Moritz Geilinger

Differentiable Dynamics and Motion Synthesis for Legged Robots

Diss. ETH No. 27946

MORITZ GEILINGER

DIFFERENTIABLE DYNAMICS AND MOTION SYNTHESIS FOR LEGGED ROBOTS

DISS. ETH NO. 27946

DIFFERENTIABLE DYNAMICS AND MOTION SYNTHESIS FOR LEGGED ROBOTS

A dissertation submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

MORITZ GEILINGER
MSc, Dipl., Eidgenössisches Polytechnikum

born on 29 January 1988
citizen of Zurich, Switzerland

accepted on the recommendation of

Prof. Dr. Stelian Coros, examiner
Prof. Dr. Paul Kry, co-examiner
Prof. Dr. Kenny Erleben, co-examiner
Prof. Dr. Auke Ijspeert, co-examiner

2021

Moritz Geilinger: *Differentiable Dynamics and Motion Synthesis for Legged Robots* , © 2021

To my family

ABSTRACT

The generation of agile and dynamic motions for legged robots has long been of interest in the fields of computer graphics and robotics. However, planning motions to control these underactuated, nonlinear dynamical systems has proven to be a difficult problem. Building on recent advances in differentiable physical models and trajectory optimization, this thesis presents several approaches to synthesizing motion controls for robots with legs and wheels, as well as compliant robots.

We begin by introducing a computational framework for motion generation of legged-wheeled robots. The user can easily design a robotic creature with an arbitrary arrangement of legs, motor joints, and various types of end effectors, such as point feet, actuated and unactuated wheels. Once the robot's morphology is determined, the user can create and edit motion targets, such as way points, using an interactive tool, while our trajectory optimization method generates physically valid motion trajectories. Finally, we fabricate prototypes designed with our system and show that the generated motions can be applied to the real world.

Next, we extend our system with a warm start technique that dramatically improves the convergence rate of the trajectory optimization. Using our computational framework, we design and build an agile robot with legs and wheels, AgileBot, which can be equipped with actuated wheels, roller-blades or ice-skates. Our trajectory optimization generates various agile motions, such as roll-walking, swizzling or skating, which are executed on the physical prototype. Finally, we use our system to generate several dynamic motions as reference trajectories for feedback control of a legged-wheeled robot.

Lastly, we introduce a differentiable physics engine capable of handling frictional contact for rigid and deformable objects in a unified framework. We combine a smoothed contact model with implicit time integration and sensitivity analysis to analytically compute derivatives with respect to the simulation parameters. We use our differentiable simulation to perform trajectory optimization that accounts for the full dynamics of a legged robot with compliant actuators and soft feet. We also demonstrate applications of our differentiable simulator to parameter estimation for deformable objects, motion planning for robot manipulations, and efficient self-supervised learning of control policies.

ZUSAMMENFASSUNG

Die Erzeugung von agilen und dynamischen Bewegungen für Roboter mit Beinen ist seit langem von Interesse in der Computergrafik und Robotik. Die Planung von Bewegungen zur Steuerung dieser nichtlinearen dynamischen Systeme hat sich jedoch als ein schwieriges Problem erwiesen. Aufbauend auf den jüngsten Fortschritten bei differenzierbaren physikalischen Modellen und der Trajektorienoptimierung werden in dieser Arbeit mehrere Ansätze zur Synthese von Bewegungen für Roboter mit Beinen und Rädern sowie für Roboter mit nachgiebigen Komponenten vorgestellt.

Wir stellen zuerst unser rechnergestütztes System vor, das Bewegungen für Roboter mit Beinen und Rädern generiert. Der Benutzer kann auf einfache Weise ein Roboter mit einer beliebigen Anordnung von Beinen, Motoren und Füssen, sowie angetriebene und freilaufende Rädern, entwerfen. Sobald die Morphologie des Roboters festgelegt ist, kann der Benutzer mithilfe einer interaktiven Software Bewegungsziele festlegen und bearbeiten, während unsere Methode zur Trajektorienoptimierung physikalisch valide Bewegungstrajektorien erzeugt. Schließlich stellen wir die mit unserem System entworfenen Prototypen her und zeigen in Experimenten, dass die erzeugten Bewegungen in der realen Welt angewendet werden können.

