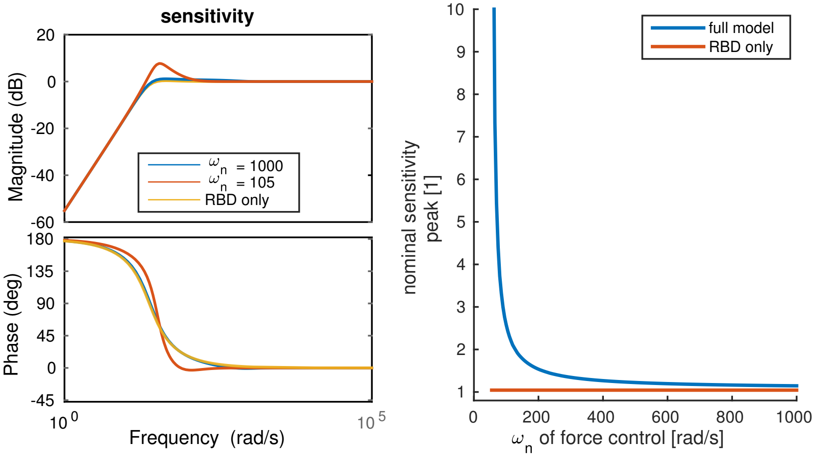


Figure 4.10: The left plot shows the sensitivity between output and feedback of the closed loop system for different values of the actuator dynamics resonance frequency ω_n as well as the actuator dynamics free simulation. The right plot shows the nominal sensitivity peak as a function of the actuator dynamics resonance frequency ω_n.

observe the sensitivity peak varies between around 1 to 100 rad/s. For the given parameters we find a sensitivity peak at a frequency around 40 rad/s. As Figure 4.10 shows, the magnitude of the peak changes with different actuation models. Slower actuator dynamics lead to a higher peak sensitivity, making the system more vulnerable to disturbances with corresponding frequency. The sensitivity analysis tells us that actuator models are less relevant to noise rejection characteristics than to disturbances. Thus, modeling actuator dynamics becomes important where we expect significant disturbances and want to verify control stability under these conditions.

Summary of effects on the control stability

We have seen that actuator dynamics and delays have a significant influence on control loop stability and sensitivity. Without modeling actuator dynamics, gain margin is infinite and thus, we might unintentionally overtune our controllers in simulation. Also, if delays are unmodeled we might oversee a lack in phase margin also rendering the controller unstable on the real hardware. While we have analyzed the influence of parameters independently, there is cross coupling between the resulting effects.

VI.4 Simulating a low-level actuation system

As the analysis above shows, we should include a model of our actuators in our simulator for control performance and robustness analysis. Hence, we implement a Gazebo plugin that simulates second-order actuator dynamics. Furthermore, we add setpoint, measurement, and output delays. Last but not least we model A/D and encoder quantization as well as sensor noise and biases. Our model of the low-level input-output and control system is evaluated at the same frequency as on hardware, ensuring consistent sampling time. Due to the simplicity of the actuator model, the simulator can still run in real-time, despite integrating 18 actuator systems at kilohertz rates. To test the improved simulator, we increase our hardware tuned, stable PD gains by a factor of 3. We then run our simulation with delays and the actuator model and once without. As shown in the video (`https://youtu.be/GNCFuhsuTkE`)), the bare simulation suggests that these gains are stable. Enabling the low-level actuation model, reveals the instability due to the over-tuning. The unaltered hardware tuned gains are stable both in the bare and in the improved simulator.

VI.5 Conclusion and Outlook

In this work, we have analyzed the importance of modeling the low-level actuation system of a robot to be able to better evaluate control performance and stability in simulation. An actuator dynamic free, noise-free simulation does not reveal gain margins, encouraging over-tuning of feedback gains. Also an actuator model can help to understand how noise and disturbances get rejected by the system. While we performed a first analysis of our improved simulator with added actuator dynamics, noise and delays, we will perform an in-depth comparison to the real system and also study the trade-off between actuator model simplicity and accuracy.

Paper VII: "An Open Source, Fiducial Based, Visual-Inertial Motion Capture System"

Authors

Michael Neunert, Michael Bloesch, Jonas Buchli

Abstract

Many robotic tasks rely on the accurate localization of moving objects within a given workspace. This information about the objects' poses and velocities are used for control, motion planning, navigation, interaction with the environment or verification. Often motion capture systems are used to obtain such a state estimate. However, these systems are often costly, limited in workspace size and not suitable for outdoor usage. Therefore, we propose a lightweight and easy to use, visual-inertial Simultaneous Localization and Mapping approach that leverages cost-efficient, paper printable artificial landmarks, so called fiducials. Results show that by fusing visual and inertial data, the system provides accurate estimates and is robust against fast motions and changing lighting conditions. Tight integration of the estimation of sensor and fiducial pose as well as extrinsics ensures accuracy, map consistency and avoids the requirement for precalibration. By providing an open source implementation and various datasets, partially with ground truth information, we enable community members to run, test, modify and extend the system either using these datasets or directly running the system on their own robotic setups.

Published in:
Proceedings of the 19th International Conference on Information Fusion (FUSION), pp. 1523–1530, 2016

VII.1 Introduction

Motivation

For many tasks in mobile robotics, it is important to estimate a robot's state with respect to its workspace, i.e. its pose and velocities expressed in an inertial coordinate system aligned to the robot's workspace. Such tasks include navigation, motion planning or manipulation. One way to measure the position and orientation of a robot is to use a motion capture system (such as e.g. Vicon [227], Optitrack [177] or PTI Visualeyez [192]). These systems are usually highly accurate and provide pose estimates w.r.t. to a calibrated reference system. However, these systems can be very costly and the user might be limited to a certain workspace size. Furthermore, many common systems such as Vicon and Optitrack use passive markers that reflect infrared light which limits their usage to indoor setups. Also, most systems require a tedious calibration procedure that needs to be repeated frequently to maintain accuracy. Since these systems do not have access to inertial data, they have to rely on finite differences of position measurements to estimate velocity information, which usually leads to highly quantized (compare Figure 4.19) or delayed data.

Another approach to state estimation is Visual Odometry (VO), which is sometimes also fused with inertial data. VO can provide very accurate local estimates of the robot motion. However, usually only a finite number of previous observations (frames) are included during the pose estimation step and no loop closure is performed. Thus, VO is prone to drift over time and does not provide a globally consistent path. Additionally, VO only provides a pose estimate to the initial pose and not to the workspace. Compared to VO, Simultaneous Localization and Mapping (SLAM) introduces the notion of a global map and therefore can ensure consistency by performing loop closure. However, map building, storing and loop-closure detection can be computationally and memory demanding.

In this work, we propose a lightweight, cost effective motion capture system based on a monocular, visual-inertial SLAM system that tightly fuses inertial measurements and observations of artificial visual landmarks, also known as "fiducials" which constitute the map. By using artificial landmarks that provide rich information, the estimation, mapping and loop closure effort is minimized. In this implementation, we use AprilTags [176] as our fiducials. Since these tags also provide a unique identification number, they can be robustly tracked and estimated in the applied Extended Kalman Filter (EKF). Additionally, loop closure is handled implicitly and no additional loop closure detection step is required. A single observation of a tag is sufficient to estimate the relative transformation between tag and robot.

Most tag based localization systems use the relative pose estimates from the tag observations. In this work, we chose a tightly coupled approach where the *corner detections* are used as observations, forming a holistic sensor fusion algorithm. Hence, the system can work with very few tags and observations while still remaining accurate. This reduces the map size of the estimation problem and lowers computational demands. Therefore, the complexity of the proposed system is much lower than common SLAM approaches while still providing accurate, globally consistent estimates. By also including inertial measurements, robust performance during fiducial occlusion, motion blur from fast motions and changing lighting conditions is ensured. The approach requires to artificially prepare the workspace but also provides relative pose information within the workspace. Hence, instead of an alternative for SLAM solutions, we see the developed system as a lightweight tool that can be used as an inexpensive outdoor-capable motion capture system, for verification of other state estimation systems or for absolute localization in a given workspace.

Related Work

Many existing fiducial-based localization systems are targeted at augmented reality or were designed to be used with cameras only. Hence, many systems use vision data only (e.g. [32, 63, 74, 176, 246]). These systems have two major drawbacks over the presented system. Firstly, they fail to provide any estimate during occlusion or motion blur. Secondly, linear velocities and body rates can only be computed based on the position and orientation estimates and are thus highly quantized. While this might be negligible for virtual reality applications, it can cause issues when closing a control loop over these estimates. To mitigate these issues, a motion model can be assumed [135]. However, this makes the approach specific to the implemented motion model.

The motion estimation and map building elements of the presented approach are closely related to monocular, visual-inertial SLAM, which has proven to be very effective [111, 119, 134, 159, 175]. The difference between our approach and fiducial-free visual-inertial SLAM solutions is that we tightly integrate artificial landmarks that result in highly robust and unique features in image space. As a result, our landmarks can be robustly redetected and their detections are almost outlier free which increases the robustness of the approach. Additionally, each landmark has a 6-DoF pose (position and orientation) rather than only 3-DoF as in commonly used point landmarks. Furthermore, since a single measurements fully constrains the 6-DoF relative pose, a single landmark is sufficient for estimating the pose. This also allows for a simple yet accurate landmark initialization which usually is a problem in monocular SLAM approaches [213]. Furthermore, pose landmarks allow for aligning the map and the estimates to a given frame in the workspace and thus, the system can provide an absolute localization in the workspace which can be crucial for tasks that assume a prepared environment.

While both, fiducial-based localization and SLAM are well studied problems, not many approaches exist that combine both. One approach where fiducials are combined with SLAM is presented in [140]. However, the inertial measurements are not used to estimate velocities but only used for a fall-back pose estimation if all fiducials are occluded. Another similar system as the one presented in this work has been described in [78]. Since this work is part of the development of a commercial product (InterSense IS-1200) the authors remain relatively vague about their sensor fusion algorithm as well as the achievable performance of their system. Furthermore, dedicated hardware is required which poses additional costs for the user and contradicts the goals of this project to provide a cost-efficient, open source framework. A third visual-inertial, fiducial based localization system is presented in [243]. While also here inertial measurements and visual data are fused in an EKF, the approach does not include measurements from an accelerometer which can be helpful during fast linear motions and can provide a notion of gravity. Additionally, it is assumed that the poses of the tags are perfectly known in a workspace frame. Therefore, one can only place the tags in known configuration and imperfect calibration will lead to an inconsistent map. In [34] a fiducial-based SLAM approach is presented. However, here the fiducials are only represented as point features and thus only the 3D positions of the fiducials are estimated.

Contributions

We present a lightweight motion estimation system based on monocular visual-inertial EKF-SLAM using artificial landmarks. This work tightly couples two proven concepts, SLAM and fiducial-based localization, by using corner observations of 6 DoF landmarks and adapting the corresponding Kalman filter innovation term. This allows for smaller map sizes and leaner estimation. In contrast to existing approaches relying on 6 DoF fiducials, the presented framework processes visual measurements and inertial data in a single estimator which results in consistent data and avoids precalibration and recalibration. We have developed this tool out of a need for an open source, lightweight, accurate visual-inertial motion capture system. We provide the system as free to use open source software. Since it only relies on standard hardware (an IMU and a camera) and a Robot Operating System (ROS) software interface, both often available on robotic platforms, it can be deployed easily. The source code, the datasets as well as a more detailed technical manual can be found at `https://bitbucket.org/adrlab/rcars`, allowing easy integration and full reproducibility of the presented results.

Notation and Conventions

In the following sections, scalars are indicated with small letters (e.g. f_x). Vectors are indicated with small, bold letters (e.g. $\boldsymbol{r}$). Matrices are indicated by non-bold capital letters (e.g. K). A capital subscript leading a variable name describes the coordinate frame that the quantity is expressed in. Position vectors are denoted by $\boldsymbol{r}$. The trailing subscript describes the direction of the vector from its origin to its goal position (read from left to right), e.g. ${}_A\boldsymbol{r}_{QP}$ is a position vector expressed in frame A that points from point Q to point P. Quaternions are denoted by $\boldsymbol{q}$. The trailing subscript denotes the coordinate systems involved in the passive rotation, e.g. $\boldsymbol{q}_{AB}$ represents the passive rotation from the coordinate system B to the coordinate system A. Hence, to rotate a position vector expressed in B to A, we would compute ${}_A\boldsymbol{r}_{QP} = \boldsymbol{q}_{AB}({}_B\boldsymbol{r}_{QP})$.

VII.2 System Description

The present localization system consists of two main components, a detector for the fiducials and an EKF for sensor fusion. In a first step, the image acquired by the camera is undistorted. Afterwards, the detector is run on the image which outputs the corner coordinates as well as a unique identifier number (id) associated with each detected tag. Furthermore, it estimates the relative transformation between each tag and the camera. This estimation is based on an iterative optimization minimizing the reprojection errors between the projected 3D corner points and their detections in image space. In a second step, the EKF uses the information from re-detected tag corners to estimate the robot's state, including pose, linear velocity and body rates. Additionally, the filter continuously estimates the position and orientation of the tags with respect to the camera coordinate frame. When a tag is seen for the first time, its pose is initialized using the relative transformation between the camera and the tag as provided by the detector. After this initialization, the tag pose will be refined within the EKF by using the reprojection errors of its corners in each subsequent re-observation. To ensure consistency, the extrinsic calibration between camera and IMU as well as the additive IMU biases are also included in the filter state.