Als Nächstes erweitern wir unser System mit einer sogenannten ”warm start”Methode, die die Konvergenzrate der Trajektorienoptimierung drastisch verbessert. Unter Verwendung unseres computergestützten Systems entwerfen und bauen wir einen agilen Roboter mit Beinen und Rädern, AgileBot, der mit angetriebenen und freilaufenden Rädern, sowie Schlittschuhen ausgestattet werden kann. Unsere Optimierung generiert verschiedene agile Bewegungen, wie z.B. rollen und gehen, ßwizzlingöder skaten, die alle auf dem physischen Prototyp gezeigt werden. Schließlich verwenden wir unser System, um mehrere dynamische Bewegungen als Referenztrajektorien für die aktive Regelung eines Roboters mit Beinen und Rädern zu erzeugen.

Zuletzt stellen wir eine differenzierbare Physik-Engine vor, die in der Lage ist, Reibungskontakt für starre und deformierbare Objekte in einem einheitlichen Rahmen zu behandeln. Wir kombinieren ein geglättetes Kontaktmodell mit impliziter Zeitintegration und Sensitivitätsanalyse, um Ableitungen in Bezug auf die Simulationsparameter analytisch zu berechnen. Wir verwenden unsere differenzierbare Simulation, um Trajektorienoptimierung

durchzuführen, die die gesamte Dynamik eines Roboters mit Beinen, nachgiebigen Aktoren und weichen Füßen berücksichtigt. Wir demonstrieren auch Anwendungen unseres differenzierbaren Simulators zur Bestimmung von Parametern für deformierbare Objekte, zur Planung von Bewegungen für Robotermanipulationen und zum effizienten selbstüberwachten Lernen von Steuerungsstrategien.

ACKNOWLEDGEMENTS

This thesis would not have been possible without the support and help of many people.

I am deeply grateful to my PhD supervisor Prof. Dr. Stelian Coros, who taught me many invaluable things. He has guided me through the PhD journey with great vision, motivating me with his passion for solving hard problems, while giving me the freedom to explore new topics and flexibly balance work and family. I truly enjoyed the emotional rollercoaster every time we submitted a paper and always felt unconditionally supported.

Thank you, Prof. Dr. Bernhard Thomaszewski, for introducing me to the field of computer graphics and computational fabrication. I still vividly remember your awesome lecture on physics-based animation and the semester thesis I did under your supervision. Your passion for your field infected me, and I have not recovered from that to this day. Thank you for encouraging me to pursue a PhD.

I would also like to express my gratitude to Peter Kaufmann, Gerhard Röthlin, Max Grosse and Michele d'Ambroggio for sharing their engineering knowledge with me, which proved to be extremely useful during my doctoral studies. I thank Prof. Markus Gross for introducing me to the intersection of computer graphics and industry. Many thanks to Prof. Paul Kry and Prof. Eitan Grinspun, who generously took the time to work together to solve various problems I had in my master's thesis. It was great to see experienced academics tackle a problem and has inspired me to help others whenever possible

Thank you to my co-authors Prof. Roi Poranne, Ruta Desai, David Hahn, Jonas Zehnder and Moritz Bächer. I was fortunate to have had the opportunity to work with you. Thank you, Sebastian Winberg, for all the hard work you put into our ambitious projects. It was a lot of fun working with you, traveling and standing next to our robots.

Thank you to Jim, Vittorio, all the Simons, Miguel, Guirec, Dongho, Florian, Liyue, Nuria, Flavio, and Oliver. It was great having you as labmates and sharing our successes and failures.

Thank you to all my friends. Maybe I have time to have a beer now.

Last but not least, thank you to my wife Seraina for all the unconditional support, understanding and love for my strange work. Thanks to my mother, father, sisters and children for being there for me.

CONTENTS

List of Figures xvi
List of Tables xviii
1 INTRODUCTION 1
- 1.1 Motivation 1
- 1.2 State of the Art 3
- 1.3 Thesis Overview 5

2 MOTION GENERATION FOR LEGGED-WHEELED ROBOTS 7
- 2.1 Introduction 8
  - 2.1.1 Contributions 9
- 2.2 Related Work 9
- 2.3 Overview 10
- 2.4 Motion Generation 12
  - 2.4.1 Optimization Model 12
  - 2.4.2 Numerical Solution 18
  - 2.4.3 Further Analysis and Optimization Speedup 19
- 2.5 Design Optimization 21
  - 2.5.1 Motivation 21
  - 2.5.2 Technical solution 22
  - 2.5.3 Manual Design Mode 23
  - 2.5.4 Semi-Automatic Design Mode 24
  - 2.5.5 Automatic Design Optimization 24
- 2.6 Results 26
  - 2.6.1 Wheeled robots 26
  - 2.6.2 Interactive Design 29
  - 2.6.3 Design optimization 29
  - 2.6.4 Fabrication 29
- 2.7 Discussion 31

3 DESIGN AND FABRICATION OF LEGGED-WHEELED ROBOTS 35
- 3.1 Introduction 35
  - 3.1.1 Contribution 36
- 3.2 Related Work 37
- 3.3 Warm-Start Routine for Motion Generation 39
- 3.4 Analysis 42
  - 3.4.1 Evaluation of Warm-Start Routine 42

3.4.2 Morphology exploration for hybrid legged/wheeled robots . . . 43
3.5 Fabricated Results . . . 44
3.5.1 From conceptual designs to physical prototypes . . . 44
3.5.2 Robot Designs . . . 45
3.5.3 Performance Evaluation . . . 47
3.6 Motion Generation for Anymal on Wheels . . . 48
3.6.1 Overview of Feedback Control System . . . 49
3.6.2 Evaluation . . . 50
3.7 Discussion . . . 52
4 DIFFERENTIABLE PHYSICS SIMULATION FOR MOTION SYNTHESIS 55
4.1 Introduction . . . 56
4.1.1 Contributions . . . 57
4.2 Related work . . . 58
4.2.1 Frictional Contact Dynamics . . . 58
4.2.2 Inverse Contact . . . 59
4.2.3 Differentiable Simulation . . . 59
4.3 Differentiable Multi-Body Dynamics: Preliminaries . . . 60
4.3.1 Implicit time-stepping for multi-body systems . . . 60
4.3.2 Simulation Derivatives . . . 62
4.4 Differentiable Frictional Contact Model . . . 63
4.4.1 Sequential Quadratic Programming . . . 65
4.4.2 Penalty methods . . . 66
4.4.3 Hybrid method . . . 67
4.4.4 Summary and evaluation . . . 69
4.5 Internal and External Forces in Generalized Coordinates . . . 73
4.5.1 Soft bodies . . . 73
4.5.2 Rigid bodies . . . 74
4.5.3 Signed distance functions as collision objects . . . 76
4.5.4 Multi-body systems . . . 78
4.6 Parameterization of Rigid Body Rotation . . . 79
4.6.1 Exponential Coordinates and Angular Velocity . . . 79
4.6.2 Reparameterization . . . 80
4.7 Solving Inverse Problems . . . 81
4.7.1 Optimization Framework . . . 82
4.7.2 Motion Generation for Compliant Robot . . . 84
4.7.3 Material parameter estimation . . . 85
4.7.4 Trajectory Optimization . . . 89
4.7.5 Self-supervised learning of control policies . . . 94

4.8 Discussion 96
5 CONCLUSION 99
5.1 Discussion 99
5.2 Conclusion 100
5.3 Future Work 101
A APPENDIX 103
A.1 Rigid body Theory 103
  A.1.1 Parameterization of Rotation 103
  A.1.2 Rigid bodies 104
  A.1.3 Rigid body dynamics 105
A.2 Supplemental Material for Chapter 4 107
  A.2.1 Comparison to CMA-ES 107
  A.2.2 Scaling of computation time 107
  A.2.3 Number of contact points 108
  A.2.4 High-stiffness elastic material 108
  A.2.5 Overview of Simulation Parameters 111