Fiducials

Over the past years, a large variety of fiducial systems have been developed. Very popular implementations include ARToolKit [117], ARTag [74], CyberCode [198] and multiring color fiducials [41]. In our implementation, we use AprilTags [176] which are 2-dimensional, printable bar codes. The reason for this choice was the achievable high accuracy [176] and the numerous available detector implementations in C/C++. In our system, we use the detector implemented in cv2cg[2]. In our evaluations, this implementation has proven to be fast and providing accurate and robust tag detections.

Hardware

The proposed system requires a camera and an IMU. While the transformation between IMU and camera can be estimated online, the camera intrinsics have to be given a-priori. In our setup, we are using a Skybotix VI-Sensor [174]. This sensor consists of two cameras in a stereo configuration and an IMU. While the sensor is a stereo camera we are only relying on the left camera in this work. The sensor is set up to output images at 20 Hz and IMU data at 200 Hz.

Camera Model

In this project, we assume a pinhole camera model which is applicable to most cameras with common field-of-views. The expected input for the detector and filter

---

[2] http://code.google.com/p/cv2cg/

is an undistorted image. Therefore, the user is free to choose a distortion model as long as an undistorted image is provided. In the case of the VI-Sensor we are using a radial tangential distortion model. The pinhole camera model is represented by the overall projection π which depends on the camera intrinsics, i.e., the focal lengths, f_x and f_y, and the camera's principle point $\boldsymbol{c} = (c_x, c_y)$. It maps a 3D point P expressed in the camera coordinate frame, ${}_V\boldsymbol{r}_{VP}$, to its corresponding pixel coordinates $\boldsymbol{p} = \pi({}_V\boldsymbol{r}_{VP})$.

Filter

In order to fuse the information gained from the observed tags together with the on-board inertial measurement we implement an extended Kalman filter. Relying on appropriate sensor models, this filter uses the inertial measurements in order to propagate the robot's state and performs an update step based on the available tag corner measurements. In the following paragraphs we will explain the sensor models used and derive the required filter equations. For readability, this derivation is carried out for the case of a single tag, but is directly applicable to the case of multiple tags.

Coordinate Systems In our filter setup we assume different coordinate frames. First, we assume an inertial workspace coordinate system W. We assume that gravity points in negative z-direction in this frame. Furthermore, we define the IMU coordinate system B and the camera frame V. Finally, we define a coordinate system T for each tag which coincides with the geometrical center of the tag and where z is perpendicular to the tag plane.

Sensor Models First, we introduce the sensor model used for the IMU. It assumes Gaussian noise as well as additive bias terms for accelerometer and gyroscope measurements. This can be formulated as follows:

$$\tilde{\boldsymbol{f}} = \boldsymbol{f} + \boldsymbol{b}_f + \boldsymbol{w}_f, \tag{4.22}$$

$$\dot{\boldsymbol{b}}_f = \boldsymbol{w}_{bf}, \tag{4.23}$$

$$\tilde{\boldsymbol{\omega}} = \boldsymbol{\omega} + \boldsymbol{b}_\omega + \boldsymbol{w}_\omega, \tag{4.24}$$

$$\dot{\boldsymbol{b}}_\omega = \boldsymbol{w}_{b\omega}, \tag{4.25}$$

where $\tilde{\boldsymbol{f}}$ and $\tilde{\boldsymbol{\omega}}$ are the actual measurements of the proper acceleration and rotational rates, $\boldsymbol{b}_f$ and $\boldsymbol{b}_\omega$ are the additive bias terms, and all terms of the form $\boldsymbol{w}_*$ represent continuous white Gaussian noise processes.

In addition to the IMU data, we will also include measurements related to the observed tags. For this measurements we propose a tight coupling by using a corner reprojection based visual model. Given the relative position and attitude of a specific tag with respect to the camera frame, ${}_{\boldsymbol{V}}\boldsymbol{r}_{\boldsymbol{VT}}$ and $\boldsymbol{q}_{\boldsymbol{TV}}$, we can compute the position of the i^{th} tag corner ${}_{\boldsymbol{T}}\boldsymbol{r}_{\boldsymbol{TC_i}}$ (fixed to the tag coordinate frame $\boldsymbol{T}$) as viewed from the camera:

$$\boldsymbol{{}_V r_{VC_i}} =_{\boldsymbol{V}} \boldsymbol{r_{VT}} + \boldsymbol{q_{TV}^{-1}}(\boldsymbol{{}_T r_{TC_i}}). \tag{4.26}$$

By using the camera projection map π, we can project the above quantity onto the image plane and derive the corresponding pixel coordinate measurement $\tilde{\boldsymbol{p}}_i$, where we assume an additive Gaussian noise model ($\boldsymbol{n}_{p,i} \sim \mathcal{N}(0, \boldsymbol{R}_p)$):

$$\tilde{\boldsymbol{p}}_i = \pi(\boldsymbol{{}_V r_{VC_i}}) + \boldsymbol{n}_{p,i}. \tag{4.27}$$

The advantage of the selected visual measurement model is that the noise is modelled directly on the pixel location of the detected corners. While the detector provides an estimation of the tag pose relative to the current camera frame and this could be directly used within the filter, fitting an accurate noise model to this relative pose would have been difficult since the magnitude of the noise strongly depends on the current location and orientation of the tag in the camera frame. In contrast, the noise on the reprojected tag corners is, to a large extent, indifferent with respect to the camera pose and can thus be assumed to be constant and identical for all tags and measurements.

Filter States and Prediction Model The above visual sensor model assumes the knowledge of the tag pose. Instead of using fixed values, which could quickly lead to inconsistencies, we propose to include the pose of the tag into the filter state. Therefore, the filter will be able to refine the tag pose and ensure map consistency. Employing a robocentric representation of the sensor state and the tag pose, we get the following filter state:

$$\begin{aligned} \boldsymbol{x} &:= \big(\boldsymbol{r}, \boldsymbol{v}, \boldsymbol{q}, \boldsymbol{b}_f, \boldsymbol{b}_\omega, \boldsymbol{r}_T, \boldsymbol{q}_T, \boldsymbol{r}_V, \boldsymbol{q}_V\big) \qquad (4.28)\\ &:= \big({}_B\boldsymbol{r}_{WB}, {}_B\boldsymbol{v}_B, \boldsymbol{q}_{WB}, {}_B\boldsymbol{b}_f, {}_B\boldsymbol{b}_\omega, \\ &\qquad {}_V\boldsymbol{r}_{VT}, \boldsymbol{q}_{TV}, {}_B\boldsymbol{r}_{BV}, \boldsymbol{q}_{VB}\big). \qquad (4.29) \end{aligned}$$

In the above state, $\boldsymbol{r}$, $\boldsymbol{v}$, and $\boldsymbol{q}$ are the robocentric position, velocity, and attitude of the sensor. Furthermore, $\boldsymbol{r}_T$ and $\boldsymbol{q}_T$ are used for parametrizing the pose of the tag, while $\boldsymbol{r}_V$ and $\boldsymbol{q}_V$ represent the extrinsic calibration between IMU and camera. Computing the total derivatives of the selected state and inserting the IMU model

(4.22)-(4.25) yields:

$$
\begin{aligned}
\dot{\boldsymbol{r}} &= -\,(\tilde{\boldsymbol{\omega}} - \boldsymbol{b}_\omega - \boldsymbol{w}_\omega)^{\times}\boldsymbol{r} + \boldsymbol{v} + \boldsymbol{w}_r, && (4.30)\\
\dot{\boldsymbol{v}} &= -\,(\tilde{\boldsymbol{\omega}} - \boldsymbol{b}_\omega - \boldsymbol{w}_\omega)^{\times}\boldsymbol{v} \\
&\qquad + \tilde{\boldsymbol{f}} - \boldsymbol{b}_f - \boldsymbol{w}_f + \boldsymbol{q}^{-1}(\boldsymbol{g}), && (4.31)\\
\dot{\boldsymbol{q}} &= -\,\boldsymbol{q}(\tilde{\boldsymbol{\omega}} - \boldsymbol{b}_\omega - \boldsymbol{w}_\omega), && (4.32)\\
\dot{\boldsymbol{b}}_f &= \boldsymbol{w}_{bf},\ \dot{\boldsymbol{b}}_\omega = \ \boldsymbol{w}_{b\omega}, && (4.33)\\
\dot{\boldsymbol{r}}_T &= -\,\boldsymbol{q}_V\left((\tilde{\boldsymbol{\omega}} - \boldsymbol{b}_\omega - \boldsymbol{w}_\omega)^{\times}(\boldsymbol{q}_V^{-1}(\boldsymbol{r}_T) + \boldsymbol{r}_V) + \boldsymbol{v}\right) \\
&\qquad + \boldsymbol{w}_{rt}, && (4.34)\\
\dot{\boldsymbol{q}}_T &= -\,(\boldsymbol{q}_T \otimes \boldsymbol{q}_V)(\tilde{\boldsymbol{\omega}} - \boldsymbol{b}_\omega - \boldsymbol{w}_\omega + \boldsymbol{w}_{qt}), && (4.35)\\
\dot{\boldsymbol{r}}_V &= \boldsymbol{w}_{rv},\ \dot{\boldsymbol{q}}_V = \ \boldsymbol{w}_{qv}. && (4.36)
\end{aligned}
$$

We include additional continuous white Gaussian noise processes $\boldsymbol{w}_r$, $\boldsymbol{w}_{rt}$, $\boldsymbol{w}_{qt}$, $\boldsymbol{w}_{rv}$, and $\boldsymbol{w}_{qv}$ in order to excite the full filter state and for modeling errors caused by the subsequent discretization of the states. For all white Gaussian noise processes $\boldsymbol{w}_*$, the corresponding covariance parameters $\boldsymbol{R}_*$ describe the magnitude of the noise. While most covariance parameters can be chosen by considering the corresponding sensor specifications, some remain as tuning parameters. Using a simple Euler forward integration scheme a set of discrete time prediction equations can be derived. In order to achieve a minimal and consistent parametrization, the derivatives of the quaternions are expressed in a 3D local angular velocity. This has to be considered during the discretization and when implementing the filter. Also note that, during the prediction, the IMU-related states $(\boldsymbol{v}, \boldsymbol{b}_\omega)$ are coupled to the estimated tag pose $(\boldsymbol{r}_T, \boldsymbol{q}_T)$ based on the IMU-camera extrinsics estimates $(\boldsymbol{r}_V, \boldsymbol{q}_V)$.

Update Model The update step is performed by directly employing the reprojection error as the Kalman filter innovation term. For each tag corner i and based on equation (4.27) we can define an innovation term $\boldsymbol{y}_i$:

$$
\boldsymbol{y}_i = \tilde{\boldsymbol{p}}_i - \pi({}_{\boldsymbol{V}}\boldsymbol{r}_{\boldsymbol{V}\boldsymbol{C}_i}). \qquad (4.37)
$$

This results in a 8D innovation term for every tag detected in the current camera frame (2D per tag corner). For each newly observed tag the state is augmented by an additional tag pose, i.e. position and attitude. The augmentation uses the estimated relative pose from the tag tracker in order to initialize the state with a good linearization point. The corresponding covariance matrices are initialized to large values and typically converge very quickly. Optionally, tags with known absolute location can also be fed to the filter. Especially for datasets with a large number of tags this comes in handy since the EKF does not scale well with increasing state dimension. Above around 20 tag poses in the filter state the prediction step becomes very costly for a single core implementation.

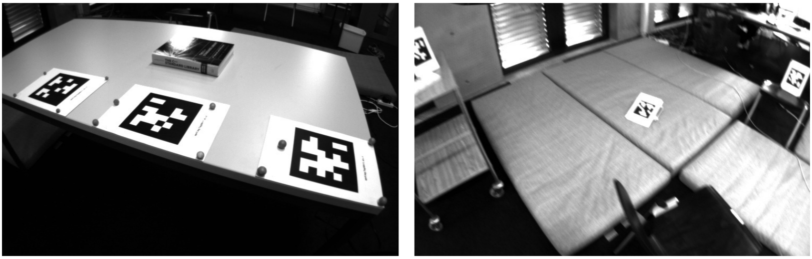


Figure 4.11: Images extracted from datasets ”table“ (left) and ”dataset_1“ (right).

VII.3 Results

In order to assess the performance of the proposed approach, we define different test procedures. In a first test, we verify the accuracy of the fiducial pose estimation. In a second test, we then evaluate the accuracy of the motion estimation computed by our EKF. Both tests are verified with ground truth data obtained from a high class external motion capture system. Additionally, two large scale datasets are processed for verifying the accuracy when closing larger loops. A third test evaluates the estimation of the extrinsic calibration. Lastly, we test the applicability of the presented motion estimation within an online closed loop control on a quadruped robot.