BIBLIOGRAPHY 113

LIST OF FIGURES

Figure 1.1 Robot inspirations . . . 4
Figure 2.1 Robotic creatures created with our computational design system . . . 7
Figure 2.2 High-level overview of our design system . . . 11
Figure 2.3 Comparison of convergence rates for the motion optimization process . . . 20
Figure 2.4 Comparison of manual and semi-automatic design editing modes . . . 25
Figure 2.5 Result of design optimization . . . 26
Figure 2.6 Swizzle. . . . 28
Figure 2.7 A demonstration of a slalom motion design . . . 28
Figure 2.8 Convergence graph for automatic design optimization. 30
Figure 2.9 A robot leg before assembly. . . . 30
Figure 2.10 A variety of unique legged/wheeled robots designed with our system. . . . 32
Figure 3.1 AgileBot and Skaterbot. . . . 37
Figure 3.2 Convergence improvement when using the warm start routine. The dashed lines denote the warm start routine, which greatly improves convergence speed. . . . 42
Figure 3.3 Overview of computational framework . . . 45
Figure 3.4 Robot components . . . 46
Figure 3.5 Skaterbot on ice . . . 47
Figure 3.6 Comparison of generated, physically simulated and motion-captured trajectories for AgileBot . . . 48
Figure 3.7 Anymal on wheels performing dynamic motions . . 51
Figure 3.8 Results of the interactive TO in combination with the MPC . . . 52
Figure 3.9 Anymal on wheels performing a concatenated motion 53
Figure 4.1 Dropping a cylinder onto an inclined plane . . . 69
Figure 4.2 Objective function landscapes . . . 70
Figure 4.3 Optimization convergence for throwing a bunny . . . 71
Figure 4.4 Dropping an object composed of three tori . . . 73
Figure 4.5 Total energy over time for bouncing rigid cubes . . . 75
Figure 4.6 Forward simulation for different collision types . . . 77

Figure 4.7 Coupling soft and rigid bodies allows us to equip this compliant robot with soft feet 85
Figure 4.8 Control trajectories executed on a robot with compliant motors and soft feet 86
Figure 4.9 Errors obtained when optimizing for ground truth initial conditions and material parameters on synthetic data 87
Figure 4.10 Parameter estimation for throwing a sphere and a cube 88
Figure 4.11 Parameter estimation with Kinect data. 89
Figure 4.12 Throwing a deformable ball 90
Figure 4.13 Throwing with multiple contacts 91
Figure 4.14 Optimizing end-effector trajectories for robotic control of coupled dynamic systems 92
Figure 4.15 Trajectory optimization of three rigid bodies linked together by soft springs 93
Figure 4.16 Trajectory optimization for a rigid body with complex collision geometry. 94
Figure 4.17 Trajectory optimization of cloth modeled as mass-spring system 94
Figure 4.18 Differentiable simulation layer in a neural network. . 95
Figure 4.19 Convergence of the learning process for the throw controller 96
Figure A.1 Optimizing throwing the bunny to a specific target . 107
Figure A.2 Optimizing control trajectories for dragging a cube along a circular arc 108
Figure A.3 Computation time per number of rigid bodies 109
Figure A.4 Restitution experiment 110
Figure A.5 Objective function value per simulation run using continuation 110

LIST OF TABLES

Table 2.1 Robot zoo 31
Table 3.1 Outline of warm-start strategy 40
Table 3.2 Differences in SwizzleBot designs and motions 43
Table 3.3 Maximum Speed for Different Designs 49
Table A.1 Overview of simulation parameters 111
Table A.2 Overview of parameters used in trajectory optimization examples. 112

1

INTRODUCTION

The variety of motor skills possessed by humans and animals is astonishing. Tasks in our daily lives, such as running down the stairs to get the newspaper, turning the pages, and catching the coffee cup when it threatens to slide off the table, seem trivial to us. However, the motor skills we display every day are truly impressive. We humans are also very good at adapting our behavior to new environments. When we first put on skates, we may initially try to move around by pushing our feet backwards, only to soon realize that we can not exert any force in the direction of the skates. Then, figuring out how the skates and the ice interact, we find that the friction between the skate's blade and the ice perpendicular to the blade is far greater. And once we gain this understanding, we tilt our feet and successfully begin to push the ice under our feet. This innate understanding of how our body behaves and interacts with the physical environment is the foundation of our mobility and dexterity.

1.1 MOTIVATION

Nowadays, we can easily create our own virtual creatures that live in the digital worlds of computer games or movies. The design of these creatures is limited only by our imagination. They can resemble animals from the real world or be completely fictional. Computer animation then breathes life into their bodies to move with the same grace and agility as their real-world counterparts.

The advent of digital manufacturing such as 3D printing has opened up the possibility of bringing these virtual figures into the real world. In the real world, however, these creatures are subject to the laws of physics that govern every interaction with the surrounding world. While the motions in the virtual world are inspired by the laws of physics, motions in the real world are at the very mercy of these laws. And consequently, the motions from the virtual world cannot simply be transferred to the real world. The laws of physics govern all interactions in the real world, and therefore understanding these physical principles is the key to successful locomotion.

At the same time, roboticists face a similar challenge. As robotic components become more powerful and accessible, roboticists come up with novel robot designs and must figure out how to control these new machines most efficiently and robustly. However, compared to their biological counterparts, robots still lag behind in terms of agility, speed, and elegance. So what's missing in robotics such that we can build machines that interact with the world as dexterous and elegantly as animals and humans?

While this question certainly has many answers, we can take inspiration from biology. Behind the wide range of abilities and adaptability to new situations that animals and humans possess lies a deep understanding of how our bodies and objects behave in the physical world around us. When we are confronted with new environments - a slippery floor, for example - the laws of physics remain the same, and only the parameters change. We can reuse previous models that our brain has developed and only adjust the parameters, such as the slipperiness of the floor. Using this physical intuition, our body and brain develop adapted motor controls to navigate and manipulate the world we live in.

However, this intuition alone would be useless if our bodies were not so uniquely built. Roughly speaking, mammalian bodies are made up of rigid components, like bones and nails, and soft components, like tendons, muscles, or skin. There are certainly many reasons why mammals are built this way. From a motor control perspective however, there are distinct advantages of introducing compliance in a dynamical system. The combination of rigid and soft structures allows humans and animals to control their bodies robustly and interact effectively with the environment.

Similarly as in the virtual world, where there's no boundaries to our creativity, roboticists have come up with novel designs unseen in the animal world, such as combining legs and wheels. These mobile robots can be viewed as under-actuated, non-linear dynamical systems with large degrees of freedom. As such, motion planning and control of these machines is difficult. This has two consequences.

First, roboticists tend to build on a proven design. It is difficult to verify whether a new design would be better, since we have to develop optimal motion plans and controls for each new design. An example of this is the popular four-legged design. But what if a robot had wheels? How would it move? Would a traditional four-legged morphology still work?

Second, today's standard algorithms for controlling robots, such as inverse dynamics, are geared towards robots that consist of rigid components. However, the targeted introduction of compliant components could be the key to making them more robust and agile. The compliant components in the form of soft, damped motors or deformable feet can already be seen in current robot designs. To effectively exploit this compliance, the physical model on which high-level motion planning algorithms and low-level controllers are built need to account for this compliance.

In this thesis, we aim to develop computational algorithms and tools that enable motion synthesis frameworks to generate locomotion controls for robots with legs and wheels, as well as compliant components. Our vision is to develop a simulation model of the real world that provides robots with an innate understanding of physics to enable lifelike and elegant interactions with their environment.

1.2 STATE OF THE ART

Content creation has been a central theme since the early days of computer graphics. A variety of tools have been developed to help content creators to animate their virtual characters. However, translating these motions to characters living in the real world presents many new challenges. While we can alter a virtual world to our liking, in the real world, the laws of physics are predetermined. To overcome these real-world challenges, researchers have developed computational tools that allow users to generate stable motions that can be transferred to the real world. Users can focus on higher-level motion goals and styles, and algorithms ensure that the resulting motion trajectories are physically feasible. This computational approach enables the generation of motions for a variety of non-standard and novel robotic morphologies, such as the five-legged creature in Figure 1.1 top left [1].

At the same time, mobile robots have become increasingly sophisticated and reliable. One design is particularly popular: it is a quadruped in which each leg consists of three motors with the same motor-axis configuration. This design has proven quite successful for legged locomotion and is used by a few quadruped platforms such as Anybotics' Anymal, Boston Dynamics' Spot Mini or MIT's Mini Cheetah, which can be seen in Figure 1.1. Focusing on a single design allows roboticists to fine-tune their motion planning and control algorithms, resulting in fast, reliable and agile locomotion.