Datasets

In total, we are using five datasets which are all available for download with the source code. The first dataset ”table“ consists of three tags that are placed flat on a table at the same orientation as shown in Figure 4.11. The distances between the tags are chosen to be of similar magnitude. The second dataset ”dataset_1“ also contains three tags. This time, we tried to create a challenging dataset, where the tags are sparsely distributed around a larger workspace of about 4x4x4m. Furthermore, the tags are intentionally oriented and located in such a way that the viewing angle for the camera is not ideal and that only a minimal amount of frames contain two neighboring tags at the same time. Additionally, the sensor is moved fast, such that motion blur occurs occasionally. Overall, this increases the level of difficulty in estimating the tags’ locations. An on-board image taken by the camera, showing the challenging setup as well as the motion blur is shown in Figure 4.11. For evaluating the extrinsic calibration estimation, we are using a set of datasets all taken within the same workspace to ensure a consistent setup. In these datasets, the sensor is subject to extensive motion in order to properly excite the full filter state and thereby promote the convergence of the estimated extrinsic calibration. The last two dataset ”cube“ and ”pavillon“ contain round trips on our campus. Both datasets include around 35 tags and span areas of approximately 25x25x6 m. By

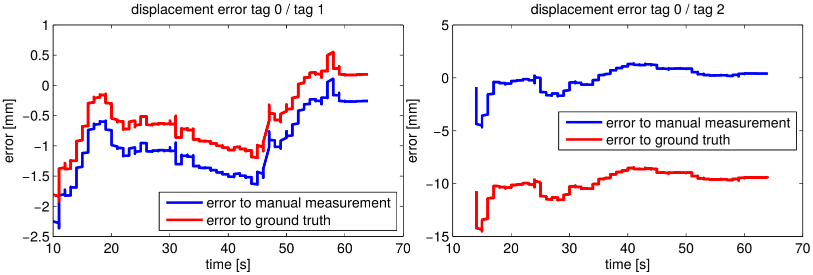


Figure 4.12: Displacement error between estimated tag positions and reference from manual measurements as well as external motion capture. As can be seen, the error decreases over time, since the tags' positions are iteratively refined by the EKF. Finally, submillimeter accuracy is achieved. The larger offset on the right plot most likely results from inaccurate marker and coordinate system placement in the external motion capture system.

moving from basement to ground level or from indoors to outdoors, these datasets are subject to changing lighting conditions. To provide comparable results, no test specific parameter tuning has been performed, i.e. the same parameters are used throughout all tests.

Fiducial Estimation Test

For the verification of the continuous fiducial estimation procedure, we use the "table" dataset. In this dataset, manually measuring the offset between the tags is simple. Thus, we can use these measurements as ground truth information and compare it to the estimates of the external motion capture system. This allows us also to evaluate the accuracy of the marker and coordinate system placement during the set up of the external motion capture system. To isolate the fiducial estimation for testing, we are disabling the extrinsic calibration in this test and use the sensor's factory calibration.

We compare the norm of the relative translation, i.e. the distance between tag 0 and tag 1 as well as between tag 0 and tag 2 with the manual distance measurements. This error plots are shown in Figure 4.12. The plots shows two interesting aspects. The errors in the beginning of the sequence is quite small. This indicates that the initial guess obtained from the reprojection error optimization on the first frame is fairly accurate. Over time, our EKF then further refines the poses, reaching approximately millimeter accuracy which is of equal magnitude as manual measuring errors.

The figures also shows the error with respect to the external motion capture measurement. Here, the error is shifted by about 1cm for the translation between tag 0 and tag 2. Since a zero mean error curve would be expected, this constant

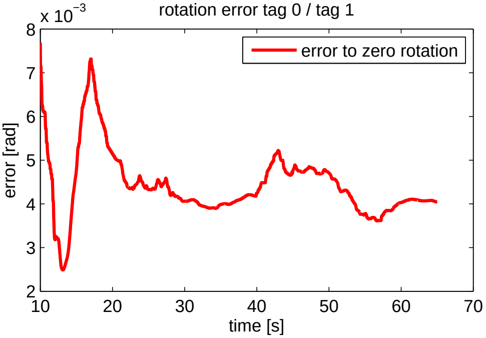


Figure 4.13: Rotation error between estimated tag positions and zero rotation. The error is obtained by converting the relative rotation to angle-axis representation of which the angle is plotted. As can be seen, the error decreases over time, since the tags' rotations are iteratively refined by the EKF. The error starts at around 0.5 degrees and reduce to about 0.2 degrees.

offset most likely results from inaccurate marker and reference coordinate system placement.

Since the tags in this dataset are placed flat on a table and aligned with the table's edge, we can also analyze the rotation error of our tag pose estimates. To do so, we compute the relative rotation between two tags. We then convert this rotation to an axis-angle representation and use the angle as our error measurement. Due to the tag alignment, the relative rotation between two tags can be assumed to be identity. This is also confirmed by the external motion capture system up to the fourth decimal of the relative rotation angle. Figure 4.13 shows the error between estimated rotation and the identity rotation for the relative rotation between tag 0 and tag 1. As can be seen, the error is initially around 0.5 degrees. Through continuous refinement of the tag poses within the EKF, this error reduces to around 0.2 degrees over time. This error is of same magnitude as printing and measurement accuracy.

The experiments described above show the high achievable fiducial estimation accuracy in translation and rotation. Furthermore, these results underline that tag pose refinement significantly reduces displacement and rotational errors present in the single frame pose estimate used for initialization. This will eventually improve the consistency of the relative tag poses and thus should also improve the robot's pose estimation.

Motion Estimation Test

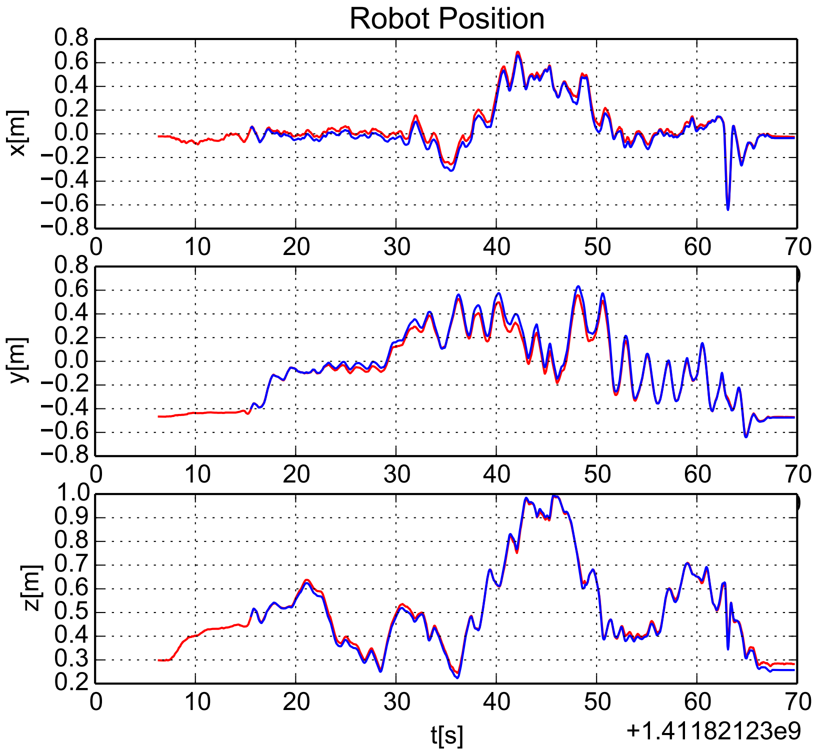


Figure 4.14: Comparison between estimated robot position (blue) and ground truth position (red) for the dataset "table". As can be seen, the maximum position offset between both measurements lie only within a centimeter scale which is the same magnitude as the achievable measurement accuracy in this setup.

Dataset *Table* To assess the performance of the motion estimation, we use both datasets described above. The goal of our estimation framework is to localize against our workspace, where we choose tag 1 as the origin. This choice is arbitrary and one could choose any tag as a reference defining the workspace location and orientation. Since our estimator automatically estimates the orientation of the workspace with respect to gravity, no manual alignment is required. Figure 4.14 shows a comparison of the position estimates of the filter and ground truth data from the external motion capture system for the *table* dataset. This plot nicely illustrates the robust tracking behavior of the system. Even though the reference tag is not detectable at every instance of the dataset, the estimated fiducials provide a stable reference for the filter to localize against, such that tracking errors remain a few centimeters.

In Figure 4.15 the estimated orientation and ground truth orientation for the same datasets are compared. Also here, the estimator shows a robust tracking with minimal deviations. The maximum error observed in pitch direction is about 0.05 rad which corresponds to less than 3 degrees. Since the ground truth reference data is a relative pose between the sensor and the reference tag computed from the individual poses, the error magnitudes observed above lie within the measurement accuracy of the ground truth data. While this underlines the performance of the approach, it does not give any indication about its limits. Therefore, we tried to push the system to its limits using *dataset_ 1* which contains several artificial challenges as described above.

Dataset *Dataset_ 1* In this experiment, again the estimated position is compared to ground truth data and the results are shown in Figure 4.16. As the plots show, the position starts to deviate from ground truth in the last third of the sequence. While results are not as good as in the *table* dataset, *dataset_ 1* can be seen as a worst-case benchmark scenario. Most of the difficulties for the algorithm are artificially posed and the performed motion is faster than in many robot applications. Due to the sparse tag placement and fast motions, the detector was unable to detect any tag in many of the images of the sequence. This is shown in Figure 4.18 where these instances are marked with the value 1. In total, the filter is provided with inertial measurements only for almost 20% of the sequence. Additionally, tag 0 is only seen together with another tag for in total 9 frames. Thus, little localization information is provided for this tag, leading to a high uncertainty of the tags pose. Still, it is the only visible tag for about 15% of the dataset. Thus, the filter is oly provided with uncertain vision information and noisy inertial measurements during these parts. However, the filter remains stable and is able to converge close to ground truth data again when the other tags are visible again.

Also in the orientation, the effects of sparsely distributed tags combined with fast motions are visible. Figure 4.17 shows the difference between ground truth and estimated orientation for *dataset_ 1*. As can be seen, the orientation estimate is fairly accurate throughout the dataset with a slight deviation in yaw at the beginning of the trajectory and a small deviation of pitch of about 9 degrees towards the end.

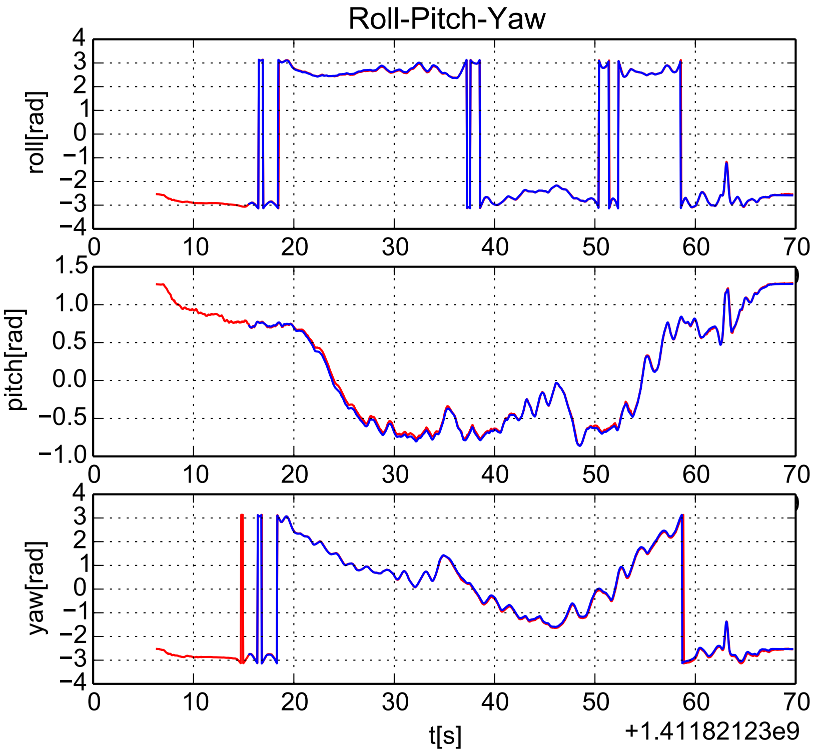


Figure 4.15: Comparison between estimated robot orientation (blue) and ground truth orientation (red) for the dataset "table". Due to the wrap-around at $+/-\pi$ the plot is discontinuous. However, since quaternions are used for the internal representation of the filter, the output of the filter is smooth. As also seen in the position data, estimated and ground truth rotations agree up to measurement uncertainty.

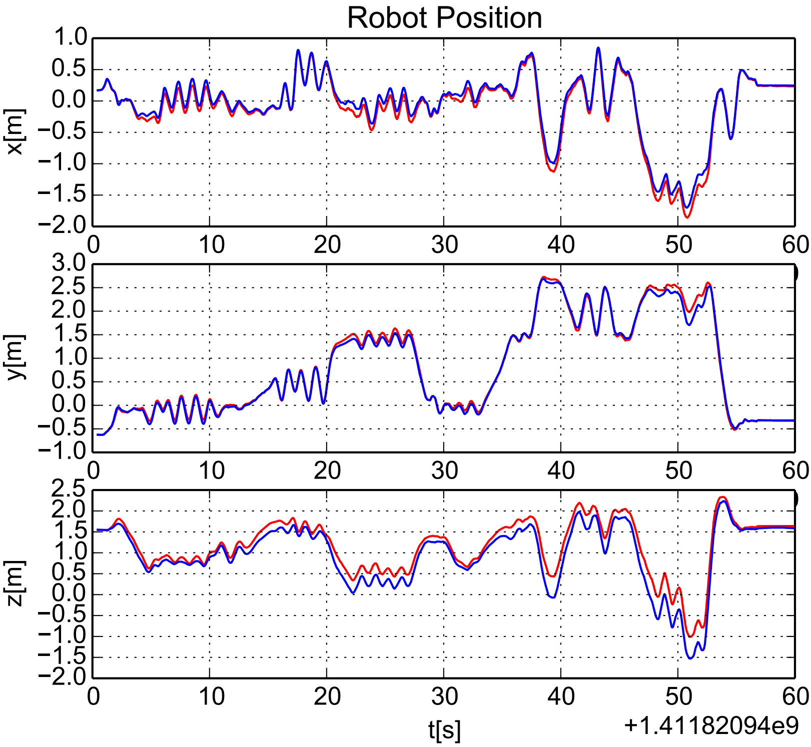


Figure 4.16: Comparison between estimated robot position (blue) and ground truth position (red) for the dataset ”dataset_1“. This dataset has been made artificially difficult with sparse tag coverage and fast motions to show the robustness of the filter. While the estimates diverges when only the briefly observed tag on the very left can be used for localization, it converges back to the ground truth information when localizing against the other tags again.

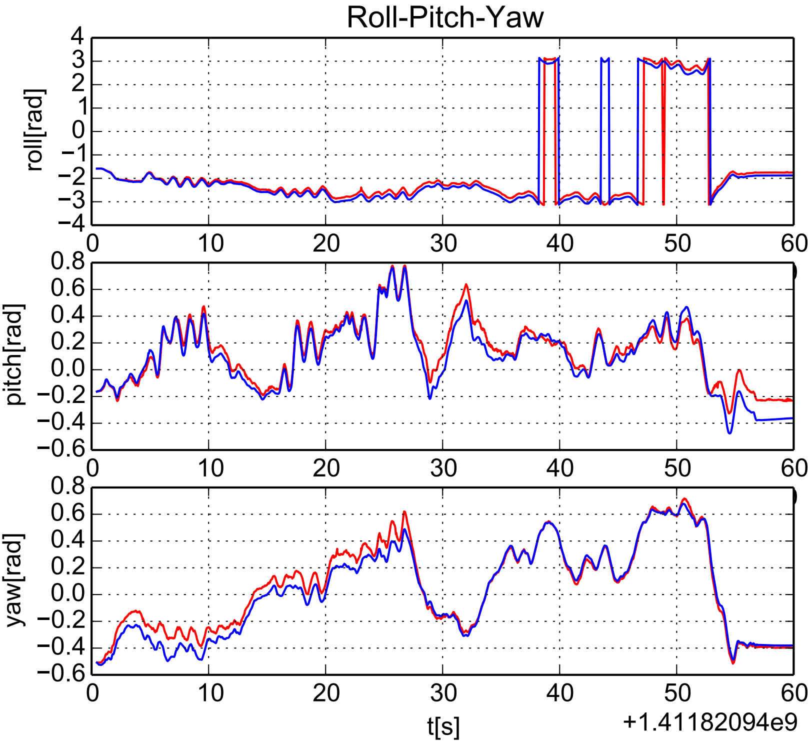


Figure 4.17: Comparison between estimated robot orientation (blue) and ground truth orientation (red) for the dataset ”orientation_1“. Due to the wrap-around at $+/-\pi$ the plot is discontinuous. However, since quaternions are used for the internal representation of the filter, the output of the filter is smooth.

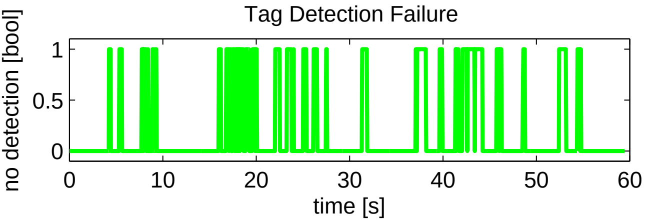


Figure 4.18: Plot indicating whether one or multiple tags were detected (indicated as 0) or no tag was detected (indicated as 1) for *dataset_ 1*. Overall, in almost 20% of all images no tag could be detected.

When looking at the linear velocity estimates for this dataset shown in Figure 4.20, one can see that the estimates agree well with the velocity data obtained by using finite differences on the ground truth data. Interestingly, the estimated velocities are virtually outlier free while the finite differences show occasional peaks. This effect still occurs, even though a high quality motion capture system has been used. This underlines the limitations of using finite differences for velocity estimates and encourages the use of inertial data.

This effect is even more pronounced when looking at Figure 4.19 which shows the rotational velocity estimates and their counterparts computed using finite differences on the ground truth orientation. The difference in noise level between both measurements is significant. One reason is that the IMU directly measures rotational rates using gyroscopes. Furthermore, rotations tend to be more difficult to estimate for external motion capture systems. This effect gets amplified when differentiating this noisy signal.

Large scale datasets One advantage of the presented approach over a commercial motion capture system is the workspace size. Since our system only relies on paper printed tags, a large workspace can be covered easily. As we did not have a motion capture system available that is capable of covering such a large area, especially not outdoors, we are using loop closure to estimate the accuracy of the approach. For this test, we are using the datasets "pavillon“ and "cube“ which both include loop closure sequences. As a quality measure, the reprojection errors as well as the offset between the estimator's predicted tag position and the detector's instantaneous tag measurement are used. Both measures are taken at the first time that we reobserve a tag after the round trip, before updating the estimator.

For the dataset "cube“ the average reprojection error at loop closure is 56.07 pixels. Taking the detector pose estimate as a reference, the position offset is 0.86 m. One round trip until loop closure is about 70 m long and follows a trail of 36 tags. Therefore, the relative position error is around 1.2 %. For the dataset "pavillon“ the average reprojection error at loop closure is 51.01 pixels. Taking the detector

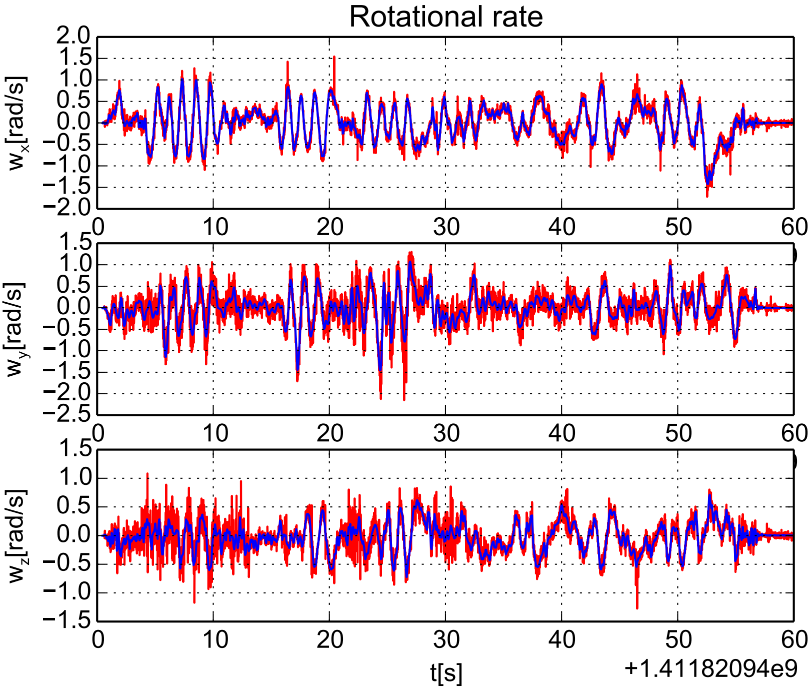


Figure 4.19: Comparison of rotational velocity estimates (blue) and rotational velocities calculated by using finite differences of the ground truth orientation data (red). Like also with the linear velocities shown in Figure 4.20, the estimation matches the ground truth data. Here the significance of using inertial measurements for low-noise estimates over finite differences on pose information is even more prominent.

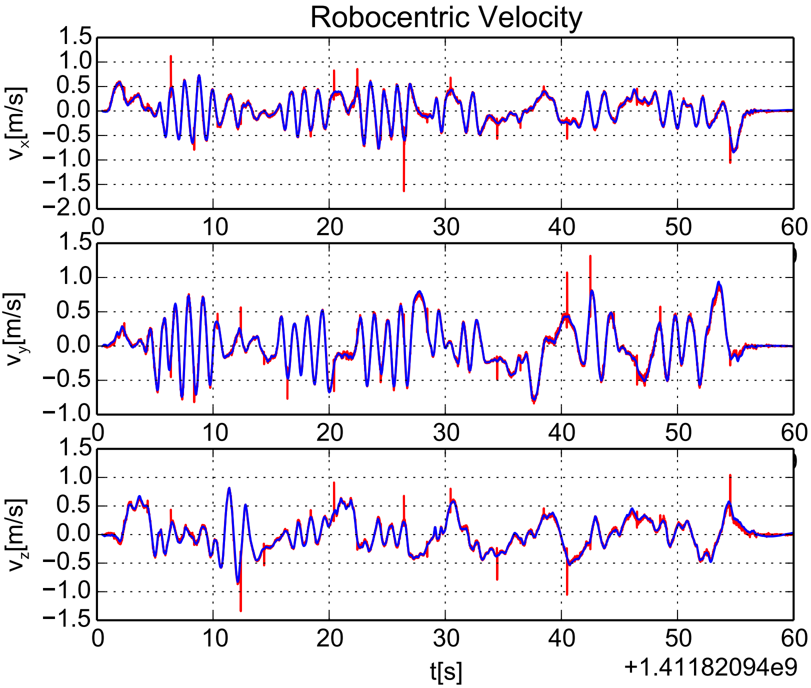


Figure 4.20: Comparison of linear velocity estimates (blue) and linear velocities calculated by using finite differences of the ground truth position data (red). While the estimated velocities agree well with the velocities computed from ground truth data, they are virtually outlier free. While a high quality external motion capture system is used, this data still shows the limitations of finite differences for velocity estimates.

pose estimate as a reference, the position offset is 0.38 m. One round trip until loop closure is about 80 m and follows a trail of 33 tags. Therefore, the relative position error is around 0.5 %. Please note that position errors are calculated using the detector estimate. This estimate cannot be assumed to be a ground truth measurement. Therefore, the accuracy figures above are subject to measurement inaccuracies of the detector and based on one measurement only.

Online Extrinsic Estimation

In order to assess the quality of the estimated camera-IMU extrinsics, we evaluated the corresponding values after the system was sufficiently excited such that the values could converge. Since no real groundtruth references were available for the extrinsics, we evaluated the repeatability of the obtained estimates. For this we recorded 10 datasets within the same environment while performing similar motions with a total duration around 50-60 seconds. The obtained RMS-values were 1.5 cm for the translational part and 0.0035 rad for the rotational part of the extrinsics. Both values ranging near what can typically be obtained through a dedicated calibration routine.

Estimation in Closed Loop Control

Motion capture systems are increasingly used in closed-loop control. Since latency, noise and outliers can significantly deteriorate the closed-loop behavior of the plant, estimation in the loop is a challenging task. Therefore, we test our system in such an application. For this test, we are using our quadruped robot HyQ on a hydraulically actuated treadmill. The control task is to keep the robot in the center of the treadmill by only regulating the speed of the treadmill, i.e. the robot's walking motion is assumed to be a disturbance. The control system is a cascade of an inner velocity and an outer position control loop. The sensor input to the position control loop are the robot's position and velocity in the workspace.

As can be seen from the accompanying video[3], the closed loop system is able to stabilize the robot's position on the treadmill while changing the walking speed. The estimate of the absolute position of the robot in the workspace allows us to move the robot to the treadmill center during initialization.

VII.4 Conclusion and Future Work

In this paper, we have presented an open-source, visual-inertial state estimation system, that tightly integrates monocular SLAM and fiducial based estimation. By relying on standard hardware already present on most robots the system can be applied cost efficiently. Experiments demonstrate its good accuracy and high robustness, which indicates that it could replace an expensive motion capture systems in applications that do not require sub-millimeter precision or very fast update rates only offered by highly expensive motion capture systems. This has

---

[3] `http://youtu.be/Ckf1QAuTKqc`

been verified by using the system in a closed-loop control task. Large scale tests have demonstrated long term accuracy, map consistency and loop closure refinement. Experiments under fast motions and sparse tag coverage of the workspace underline the importance of including inertial measurements compared to fiducial only approaches. Furthermore, the inertial measurements ensure high quality translational and rotational velocity estimates which can outperform these of a commercial system. Results have shown that good coverage of fiducials is important for a good estimation quality. In the future, we aim at supporting differently sized fiducials such that their size can be optimized for their intended location. Furthermore, we will investigate the combination with natural landmarks in order to further improve the estimation accuracy.

VII.5 Acknowledgement

The authors would like to thank Sammy Omari, the Autonomous Systems Lab and Skybotix for their support with the external motion capture system and the VI-Sensor. Furthermore, the authors would like to thank Manuel Lussi for the support with the treadmill experiments.

Chapter 5

Conclusion and Outlook

1 Summary

This work presents a major step towards applying Numerical Optimal Control as Model Predictive Control using fully dynamic models on physical robots. It underlines the potential of such approaches for serving as a general motion planning and control framework for different robot types and tasks. With its focus on hardware experiments, the thesis provides insight into transferability from simulation to hardware. In this work, we have demonstrated that Numerical Optimal Control is a suitable tool for solving complex motion planning tasks such as locomotion or flying in constrained spaces. Furthermore, this work has demonstrated that Trajectory Optimization cannot only optimize periodic gait patterns but also discover them automatically and adjust them during execution. By applying our framework to flying, ground and legged robots, we have shown that the same Numerical Optimal Control approach can be applied to vastly different robot types and tasks, fulfilling our vision of "One Algorithm – Many Robots and Tasks". Additionally, this work addresses fundamental issues of running Trajectory Optimization online on physical robots. We have implemented a customized, high performance solver for one of the fastest Numerical Optimal Control approaches and paired it with efficient system modelling and derivative computation. This leads to a framework that outperforms state-of-the-art Numerical Optimal Control approaches. This enables us to run Nonlinear Model Predictive Control on more complex problems as well as at larger time horizons while achieving a shorter computational time than previous approaches.

2 Discussion and Future Work

While Numerical Optimal Control is a powerful tool for tackling motion planning and control, it is not the silver bullet that solves all robotic motion planning and control. Instead, it is an essential tool which offers the possibility to be combined with other algorithms.