But what about other robot morphologies? One area we would like to highlight is locomotion with legs and wheels: Legs allow for dexterous mobility and wheels are fast and efficient. However, the combination of legs and wheels also poses additional challenges to the design and control

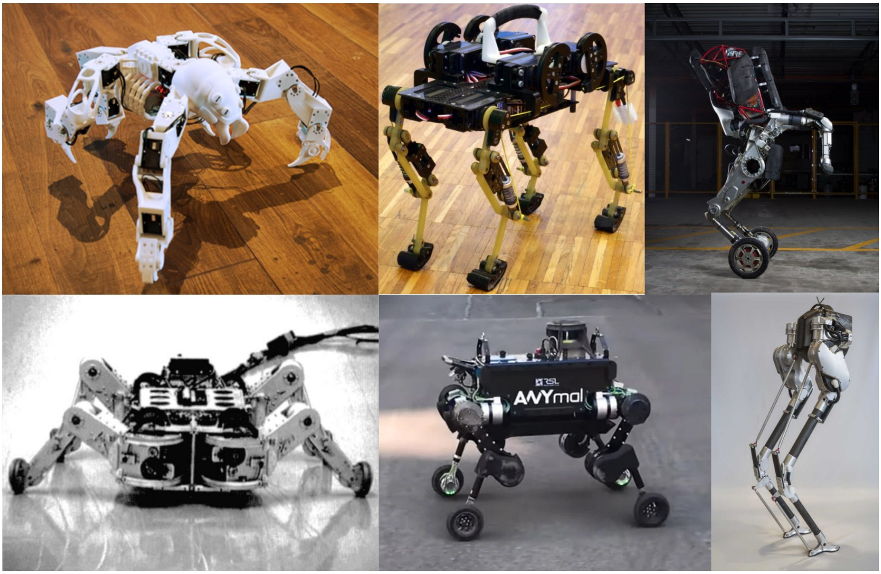


FIGURE 1.1: Robots with legs, wheels or compliant components that have inspired our work. From top-left to bottom-right: Robot designed with a computational framework [1]; Cheetah-Cub [2]; Boston Dynamics' Handle; Roller-Walker robot [3]; Anymal on wheels [4]; Agility Robotics' Cassie.

of these robots. Therefore, there are few robots today that have both legs and wheels. A notable example is the Roller-Walker robot, shown in Figure 1.1. It combines legs and *unactuated* wheels and can move without lifting its feet, resulting in a movement called "swizzling". The motions performed by this inspiring robot were hand-crafted specifically for this morphological design. Other robots combine legs and *actuated* wheels. Examples include Boston Dynamics' Handle and Anymal on wheels, both shown in Figure 1.1.

While wheels promise faster and more efficient locomotion, the motivation for building a robot with soft and compliant components is increased robustness, safety, and, potentially, performance. Every dynamic system exhibits some degree of desirable or undesirable compliance, even robots made predominantly of rigid components. The quadrupeds mentioned above all have soft feet and their motors have some compliance and damping. Another example is Agility's robot Cassie (Figure 1.1), which has a deformable plate in the linkage that controls the upper part of its leg. These are examples of rather rigidly built robots that add some compliance to their design. As more and more deformable components are introduced into robot designs, the grand challenge is to develop appropriate control strategies that govern these complex dynamical systems. To meet this challenge, much time and effort has been invested in developing differentiable physical models for hybrid multibody systems. These models can account for the compliance of robots and can then be used by motion planning and control algorithms. The models must be expressive enough to capture real-world phenomena such as deformation, friction, and contact while providing meaningful derivatives in terms of control parameters.

Despite these achievements, the agility, grace, and elegance of real animals remain unsurpassed. And the development of robots with novel morphological designs, compliant components, or wheels still requires an immense amount of engineering and prototyping. Every design decision-whether adding another limb, changing the stiffness of a component, or choosing between normal feet and wheels-is equivalent to changing a high-dimensional, nonlinear, and underactuted dynamical system. The nature of this dynamic system determines the space of motions the robot can potentially perform. And so every design decision affects the way the robot might be able to move.

The robots for which we want to generate motions may have an arbitrary arrangement of legs and joints, be equipped with unactuated or actuated wheels or ice-skates, or be designed with compliant motors or soft feet. The generated motions must take into account the dynamics of the particular robot design and exploit the unique characteristics of each robot morphology. Our goal is to develop algorithms and computational tools that unlock the motion capabilities of this wide range of robot morphologies and designs.