As a model based approach the performance of Optimal Control is bound by the accuracy of our system description. This suggests to combine it with Machine Learning approaches to improve performance in presence of modelling errors. In early work related to this thesis [68], we have taken first steps towards using reinforcement learning to improve a model-based Optimal Control policy. Furthermore, Numerical Optimal Control could efficiently be combined with model learning, with early work e.g. presented in [129, 153]. We believe there is a huge potential of using prior knowledge from a model and the associated gradient information, while using data from the physical platform to improve the model or the policy. Such approaches combine best of both worlds, i.e. fast optimization while also adapting to the physical system with a minimal amount of time-intensive hardware experiments required.

As a gradient-based approach Numerical Optimal Control is still a local optimization method that relies on the convexity of the problem and smooth gradients. However, it is a challenge in itself to find such problem descriptions. Especially when dealing with obstacles or rough terrain locomotion, we face issues with our current formulation. However, there are known algorithms that tackle these issues such as randomized or hierarchical planners. Therefore, when facing practical real world problems, combining Trajectory Optimization with such planners remains a viable option. The work on Model Predictive Control for ballbots and UAVs in this thesis has underlined the potential of using a combined approach for planning and control. Similar trends can be observed in hierarchical planners for legged locomotion, where the individual hierarchies are now often run in receding horizon fashion as well [128, 247]. Ideally, with increase in computational power, the strict hierarchical approach can be broken up and more parts of the overall problem can solved simultaneously rather than sequentially.

As a step in this direction, we have pushed the boundaries with respect to the efficiency of our solvers and also our system dynamics computations. Furthermore, we have tackled the drawback of DDP-style approaches struggling to handle constraints without loosing linear time complexity by introducing penalty based constraints. However, this comes at the cost of additional iterations. Therefore, a direct comparison to hard constraint methods such as [201] should be made.

NLP and DDP-style approaches have finally fully converged in their solver strategies [54, 84], resulting in a unique opportunity to leverage the advantages of both algorithms. NLP methods add proper constraint handling while DDP-style algorithms add a feedback control view, allowing to use what seem to be numerical byproducts as feedback control gains. This gives way for plenty of interesting applications in the field of MPC and Numerical Optimal Control. It enables solvers that are highly efficient but also compute constraint satisfactory solutions.

There is also still considerable amount of work to be done on modelling and simulation. As we have shown, simulators are not very suitable tools for control, neither as modelling frameworks for Numerical Optimal Control nor as verification tools. On the other hand, there are very capable Rigid Body Dynamic libraries that allow efficient computation of such models. These tools slowly start to grow together but so far there is still a gap in-between. Furthermore, the performance gain of Numerical Optimal Control allows for considering more complex models that potentially could include effects such as actuator dynamics or sensor properties. Including such models in our system description will further improve performance. Additionally, such more accurate models or simulations could also support efforts in the learning community, reducing the amount of experimental data to be collected on the physical system. On the other hand, Numerical Optimal Control approaches can learn from physics engines how to deal with Rigid Body Dynamics. Many physics engines already address the issues of stiff systems resulting from Rigid Body Dynamics in contacts. Concepts such as implicit integration schemes or variable contact stiffness are state-of-the-art approaches in physics engines and could directly be applied to the methods present in this work.

One major challenge remaining is robustness. We have shown that Model Predictive Control has the potential to reject large disturbances and we consider a combination of fast feedback controllers and large time horizon predictive controllers an ideal combination for rejecting fast, small amplitude as well slow, larger amplitude disturbances. However, the “risk” or robustness of a trajectory is not factored in during the optimization phase of most Numerical Optimal Controllers. While heuristic based robustness criteria can be introduced, we still have to tackle this issue in a more principled way. For Differential Dynamic Programming approaches, promising early results exist [67, 149, 187] but they still have to be verified with practical applications and hardware experiments.

Bibliography

[1] M. W. Achtelik, S. Lynen, M. Chli, and R. Siegwart. Inversion based direct position control and trajectory following for micro aerial vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2933–2939, 2013.

[2] F. Aghili. A Unified Approach for Inverse and Direct Dynamics of Constrained Multibody Systems Based on Linear Projection Operator : Applications to Control and Simulation. *IEEE Transactions on Robotics*, 21(5):834–849, 2005.

[3] I. Agustí-Juan, F. Müller, N. Hack, T. Wangler, and G. Habert. Potential benefits of digital fabrication for complex structures: Environmental assessment of a robotically fabricated concrete wall. *Journal of Cleaner Production*, 154:330–340, 2017. doi: https://doi.org/10.1016/j.jclepro.2017.04.002.

[4] S. Ahn, M. Choi, J. Choi, and W. Chung. Data Association Using Visual Object Recognition for EKF-SLAM in Home Environment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2588–2594, 2006. doi: 10.1109/IROS.2006.281936.

[5] K. Alexis, C. Papachristos, G. Nikolakopoulos, and A. Tzes. Model predictive quadrotor indoor position control. In *Mediterranean Conference on Control & Automation (MED)*, pages 1247–1252, 2011.

[6] J. Andersson, J. Akesson, and M. Diehl. CasADi: A symbolic package for automatic differentiation and optimal control. In *Recent advances in algorithmic differentiation*, pages 297–307. Springer, 2012.

[7] H. Asada and J.-J. E. Slotine. Robot analysis and control. *Automatica*, 24 (2):289, 1988. doi: 10.1016/0005-1098(88)90042-8.

[8] Ascending Technologies. Firefly. URL `http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-firefly/`. date retrieved: 2017-04-25.

[9] M. Azad and R. Featherstone. A new nonlinear model of contact normal force. *IEEE Transactions on Robotics*, 30(3):736–739, 2014. doi: 10.1109/TRO.2013.2293833.

[10] M. Azad, V. Ortenzi, H.-C. Lin, E. Rueckert, and M. Mistry. Model Estimation and Control of Compliant Contact Normal Force. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 442–447, 2016. doi: 10.1109/HUMANOIDS.2016.7803313.

[11] V. Barasuol, J. Buchli, C. Semini, M. Frigerio, E. R. De Pieri, and D. G. Caldwell. A reactive controller framework for quadrupedal locomotion on challenging terrain. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2554–2561, 2013. doi: 10.1109/ICRA.2013.6630926.

[12] B. M. Bell. CppAD: a package for C++ algorithmic differentiation. *Computational Infrastructure for Operations Research*, 2012. URL `http://www.coin-or.org/CppAD`. date retrieved: 2017-04-25.

[13] R. Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719, 1952.

[14] C. Bendtsen and O. Stauning. Fadbad, a flexible C++ package for automatic differentiation. Technical report, Department of Mathematical Modelling, Technical University of Denmark, 1996.

[15] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific Belmont, MA, 1996.

[16] J. T. Betts. Survey of Numerical Methods for Trajectory Optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, 1998. doi: 10.2514/2.4231.

[17] C. H. Bischof. On the automatic differentiation of computer programs and an application to multibody systems. In *IUTAM Symposium on Optimization of Mechanical Systems*, pages 41–48, 1996.

[18] S. Blackmore, B. Stout, M. Wang, and B. Runov. Robotic agriculture - The future of agricultural mechanisation? In *5th European Conference on Precision Agriculture (ECPA)*, pages 621–628, 2005. doi: 10.1007/s00586-005-0966-7.

[19] M. Bloesch, M. Hutter, M. Hoepflinger, S. Leutenegger, C. Gehring, C. D. Remy, and R. Siegwart. State Estimation for Legged Robots - Consistent Fusion of Leg Kinematics and IMU. In *Robotics: Science and Systems*, pages 17–24, 2012. doi: 10.15607/RSS.2012.VIII.003.

[20] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart. Robust visual inertial odometry using a direct EKF-based approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 298–304, 2015. doi: 10.1109/IROS.2015.7353389.

[21] T. Boaventura. *Hydraulic compliance control of the quadruped robot HyQ*. PhD thesis, Advanced Robotics Department, University of Genova, 2013.

[22] J. E. Bobrow. Optimal Robot Path Planning Using the Minimum-Time Criterion. *IEEE Journal on Robotics and Automation*, 4(4):443–450, 1988. doi: 10.1109/56.811.

[23] H. G. Bock and K.-J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems. In *IFAC World Congress*, 1984.

[24] T. Bock. Construction Robotics. *Autonomous Robots*, 22(3):201–209, 2007.

[25] R. Bogue. The role of robotics in non-destructive testing. *Industrial Robot: An International Journal*, 37(5):421–426, 2010. doi: 10.1108/01439911011063236.

[26] R. Bongiovanni and J. Lowenberg-Deboer. Precision agriculture and sustainability. *Precision Agriculture*, 5(4):359–387, 2004. doi: 10.1023/B:PRAG.0000040806.39604.aa.

[27] Boston Dynamics. Atlas Robot. URL `http://www.bostondynamics.com/robot_Atlas.html`. date retrieved: 2017-04-25.

[28] S. Bouabdallah and R. Siegwart. Full control of a quadrotor. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 153–158, 2007.

[29] S. Bouabdallah, A. Noth, and R. Siegwart. PID vs LQ Control Techniques Applied to an Indoor Micro Quadrotor. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2451–2456, 2004. doi: 10.1109/IROS.2004.1389776.

[30] P. Bouffard, A. Aswani, and C. Tomlin. Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 279–284, 2012.

[31] G. Bouza and G. Still. Mathematical programs with complementarity constraints: convergence properties of a smoothing method. *Mathematics of Operations research*, 32(2):467–483, 2007.

[32] A. Breitenmoser, L. Kneip, and R. Siegwart. A monocular vision-based system for 6D relative robot localization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 79–85, 2011.

[33] B. Brogliato. *Nonsmooth Mechanics*. Springer Verlag, 1999. doi: 10.1007/978-1-4471-0557-2.

[34] M. Bryson and S. Sukkarieh. Building a robust implementation of bearing-only inertial SLAM for a UAV. *Journal of Field Robotics*, 24(1-2):113–143, 2007.

[35] R. S. C. Gehring, S. Coros, M. Hutter, C. Dario Bellicoso, H. Heijnen, R. Diethelm, M. Bloesch, P. Fankhauser, J. Hwangbo, M. Hoepflinger. Practice makes perfect: an optimization-based approach to controlling agile motions for a quadruped robot. *IEEE Robotics & Automation Magazine*, 23(1):34–43, 2016.

[36] A. Callejo, S. H. K. Narayanan, J. G. de Jalón, and B. Norris. Performance of automatic differentiation tools in the dynamic simulation of multibody systems. *Advances in Engineering Software*, 73:35–44, 2014.

[37] S. Caron and Q.-C. Pham. When to make a step? Tackling the timing problem in multi-contact locomotion by TOPP-MPC. *arXiv preprint arXiv:1609.04600*, 2016.

[38] B. Carpenter, M. D. Hoffman, M. Brubaker, D. Lee, P. Li, and M. Betancourt. The stan math library: Reverse-mode automatic differentiation in C++. *arXiv preprint arXiv:1509.07164*, 2015.

[39] J. Carpentier, S. Tonneau, M. Naveau, O. Stasse, and N. Mansard. A versatile and efficient pattern generator for generalized legged locomotion. In *International Conference on Robotics and Automation*, pages 3555–3561, 2016.

[40] E. Catto. Soft Constraints – Reinventing the Spring. Game Developmers Conference, 2011.

[41] Y. Cho, J. Lee, and U. Neumann. A multi-ring color fiducial system and an intensity-invariant detection method for scalable fiducial-tracking augmented reality. In *IEEE International Workshop on Augmented Reality*, 1998.

[42] D. M. Cole and P. M. Newman. Using laser range data for 3D SLAM in outdoor environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1556–1563, 2006. doi: 10.1109/robot.2006.1641929.

[43] S. Coros, A. Karpathy, B. Jones, L. Reveret, and M. van de Panne. Locomotion skills for simulated quadrupeds. *ACM Transactions on Graphics*, 30(4):59, 2011. doi: 10.1145/2010324.1964954.

[44] E. Coumans. Bullet physics engine. URL `http://www.bulletphysics.org`. date retrieved: 2017-04-24.

[45] I. D. Cowling, J. F. Whidborne, and A. K. Cooke. Optimal Trajectory Planning and LQR Control for a Quadrotor UAV. *Automatica*, 37:1057–1064, 2001.

[46] H. Dai, A. Valenzuela, and R. Tedrake. Whole-body motion planning with centroidal dynamics and full kinematics. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 295–302, 2015. doi: 10.1109/HUMANOIDS.2014.7041375.

[47] C. De Crousaz, F. Farshidian, **M. Neunert**, and J. Buchli. Unified Motion Control for Dynamic Quadrotor Maneuvers Demonstrated on Slung Load and Rotor Failure Tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2223–2229, 2015. doi: 10.1109/ICRA.2015.7139493.

[48] R. Deits and R. Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. In *International Conference on Humanoid Robots*, pages 279–286, 2014.

[49] A. Del Prete and N. Mansard. Addressing Constraint Robustness to Torque Errors in Task-Space Inverse Dynamics. In *Robotics: Science and Systems (RSS)*, 2015. doi: 10.15607/RSS.2015.XI.027.

[50] M. Diehl and S. Gros. *Numerical Optimal Control.* expected to be published in 2017.

[51] M. Diehl, D. Leineweber, and A. Schäfer. MUSCOD-II users' manual. Technical report, Interdisciplinary Center for Scientific Computing (IWR), University of Heidelberg, Germany, 2001.

[52] M. Diehl, H. G. Bock, and J. P. Schlöder. A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on control and optimization*, 43(5):1714–1736, 2005.

[53] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber. Fast Direct Multiple Shooting Algorithms for Optimal Robot Control. *Fast motions in biomechanics and robotics*, pages 65–93, 2006.

[54] M. Diehl, H. Ferreau, and N. Haverbeke. Efficient numerical methods for nonlinear MPC and moving horizon estimation. *Nonlinear Model Predictive Control*, pages 391–417, 2009. doi: 10.1007/978-3-642-01094-1_32.

[55] P. Donald, B. and Xavier. Kinodynamic Motion Planning. *Journal of the ACM (JACM)*, 40(5):1048–1066, 1993. doi: 10.1109/MRA.2010.935794.

[56] A. Dürrbaum, W. Klier, and H. Hahn. Comparison of automatic and symbolic differentiation in mathematical modeling and computer simulation of rigid-body systems. *Multibody System Dynamics*, 7(4):331–355, 2002.

[57] P. Eberhard and C. Bischof. Automatic differentiation of numerical integration algorithms. *Mathematics of Computation of the American Mathematical Society*, 68(226):717–731, 1999.

[58] Eigen Linear Algebra Library. URL `http://eigen.tuxfamily.org`.

[59] Embotech. FORCES Pro. URL `https://www.embotech.com/FORCES-Pro`. date retrieved: 2017-04-24.

[60] T. Erez, Y. Tassa, and E. Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. In *Robotics: Science and Systems*, pages 73–79, 2012.

[61] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov. An integrated system for real-time model predictive control of humanoid robots. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 292–299, 2013. doi: 10.1109/HUMANOIDS.2013.7029990.

[62] T. Erez, S. Kolev, and E. Todorov. Receding-horizon online optimization for dexterous object manipulation. Preprint available onliine: https://homes.cs.washington.edu/ todorov/papers/ErezICRA14.pdf, 2014.

[63] M. Faessler, E. Mueggler, K. Schwabe, and D. Scaramuzza. A monocular pose estimation system based on infrared leds. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 907–913, 2014.

[64] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza. Aggressive Quadrotor Flight through Narrow Gaps with Onboard Sensing and Computing. *IEEE Robotics and Automation Letters*, 2016.

[65] P. Fankhauser and C. Gwerder. Modeling and control of a ballbot. *Bachelor thesis, ETH Zurich*, 2010.

[66] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart. Robot-Centric Elevation Mapping with Uncertainty Estimates. In *Mobile Service Robotics: International Conference on Climbing and Walking Robots (CLAWAR)*, pages 433–440, 2014. doi: 10.1142/9789814623353.

[67] F. Farshidian and J. Buchli. Risk Sensitive, Nonlinear Optimal Control: Iterative Linear Exponential-Quadratic Optimal Control with Gaussian Noise. *arXiv preprint arXiv:1512.07173*, 2015.

[68] F. Farshidian, **M. Neunert**, and J. Buchli. Learning of closed-loop motion control. In *IEEE International Conference on Intelligent Robots and Systems*, pages 1441–1446, 2014. doi: 10.1109/IROS.2014.6942746.

[69] F. Farshidian, N. Neunert, and J. Buchli. Learning of closed-loop motion control. In *International Conference on Intelligent Robots and Systems*, 2014.

[70] F. Farshidian, M. Kamgarpour, D. Pardo, and J. Buchli. Sequential Linear Quadratic Optimal Control for Nonlinear Switched Systems. In *International Federation of Automatic Control (IFAC)*, 2017.

[71] F. Farshidian, **M. Neunert**, A. W. Winkler, G. Rey, and J. Buchli. An efficient optimal planning and control framework for quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[72] R. Featherstone. *Rigid Body Dynamics Algorithms.* Springer, 2008. doi: 10.1007/978-0-387-74315-8.

[73] M. L. Felis. RBDL: an efficient rigid-body dynamics library using recursive algorithms. *Autonomous Robots*, 14(2):495–511, 2016. doi: 10.1007/s10514-016-9574-0.

[74] M. Fiala. ARTag, a fiducial marker system using digital techniques. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 590–596, 2005.

[75] N. Firasta, M. Buxton, P. Jinbo, K. Nasri, and S. Kuo. Intel avx: New frontiers in performance improvements and energy efficiency. *Intel white paper*, 19:20, 2008.

[76] Focus-Project Ballbot. Rezero. URL `http://www.rezero.ethz.ch/`. date retrieved: 2017-04-25.

[77] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22, 2014. doi: 10.1109/ICRA.2014.6906584.

[78] E. Foxlin and L. Naimark. Vis-tracker: a wearable vision-inertial self-tracker. In *Virtual Reality, 2003. Proceedings. IEEE*, March 2003.

[79] M. Frigerio, J. Buchli, D. G. Caldwell, and C. Semini. RobCoGen : a code generator for efficient kinematics and dynamics of articulated robots , based on Domain Specific Languages. *Journal of Software Engineering for Robotics*, 7(July):36–54, 2016.

[80] G. Garimella and M. Kobilarov. Towards model-predictive control for aerial pick-and-place. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4692–4697, 2015. doi: 10.1109/ICRA.2015.7139850.

[81] G. Garofalo, C. Ott, and A. Albu-Schaffer. On the closed form computation of the dynamic matrices and their differentiations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2364–2369, 2013. doi: 10.1109/IROS.2013.6696688.

[82] C. Gehring, S. Coros, M. Hutter, M. Bloesch, M. A. Hoepflinger, and R. Siegwart. Control of dynamic gaits for a quadrupedal robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3287–3292, 2013. doi: 10.1109/ICRA.2013.6631035.

[83] M. Giftthaler, F. Farshidian, T. Sandy, L. Stadelmann, and J. Buchli. Efficient Kinematic Planning for Mobile Manipulators with Non-holonomic Constraints Using Optimal Control. *arXiv preprint arXiv:1701.08051*, 2017.

[84] M. Giftthaler, **M. Neunert**, M. Stäuble, J. Buchli, and M. Diehl. A Family of Iterative Gauss-Newton Shooting Methods for Nonlinear Optimal Control. arXiv:1711.11006 [cs.SY], 2017.

[85] M. Giftthaler*, **M. Neunert***, M. Stäuble, M. Frigerio, C. Semini, and J. Buchli. Automatic differentiation of rigid body dynamics for optimal control and estimation. *Advanced Robotics*, 31(22):1225–1237, 2017. doi: 10.1080/01691864.2017.1395361. URL `https://doi.org/10.1080/01691864.2017.1395361`. (*these authors contributed equally to this work.).

[86] M. Giftthaler, T. Sandy, K. Dörfler, I. Brooks, M. Buckingham, G. Rey, M. Kohler, F. Gramazio, and J. Buchli. Mobile Robotic Fabrication at 1:1 scale: the In situ Fabricator. *arXiv preprint arXiv:1701.03573*, 2017.

[87] M. Giftthaler*, **M. Neunert***, M. Stäuble, and J. Buchli. The Control Toolbox - an open-source C++ library for robotics, optimal and model predictive control. `https://adrlab.bitbucket.io/ct`, 2018. arXiv:1801.04290 [cs.RO], (*these authors contributed equally to this work.).

[88] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization. *SIAM Journal on Optimization*, 12(4):979–1006, 2002. doi: 10.1137/S1052623499350013.

[89] R. Ginhoux, J. Gangloff, M. de Mathelin, L. Soler, M. Sanchez, and J. Marescaux. Beating heart tracking in robotic surgery using 500 hz visual servoing, model predictive control and an adaptive observer. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 274–279, 2004.

[90] C. Glocker and C. Studer. Formulation and preparation for numerical evaluation of linear complementarity systems in dynamics. *Multibody System Dynamics*, 13(4):447–463, 2005. doi: 10.1007/s11044-005-2519-6.

[91] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation.* Society for Industrial and Applied Mathematics, 2008.

[92] D. T. Griffith, J. D. Turner, and J. L. Junkins. Some applications of automatic differentiation to rigid, flexible, and constrained multibody dynamics. In *ASME International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE)*, pages 271–281, 2005.

[93] G. Guennebaud, B. Jacob, et al. Eigen v3. URL `http://eigen.tuxfamily.org`.

[94] E. Guizzo and E. Ackerman. Who Is SCHAFT, the Robot Company Bought by Google and Winner of the DRC? URL `http://spectrum.ieee.org/automaton/robotics/humanoids/schaft-robot-company-bought-by-google-darpa-robotics-challenge-winner`. date retrieved: 2017-04-25.

[95] E. Guizzo and E. Ackerman. The hard lessons of DARPA's robotics challenge. *IEEE Spectrum*, 52(8):11–13, 2015. doi: 10.1109/MSPEC.2015.7164385.

[96] N. Hansen. The CMA evolution strategy: A comparing review. *Studies in Fuzziness and Soft Computing*, 192:75–102, 2006. doi: 10.1007/11007937_4.

[97] P. Hebert, M. Bajracharya, J. Ma, N. Hudson, A. Aydemir, J. Reid, C. Bergh, J. Borders, M. Frost, M. Hagman, J. Leichty, P. Backes, B. Kennedy, P. Karplus, B. Satzinger, K. Byl, K. Shankar, and J. Burdick. Mobile manipulation and mobility as manipulation - Design and algorithms of RoboSimian. *Journal of Field Robotics*, 32(2):255–274, 2015. doi: 10.1002/rob.21566.

[98] M. Hehn and R. D'Andrea. Real-Time Trajectory Generation for Quadrocopters. *IEEE Transactions on Robotics*, 31(4):877–892, 2015. doi: 10.1109/TRO.2015.2432611.

[99] A. Herdt, H. Diedam, P.-B. Wieber, D. Dimitrov, K. Mombaur, and M. Diehl. Online Walking Motion Generation with Automatic Footstep Placement. *Advanced Robotics*, 24(5-6):719–737, 2010. doi: 10.1163/016918610X493552.

[100] H. Hertz. Ueber die Beruehrung fester elastischer Koerper. *Journal für die reine und angewandte Mathematik*, 92:156–171, 1882. doi: 10.1515/crll.1882.92.156.

[101] A. Herzog, N. Rotella, S. Mason, F. Grimminger, S. Schaal, and L. Righetti. Momentum Control with Hierarchical Inverse Dynamics on a Torque-Controlled Humanoid. *Autonomous Robots*, 40(3):473–491, 2014.

[102] A. Herzog, N. Rotella, S. Schaal, and L. Righetti. Trajectory generation for multi-contact momentum-control. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 874–880, 2015. doi: 10.1109/HUMANOIDS.2015.7363464.

[103] R. J. Hogan. Fast reverse-mode automatic differentiation using expression templates in C++. *ACM Transactions on Mathematical Software (TOMS)*, 40(4):26, 2014.

[104] A. Hornung, K. Wurm, M. Bennewitz, and C. Stachniss. OctoMap - An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[105] B. Houska, H. J. Ferreau, and M. Diehl. ACADO toolkit-An open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011. doi: 10.1002/oca.

[106] M. Hutter. *StarlETH & Co – design and control of legged robots with compliant actuation.* PhD thesis, ETH Zürich, 2013.

[107] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer, and M. Hoepflinger. Anymal - a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 38–44, Oct 2016.

[108] J. Hwangbo, C. Gehring, H. Sommer, R. Siegwart, and J. Buchli. ROCK*—Efficient black-box optimization for policy learning. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 535–540, 2014.

[109] K. L. Johanson. *Contact mechanics.* Cambridge University Press, 1985. doi: 10.1115/1.3261297.

[110] S. G. Johnson. The NLopt nonlinear-optimization package. URL `http://ab-initio.mit.edu/nlopt`. date retrieved: 2017-04-10.

[111] E. S. Jones and S. Soatto. Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *The International Journal of Robotics Research*, 30(4):407–430, 2011.

[112] S. Kajita, F. Kanehiro, K. Kaneko, Y. Kazuhito, and H. Hirukawa. The 3D Linear Inverted Pendulum Mode: A Simple Modeling for a Biped Walking Pattern Generation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 239–246, 2001. doi: 10.1109/IROS.2001.973365.

[113] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal. Learning, planning, and control for quadruped locomotion over challenging terrain. *The International Journal of Robotics Research*, 30(2):236–258, 2011. doi: 10.1177/0278364910388677.

[114] M. Kalakrishnan, S. Chitta, and E. Theodorou. STOMP: Stochastic trajectory optimization for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4569–4574, 2011.

[115] M. Kamel, K. Alexis, M. Achtelik, and R. Siegwart. Fast nonlinear model predictive control for multicopter attitude tracking on SO(3). In *IEEE Conference on Control Applications (CCA)*, pages 1160–1166, 2015.

[116] M. Kamel, M. Burri, and R. Siegwart. Linear vs Nonlinear MPC for Trajectory Tracking Applied to Rotary Wing Micro Aerial Vehicles. *arXiv preprint arXiv:1611.09240*, 2016.

[117] K. Kato, M. Billinghurst, and I. Poupyrev. ARToolkit user manual. Technical report, Human Interface Technology Lab, University of Washington, 2000.

[118] N. Keivan and G. Sibley. Realtime simulation-in-the-loop control for agile ground vehicles. In *Towards Autonomous Robotic Systems*, Lecture Notes in Computer Science. Springer, 2014.

[119] J. Kelly and G. S. Sukhatme. Visual-Inertial Sensor Fusion: Localization, Mapping and Sensor-to-Sensor Self-calibration. *International Journal of Robotics Research (IJRR)*, 30(1):56–79, 2011.

[120] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal of Robotics and Automation*, 3(1):43–53, 1987. doi: 10.1109/JRA.1987.1087068.

[121] A. Kleiner and C. Dornhege. Real-time localization and elevation mapping within urban search and rescue scenarios. *Journal of Field Robotics*, 24(8-9): 723–745, 2007. doi: 10.1002/rob.20208.

[122] K. H. Koch, K. Mombaur, and P. Soueres. Optimization-based walking generation for humanoid robot. In *IFAC Symposium on Robot Control*, pages 498–504, 2012.