1.3 THESIS OVERVIEW

The work presented in this thesis explores different physical models of robots and trajectory optimization algorithms with the goal to generate compelling motions for legged-wheeled, and compliant robots. Each of the main Chapters 2 - 4 contains an introduction identifying the main contributions, followed by a Section on related work, a Section on results, and a discussion at the end of each Chapter.

We begin by introducing the project *Skaterbots* (Chapter 2), which presents a framework for motion generation of robots with legs and wheels. It allows users to create their own robot designs and generate appropriate motions. We use this system to design our own quadruped, *AgileBot* (Chapter 3), which can be equipped with actuated and passive wheels. Thanks to the flexibility of our motion planning tool, we can easily synthesise walking, rolling, swizzling and skating motions. Equipped with ice-skates, it is (one of) the first robots to master the art of skating. Using the experience gained from building Agilebot, we aim to create a new physics model that can capture the full dynamics and compliant behaviour of a real robot. This leads to the development of a *differentiable multibody simulator* (Chapter 4). We use it to generate motions for a compliant quadruped with soft feet. Unlike in previous Chapters, the underlying physical model is more general and can be used to model many types of robotics problems. We show several initial applications in the areas of trajectory optimization for object manipulation, parameter estimation, and machine learning. We conclude this thesis with a conclusion and promising directions for future work (Chapter 5).

2

MOTION GENERATION FOR LEGGED-WHEELED ROBOTS

In this Chapter we present a computation-driven approach to designing, optimizing and synthesizing motions for different breeds of legged-wheeled robots. At the core of our work lies an efficient trajectory optimization formulation tailored to the specific challenges of this class of robotic creatures. Through a unified treatment of feet and wheels, our model enables automatic generation of stable, physically-valid walking, rolling and skating motions for user-designed robots.

Although these motions are optimal with respect to the morphological characteristics of each individual robot, not all robots are created equal. Indeed, the motor capabilities of different robots can vary drastically. Optimizing design parameters for user created robots is therefore an indispensable piece of the puzzle which we also address in this Chapter. To this end, we develop a suite of computational tools that leverage sensitivity analysis to support manual, semi-automatic and fully automatic design exploration and optimization.

Using our robot design tool, we create a variety of unique robot morphologies with different arrangements of legs and end-effector types. We demonstrate the effectiveness of our computational approach by generating motions for each of the robots. To show that the motions transfer to the real world, we fabricate three of the robot designs and run the generated motions.

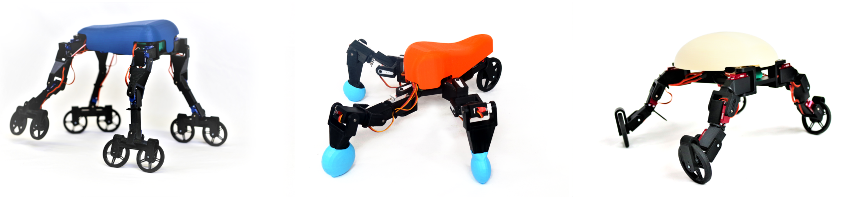


Figure 2.1: Robotic creatures created with our computational design system employ arbitrary arrangements of legs and wheels to locomote.

Adapted versions of this Chapter have been published as Geilinger, M., Poranne, R., Desai, R., Thomaszewski, B. & Coros, S. Skaterbots:

Optimization-Based Design and Motion Synthesis for Robotic Creatures with Legs and Wheels, *ACM Transactions on Graphics (TOG) 37, 1 (2018).*

2.1 INTRODUCTION

Whether it is to help with chores, keep us company, or entertain us, personal robots promise to play a central role in our increasingly technology-driven society. Echoing the trend of mass customization and leveraging recent advances in digital fabrication, our long term goal is to develop algorithmic foundations that will enable these robots to be created on-demand according to the individual needs and preferences of those they serve. In this quest, we join recent research efforts that bridge the fields of animation, fabrication-oriented design and robotics [5–7]. Complementing this body of work, we introduce a novel design system for a rich class of mobile robots that employ arbitrary arrangements of legs and wheels for locomotion. Such hybrid robots enjoy the combined versatility of legged and wheeled systems, but they also inherit their compounded challenges: they have many actuated degrees of freedom that need to be precisely coordinated in order to generate motions that are balanced, elegant, and efficient; their kinematics and dynamics are governed by highly non-linear equations; and their motor capabilities and physical design characteristics are inseparably intertwined. For these reasons, creating hybrid mobile robots remains a very difficult and error-prone task.