[123] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard. Whole-body model-predictive control applied to the HRP-2 humanoid. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3346–3351, 2015. doi: 10.1109/IROS.2015.7353843.

[124] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154, 2004. doi: 10.1109/IROS.2004.1389727.

[125] V. Konnyukh. Robotics for mining. *Mineral Resources Engineering*, 11(1): 73–88, 2002. doi: 10.1142/S0950609802000860.

[126] T. Koolen, J. Smith, G. Thomas, S. Bertrand, J. Carff, N. Mertins, D. Stephen, P. Abeles, J. Englsberger, S. Mccrory, et al. Summary of team IHMC's virtual robotics challenge entry. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 307–314, 2013.

[127] K. Koyanagi, H. Hirukawa, S. Hattori, M. Morisawa, S. Nakaoka, K. Harada, and S. Kajita. A pattern generator of humanoid robots walking on a rough terrain using a handrail. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2617–2622, 2008. doi: 10.1109/IROS.2008.4650686.

[128] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots*, 40(3):429–455, 2016. doi: 10.1007/s10514-015-9479-3.

[129] V. Kumar, E. Todorov, and S. Levine. Optimal control with learned local models: Application to dexterous manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–383, 2016. doi: 10.1109/ICRA.2016.7487156.

[130] S. LaValle. *Planning algorithms.* Cambridge University Press, 2006. doi: 10.1017/CBO9780511546877.

[131] S. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[132] J. R. Leal. CppADCodeGen. `http://github.com/joaoleal/CppADCodeGen`, 2017. date retrieved: 2017-04-25.

[133] T. Lee, M. Leoky, and N. McClamroch. Geometric tracking control of a quadrotor UAV on SE(3). In *IEEE Conference on Decision and Control (CDC)*, pages 5420–5425, 2010. doi: 10.1109/CDC.2010.5717652.

[134] M. Li and A. I. Mourikis. High-precision, consistent EKF-based visual-inertial odometry. *The International Journal of Robotics Research (IJRR)*, 32(6), 2013.

[135] H. Lim and Y.-S. Lee. Real-time single camera SLAM using fiducial markers. In *ICROS-SICE International Joint Conference*, pages 177–182, 2009.

[136] T. Lin and J. Arora. Differential dynamic programming technique for constrained optimal control. *Computational mechanics*, 9(1):27–40, 1991.

[137] A. Liniger, A. Domahidi, and M. Morari. Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods*, 36 (5):628–647, 2014.

[138] H. Liu, Q. Sun, and T. Zhang. Hierarchical RRT for humanoid robot footstep planning with multiple constraints in complex environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3187–3194, 2012. doi: 10.1109/IROS.2012.6385836.

[139] K. Liu. Dynamic Animation and Robotics Toolkit, 2014. URL `http://dartsim.github.io`.

[140] M. Maidi, F. Ababsa, and M. Mallem. Vision-inertial tracking system for robust fiducials registration in augmented reality. In *IEEE Symposium on Computational Intelligence for Multimedia Signal and Vision Processing (CIMSVP)*, pages 83–90, 2009.

[141] Z. Manchester and S. Kuindersma. Derivative-free trajectory optimization with unscented dynamic programming. In *IEEE Conference on Decision and Control (CDC)*, pages 3642–3647, 2016. doi: 10.1109/CDC.2016.7798817.

[142] P. Martin and E. Salaun. The true role of accelerometer feedback in quadrotor control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1623–1629, 2010.

[143] C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell, and C. Semini. Hierarchical planning of dynamic movements without scheduled contact sequences. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4636–4641, 2016. doi: 10.1109/ICRA.2016.7487664.

[144] C. Mastalli, M. Focchi, I. Havoutis, A. Radulescu, S. Calinon, J. Buchli, D. G. Caldwell, and C. Semini. Trajectory and Foothold Optimization using Low-Dimensional Models for Rough Terrain Locomotion. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[145] MathWorks Inc. Matlab & simulink software, 2017. URL `http://www.mathworks.com`. date retrieved: 2017-04-25.

[146] Maxima. Maxima, a computer algebra system. `http://maxima.sourceforge.net`, 2017. date retrieved: 2017-04-25.

[147] D. Mayne. A Second-order Gradient Method for Determining Optimal Trajectories of Non-linear Discrete-time Systems. *International Journal of Control*, 3(1):85–95, 1966. doi: 10.1080/00207176608921369.

[148] A. McBratney, B. Whelan, T. Ancev, and J. Bouma. Future directions of precision agriculture. *Precision Agriculture*, 6(1):7–23, 2005. doi: 10.1007/s11119-005-0681-8.

[149] J. R. Medina and S. Hirche. Uncertainty-dependent optimal control for robot control considering high-order cost statistics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3995–4002, 2015. doi: 10.1109/IROS.2015.7353940.

[150] D. Mellinger. Trajectory Generation and Control for Quadrotors. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2520–2525, 2011. doi: 10.1109/ICRA.2011.5980409.

[151] O. Michel. WebotsTM: Professional mobile robot simulation. *arXiv preprint arXiv:cs/0412052*, 2004.

[152] M. Mistry, J. Buchli, and S. Schaal. Inverse dynamics control of floating base systems using orthogonal decomposition. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3406–3412, 2010. doi: 10.1109/ROBOT.2010.5509646.

[153] D. Mitrovic, S. Klanke, and S. Vijayakumar. Adaptive optimal feedback control with learned internal dynamics models. *From Motor Learning to Interaction Learning in Robots*, 264:65–84, 2010. doi: 157.2924.

[154] K. Mombaur. Using optimization to create self-stable human-like running. *Robotica*, 27(03):321, 2009. doi: 10.1017/S0263574708004724.

[155] I. Mordatch, M. de Lasa, and A. Hertzmann. Robust physics-based locomotion using low-dimensional planning. *ACM Transactions on Graphics (TOG)*, 29 (4):71, 2010.

[156] I. Mordatch, E. Todorov, and Z. Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics*, 31 (4):1–8, 2012. doi: 10.1145/2185520.2185539.

[157] I. Mordatch, K. Lowrey, and E. Todorov. Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2015-Decem, pages 5307–5314, 2015. doi: 10.1109/IROS.2015.7354126.

[158] J. J. Moreau. Unilateral Contact and Dry Friction in Finite Freedom Dynamics. In *Nonsmooth mechanics and Applications*, pages 1–82. Springer, 1988. doi: 10.1007/978-3-7091-2624-0_1.

[159] A. Mourikis, S. Roumeliotis, et al. A multi-state constraint Kalman filter for vision-aided inertial navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3565–3572, 2007.

[160] R. R. Murphy, S. Tadokoro, D. N. Jacoff, P. Fiorini, H. Choset, and Aydan M Erkmen. Search and Rescue Robotics. In *Handbook of Robotics*, pages 1151–1173. Springer, 2008. doi: 10.1007/978-3-540-30301-5_51.

[161] R. M. Murray, M. Rathinam, and W. Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. In *ASME Mechanical Engineering Congress and Expo*, pages 1–9, 1995.

[162] **M. Neunert**, F. Farshidian, and J. Buchli. Adaptive real-time nonlinear model predictive motion control. In *IROS 2014 Workshop on Machine Learning in Planning and Control of Robot Motion*, 2014.

[163] **M. Neunert**, M. Bloesch, and J. Buchli. An Open Source, Fiducial Based, Visual-Inertial Motion Capture System. In *19th International Conference on Information Fusion (FUSION)*, pages 1523–1530, 2016.

[164] **M. Neunert**, T. Boaventura, J. Buchli, and A. Dynamics. Why off-the-shelf physics simulators fail in evaluating feedback controller performance - a case study for quadrupedal robots. *Advances in Cooperative Robotics: Proceedings of the 19th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines, CLAWAR 2016*, pages 464–472, 2016.

[165] **M. Neunert**, C. De Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli. Fast Nonlinear Model Predictive Control for Unified trajectory optimization and tracking. In *IEEE International Conference on Robotics and Automation*, pages 1398–1404, 2016. doi: 10.1109/ICRA.2016.7487274.

[166] **M. Neunert**, F. Farshidian, and J. Buchli. Efficient whole-body trajectory optimization using contact constraint relaxation. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 43–48, 2016. doi: 10.1109/HUMANOIDS.2016.7803252.

[167] **M. Neunert**, M. Giftthaler, M. Frigerio, C. Semini, and J. Buchli. Fast Derivatives of Rigid Body Dynamics for Control, Optimization and Estimation. In *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 91–97, 2016. doi: 10.1109/SIMPAR.2016.7862380.

[168] **M. Neunert**, F. Farshidian, A. W. Winkler, and J. Buchli. Trajectory optimization through contacts and automatic gait discovery for quadrupeds. *IEEE Robotics and Automation Letters (RA-L)*, 2(3):1502–1509, 2017. doi: 10.1109/LRA.2017.2665685.

[169] **M. Neunert**, M. Stäuble, M. Giftthaler, C. D. Bellicoso, J. Carius, C. Gehring, M. Hutter, and J. Buchli. Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds. In *ArXiv preprint*, 2017. To be published in Robotics and Automation Letters (RA-L) 2018 and to be presented at the IEEE International Conference on Robotics and Automation (ICRA) 2018.

[170] U. Nagarajan. Dynamic Constraint-based Optimal Shape Trajectory Planner for Shape-Accelerated Underactuated Balancing Systems. In *Robotics: Science and Systems*, pages 243–250, 2010.

[171] U. Nagarajan, G. Kantor, and R. L. Hollis. Trajectory planning and control of an underactuated dynamically stable single spherical wheeled mobile robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3743–3748, 2009. doi: 10.1109/ROBOT.2009.5152624.

[172] J. Nakanishi, M. Mistry, and S. Schaal. Inverse dynamics control with floating base and constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1942–1947, 2007.

[173] M. Neunert, M. Giftthaler, and J. Buchli. An Open-Source C++ Library for Robotics and Optimal Control. `http://www.adrl.ethz.ch/software`, ETH Zürich, Switzerland, 2017.

[174] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Y. Siegwart. A synchronized visual-inertial sensor system with FPGA preprocessing for accurate real-time SLAM. In *IEEE International Conference on Robotics and Automation*, 2014.

[175] G. Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart. Fusion of IMU and vision for absolute scale estimation in monocular SLAM. *Journal of Intelligent & Robotic systems*, 61(1-4), 2011.

[176] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407, 2011.

[177] OptiTrack. Motion Capture Systems. URL `http://optitrack.com/`. date retrieved: 2017-04-24.

[178] D. E. Orin, A. Goswami, and S. H. Lee. Centroidal dynamics of a humanoid robot. *Autonomous Robots*, 35(2-3):161–176, 2013. doi: 10.1007/s10514-013-9341-4.

[179] K. Paes, W. Dewulf, K. V. Elst, K. Kellens, and P. Slaets. Energy efficient trajectories for an industrial ABB robot. *Procedia CIRP*, 15:105 – 110, 2014.

[180] J. F. O. D. O. Pantoja and D. Q. Mayne. A sequential quadratic programming algorithm for discrete optimal control problems with control inequality constraints. In *Proceedings of the 28th IEEE Conference on Decision and Control,*, 1989.

[181] D. Pardo, **M. Neunert**, A. W. Winkler, and J. Buchli. Projection based whole body motion planning for legged robots. *arXiv preprint arXiv:1510.01625*, 2015.

[182] D. Pardo, L. Moller, **M. Neunert**, A. W. Winkler, and J. Buchli. Evaluating Direct Transcription and Nonlinear Optimization Methods for Robot Motion Planning. *IEEE Robotics and Automation Letters (RA-L)*, 1(2):946–953, 2016. doi: 10.1109/LRA.2016.2527062.

[183] N. Perrin, O. Stasse, F. Lamiraux, Y. J. Kim, and D. Manocha. Real-time footstep planning for humanoid robots among 3D obstacles using a hybrid bounding box. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 977–982, 2012. doi: 10.1109/ICRA.2012.6224555.

[184] S. Peters and J. Hsu. Comparison of rigid body dynamic simulators for robotic simulation in gazebo. In *ROSCon*, 2014.

[185] Q. C. Pham. A general, fast, and robust implementation of the time-optimal path parameterization algorithm. *IEEE Transactions on Robotics*, 30(6): 1533–1540, 2014. doi: 10.1109/TRO.2014.2351113.

[186] F. Pomerleau, F. Colas, and R. Siegwart. A Review of Point Cloud Registration Algorithms for Mobile Robotics. *Foundations and Trends in Robotics*, 4 (1):1–104, 2015. doi: 10.1561/2300000035.

[187] B. Ponton, S. Schaal, and L. Righetti. Risk sensitive nonlinear optimal control with measurement uncertainty. *arXiv preprint arXiv:1605.04344*, 2016.

[188] M. Posa, C. Cantu, and R. Tedrake. A Direct Method for Trajectory Optimization of Rigid Bodies Through Contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014. doi: 10.1177/0278364913506757.

[189] M. Posa, S. Kuindersma, and R. Tedrake. Optimization and stabilization of trajectories for constrained dynamical systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1366–1373, 2016. doi: 10.1109/ICRA.2016.7487270.

[190] A. Prabhakar, K. Flaßkamp, and T. D. Murphey. Symplectic integration for optimal ergodic control. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 2594–2600, Dec 2015.

[191] J. Pratt, C.-m. Chew, A. Torres, P. Dilworth, and G. Pratt. Virtual Model Control: An Intuitive Approach. *The International Journal of Robotics Research*, 20(2):129–143, 2001.

[192] PTI Phoenix Technologies. VisualEyez III. URL `http://www.ptiphoenix.com/`. date retrieved: 2017-04-24.

[193] Z. Qiu, A. Escande, A. Micaelli, and T. Robert. A hierarchical framework for realizing dynamically-stable motions of humanoid robot in obstacle-cluttered environments. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 867–874, 2012. doi: 10.1109/HUMANOIDS.2012.6651622.

[194] R. Quirynen, M. Vukov, M. Zanon, and M. Diehl. Autogenerating Microsecond Solvers for Nonlinear MPC: a Tutorial Using ACADO Integrators. *Optimal Control Applications and Methods*, 2014.

[195] A. Radulescu, I. Havoutis, D. G. Caldwell, and C. Semini. Whole-body Trajectory Optimization for Non-periodic Dynamic Motions on Quadrupedal Systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[196] D. Ralph and S. J. Wright. Some properties of regularization and penalization schemes for mpecs. *Optimization Methods and Software*, 19(5):527–556, 2004.

[197] A. V. Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135(1):497–528, 2009. doi: 10.1515/jnum-2014-0003.

[198] J. Rekimoto and Y. Ayatsuka. CyberCode: designing augmented reality environments with visual tags. In *Designing augmented reality environments (DARE)*, pages 1–10, 2000.

[199] L. Righetti and S. Schaal. Quadratic programming for inverse dynamics with optimal distribution of contact forces. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 538–543, 2012. doi: 10.1109/HUMANOIDS.2012.6651572.

[200] L. Righetti, M. Mistry, J. Buchli, and S. Schaal. Inverse Dynamics Control of Floating-Base Robots With External Contraints: an Unified View. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1085–1090, 2011.

[201] L. A. Rodriguez and A. Sideris. A Sequential Linear Quadratic Approach for Constrained Nonlinear Optimal Control. In *American Control Conference (ACC)*, pages 1470–1475, 2011. doi: 10.0/Linux-x86_64.

[202] E. Rohmer, S. P. N. Singh, and M. Freese. V-REP : a Versatile and Scalable Robot Simulation Framework. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1321–1326, 2013.

[203] N. Rotella, M. Bloesch, L. Righetti, and S. Schaal. State estimation for a humanoid robot. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 952–958, 2014. doi: 10.1109/IROS.2014.6942674.

[204] G. Schultz and K. Mombaur. Modeling and optimal control of human-like running. *IEEE/ASME Transactions on Mechatronics*, 15(5):783–792, 2010. doi: 10.1109/TMECH.2009.2035112.

[205] C. Semini and N. Tsagarakis. Design of HyQ – a hydraulically and electrically actuated quadruped robot. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 225(6): 831–849, 2011.

[206] L. Sentis and O. Khatib. Control of Free-Floating Humanoid Robots Through Task Prioritization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 18–23, 2005. doi: 10.1109/ROBOT.2005.1570361.

[207] C. Sferrazza, D. Pardo, and J. Buchli. Numerical search for local (partial) differential flatness. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3640–3646, 2016. doi: 10.1109/IROS.2016.7759536.

[208] M. A. Sherman, A. Seth, and S. L. Delp. Simbody: Multibody dynamics for biomedical research. In *Procedia IUTAM*, volume 2, pages 241–261, 2011. doi: 10.1016/j.piutam.2011.04.023.

[209] M. Shomin and R. Hollis. Differentially flat trajectory generation for a dynamically stable mobile robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4467–4472, 2013. doi: 10.1109/ICRA.2013.6631211.

[210] A. Sideris and J. E. Bobrow. A Fast Sequential Linear Quadratic Algorithm for Solving Unconstrained Nonlinear Optimal Control Problems. In *American Control Conference*, pages 2275–2280, 2005.

[211] A. Sideris and J. E. Bobrow. An efficient sequential linear quadratic algorithm for solving nonlinear optimal control problems. *IEEE Transactions on Automatic Control*, 50(12):2043–2047, 2005.

[212] R. Smith. Open Dynamics Engine. URL `http://ode.org`. date retrieved: 2017-04-24.

[213] J. Sola, T. Vidal-Calleja, J. Civera, and J. M. M. Montiel. Impact of landmark parametrization on monocular EKF-SLAM with points and lines. *International Journal of Computer Vision*, 97(3), 2012.

[214] K. Sreenath, T. Lee, and V. Kumar. Geometric control and differential flatness of a quadrotor UAV with a cable-suspended load. In *IEEE Conference on Decision and Control*, pages 2269–2274, 2013. doi: 10.1109/CDC.2013.6760219.

[215] F. Stulp and O. Sigaud. Path Integral Policy Improvement with Covariance Matrix Adaptation. *International Conference on Machine Learning (ICML)*, pages 281–288, 2012.

[216] J. Tan, K. Liu, and G. Turk. Stable proportional-derivative controllers. *IEEE Computer Graphics and Applications*, 31(4):34–44, 2011.

[217] Y. Tassa, T. Erez, and E. Todorov. Synthesis of Robust Behaviors through Online Trajectory Optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4906–4913, 2012.

[218] Y. Tassa, N. Mansard, and E. Todorov. Control-limited differential dynamic programming. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1168–1175, 2014. doi: 10.1109/ICRA.2014.6907001.

[219] R. Tedrake. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, . URL `http://drake.mit.edu`. date retrieved: 2017-04-10.

[220] R. Tedrake. MIT Lecture 6.832 Underactuated Robotics, . URL `https://www.youtube.com/watch?v=ZTL13eiA4vo&feature=youtu.be`. date retrieved: 2017-04-26.

[221] E. Theodorou, J. Buchli, and S. Schaal. A Generalized Path Integral Control Approach to Reinforcement Learning. *Journal of Machine Learning Research*, 11(Nov):3137–3181, 2010.

[222] E. Todorov. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the American Control Conference*, pages 300–306, 2005. doi: 10.1109/ACC.2005.1469949.

[223] E. Todorov. A convex, smooth and invertible contact model for trajectory optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1071–1076, 2011. doi: 10.1109/ICRA.2011.5979814.

[224] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

[225] N. Tsagarakis, D. Caldwell, A. Bicchi, F. Negrello, M. Garabini, W. Choi, L. Baccelliere, V. Loc, J. Noorden, M. Catalano, M. Ferrati, L. Muratore, A. Margan, L. Natale, E. Mingo, H. Dallali, A. Settimi, A. Rocchi, V. Varricchio, L. Pallottino, C. Pavan, A. Ajoudani, J. Lee, P. Kryczka, and D. Kanoulas. WALK-MAN: A High Performance Humanoid Platform for Realistic Environments. *Journal of Field Robotics*, 2016.

[226] S. Vasudevan, S. Gachter, M. Berger, and R. Siegwart. Cognitive Maps for Mobile Robots: An Object based Approach. *Robotics and Autonomous Systems*, 55(5):359–371, 2007.

[227] VICON. Motion Capture Systems. URL `https://www.vicon.com/`. date retrieved: 2017-04-24.

[228] M. Vukobratović and B. Borovac. Zero-Moment Point – Thirty Five Years of its Life. *International Journal of Humanoid Robotics*, 01(01):157–173, 2004. doi: 10.1142/S0219843604000083.

[229] M. Vukov, A. Domahidi, H. J. Ferreau, M. Morari, and M. Diehl. Auto-generated Algorithms for Nonlinear Model Predicitive Control on Long and on Short Horizons. In *IEEE Conference on Decision and Control (CDC)*, pages 5113–5118, 2013.

[230] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. doi: 10.1007/s10107-004-0559-y.

[231] A. Walther and A. Griewank. Getting started with adol-c. Technical report, Chapman-Hall CRC Computational Science, 2012.

[232] M. Wasielica and D. Belter. RRT-based Motion Planner and Balance Controller for a Biped Robot. In *Advances in Cooperative Robotics: Proceedings of the 19th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines, CLAWAR 2016*, pages 404–411, 2016.

[233] D. J. Webb and J. van den Berg. Kinodynamic RRT*: Optimal Motion Planning for Systems with Linear Differential Constraints. In *IEEE Internation Conference on Robotics and Automation (ICRA)*, pages 5054–5061, 2013.

[234] B. H. Wilcox, T. Litwin, J. Biesiadecki, J. Matthews, M. Heverly, J. Morrison, J. Townsend, N. Ahmad, A. Sirota, and B. Cooper. CHIMP. *Journal of Field Robotics*, 24(5):421–434, 2007. doi: 10.1002/rob.

[235] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou. Aggressive driving with model predictive path integral control. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2016-June, pages 1433–1440, 2016. doi: 10.1109/ICRA.2016.7487277.

[236] A. W. Winkler, C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell, and C. Semini. Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5148–5154, 2015. doi: 10.1109/ICRA.2015.7139916.

[237] A. W. Winkler, F. Farshidian, **M. Neunert**, D. Pardo, and J. Buchli. Online Walking Motion and Foothold Optimization for Quadruped Locomotion. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[238] A. W. Winkler, F. Farshidian, M. Neunert, D. Pardo, and J. Buchli. Online Walking Motion and Foothold Optimization for Quadruped Locomotion. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[239] J. Wojtusch, J. Kunz, and O. von Stryk. MBSlib - An Efficient Multibody Systems Library for Kinematics and Dynamics Simulation, Optimization and Sensitivity Analysis. *IEEE Robotics and Automation Letters*, 1(2):954–960, 2016.

[240] Wolfram Research. Mathematica software. `http://www.wolfram.com`, 2017. date retrieved: 2017-04-25.

[241] Z. Xia, G. Chen, J. Xiong, Q. Zhao, and K. Chen. A random sampling-based approach to goal-directed footstep planning for humanoid robots. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 168–173, 2009. doi: 10.1109/AIM.2009.5230019.

[242] S. J. Yi, S. McGill, L. Vadakedathu, Q. He, I. Ha, J. Han, H. Song, M. Rouleau, D. Hong, and D. D. Lee. THOR-OP humanoid robot for DARPA Robotics Challenge Trials 2013. In *International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 359–363, 2014. doi: 10.1109/URAI.2014.7057369.

[243] S. You and U. Neumann. Fusion of vision and gyro tracking for robust augmented reality registration. In *IEEE Virtual Reality*, pages 71–78, 2001.

[244] K. Yunt and C. Glocker. Trajectory Optimization of Mechanical Hybrid Systems Using SUMT. In *IEEE International Workshop on Advanced Motion Control*, pages 665–671, 2005.

[245] K. Yunt and C. Glocker. Modeling and Optimal Control of Hybrid Rigid-body Mechanical Systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 614–627, 2007.

[246] S. Zickler, T. Laue, O. Birbach, M. Wongphati, and M. Veloso. Ssl-vision: The shared vision system for the robocup small size league. In *RoboCup 2009: Robot Soccer World Cup XIII*. Springer, 2010.

[247] D. Zimmermann, S. Coros, Y. Ye, R. W. Sumner, and M. Gross. Hierarchical planning and control for complex motor tasks. In *14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 73–81, 2015. doi: 10.1145/2786784.2786795.

[248] M. Zucker, N. Ratliff, M. Stolle, J. Chestnutt, J. A. Bagnell, C. G. Atkeson, and J. Kuffner. Optimization and learning for rough terrain legged locomotion. *The International Journal of Robotics Research*, 30(2):175–191, 2011. doi: 10.1177/0278364910392608.

Curriculum Vitae

Michael Peter Neunert
born August 04, 1988
citizen of Germany

Education

2012–2017	*ETH Zurich, Switzerland* Doctoral studies at the Agile & Dexterous Robotics Lab Supervised by Prof. Dr. Jonas Buchli
2010–2012	*ETH Zurich, Switzerland* Master of Science in Mechanical Engineering Supervised by Prof. Dr. Roland Siegwart Master thesis: “Localization system for an autonomous electric vehicle” Semester thesis: “Development of a highly dynamic BLDC motor controller”“
2006–2010	*ETH Zurich, Switzerland* Bachelor of Science in Mechanical Engineering Bachelor thesis: “Obstacle management and interaction of a Ballbot”
1998–2006	*Gymnasium am Hoptbühl, Villingen-Schwenningen, Germany* Allgemeine Hochschulreife (German A-levels)

Scientific Projects

2011–2012	*Project “Realize”, Tongji University, Shanghai, China* Design and small batch production of robotics parts
2009–2010	*Project “Rezero”, ETH Zurich, Switzerland* Development of a highly agile ball balancing robot (“ballbot”) Co-founder of the project, head of software development

Internships

2010	*Product Management, ALSTOM, Baden, Switzerland* Market analyses in the gas turbine sector
2007	*Siemens VDO, Villingen-Schwenningen, Germany* Technical training on metal working processes
2006	*Turbo Machines Group, ALSTOM, Baden, Switzerland* Project lead in developing an approval process for external publications and presentations

Academic and Social Activities

2012	*Teaching Assistant, IMES, ETH Zurich, Switzerland* Teaching and technical support for "Tools Course Computational Mathematics"
2010	*Teaching Assistant, IDSC, ETH Zurich, Switzerland* Teaching of practice lesson in "Control Systems I" course
2010–2011	*Board Member, AMIV Association, ETH Zurich, Switzerland* Co-responsible for operational activities Head of university politics team with ten members

Academic Awards

2016	*Best Paper Award, SIMPAR 2016*
2016	*Industrial Robot Innovation Award, CLAWAR 2016* Awarded for "Practical innovations in the field of robotics"
2016	*Runner-up for the Best PhD Paper Award, NCCR Robotics*
2012	*Johann Puch Innovation Award, Magna Steyr, Austria* Awarded for the best diploma thesis in automotive engineering
2010	*Siemens PLM Award 2010* Awarded to the "Rezero" team for the efficient usage of CAD software and its integration with control and system modelling