We present a computation-driven approach to designing, optimizing and synthesizing motions for different breeds of legged-wheeled robots. At the core of our work lies an efficient trajectory optimization formulation tailored to the specific challenges of this class of robotic creatures. Through a unified treatment of feet and wheels, our model enables automatic generation of stable, physically-valid walking, rolling and skating motions for user-designed robots. Although these motions are optimal accordig to motion goals and with respect to the morphological characteristics of each individual robot, not all robots are created equal. Indeed, the motor capabilities of different robots can vary drastically. Optimizing design parameters for user created robots is therefore an indispensable piece of the puzzle which we also address in this work. To this end, we develop a suite of computational tools that leverage sensitivity analysis to support manual, semi-automatic and fully automatic design exploration and optimization.

To validate our work, we designed a variety of robotic creatures and corresponding motions, all of which were tested using off-the-shelf physics-based simulators. We further fabricated three of our designs to assess the degree to which physical prototypes match our simulation results.

2.1.1 *Contributions*

Succinctly, our main contributions are:

- A versatile trajectory optimization formulation that is used to generate stable, physically-valid motions for a large variety of robots that employ legs and wheels for locomotion
- An analysis of the underlying numerical solver that reveals an effective way to drastically increase convergence rates for the motion optimization process
- A suite of user-guided computational tools that support manual, semi-automatic and fully automatic optimization of the robot's physical dimensions

2.2 RELATED WORK

Fabrication-aware design is a flourishing topic in Computer Graphics research. This is not surprising, since content generation has been a core topic of the field since its very beginnings, and digital fabrication machines are simply new types of output devices. However, going from virtual environments to the real world introduces many new challenges that must be addressed. For example, in recent years we have seen computational design approaches for objects that are lightweight yet strong [8, 9], objects whose optimized mass distribution allows them to stand, spin or float stably [10–12], physical characters that mirror the range of motion of their virtual counterparts [13–15], and increasingly complex mechanisms and mechanical automata designed to generate specific motions [16–20]. These computational tools share the same high-level goal as ours: enabling non-experts to create complex physical artifacts without requiring domain specific knowledge.

To increase the range of functionality for digitally-fabricated objects, researchers are also investigating computational approaches to embedding sensors and various other electromechanical components into their designs [21–24]. These research efforts build a bridge between the fields of HCI, Computer Graphics and Robotics, and our work follows this spirit. In particular, closely related to our work are algorithmic methods to design origami-inspired robots [6] as well as walking automata [25] and robotic creatures [26]. The computational techniques we describe in this thesis complement this body of work by targeting a diverse class of mobile robots that move using arbitrary arrangements of legs and wheels. The designs we support may have any number of morphologically different legs each equipped with a foot, or (un)actuated wheel. The many degrees of freedom of these robot designs and their hybrid legged-wheeled nature demands motion repertoires that are much richer and more intricate than those of robots relying largely on quasi-static walking [26]. We therefore present a new, highly efficient trajectory optimization approach that automatically generates walking, rolling, skating or gliding motions, as appropriate given the morphological designs of different robots.

Building on a growing body of literature [16, 20, 27–29], we leverage sensitivity analysis to establish a relationship between the motions a robot can generate and its physical design parameters. In particular, the method described in [29] nicely complements our work: while their formulation fine-tunes robot designs such that actuation forces are reduced, the suite of computational tools that we propose are specifically developed to support motion-aware manual, semi-automatic and fully automatic design exploration and optimization. Our work also draws inspiration from a number of specific, hybrid robot designs presented in the robotics literature [30–32]. These one-off designs are feats of engineering developed by teams of seasoned domain experts. Our long term goal is to allow even casual users to create robotic creations that approach the same level of sophistication.

2.3 OVERVIEW

Our system allows its users to create unique robot designs by connecting together different types of mechanical components in a mix-and-match manner. This design process is illustrated in Fig. 2.2 and can also be seen in the video [33]. Our implementation of the underlying graphical user interface is similar to the one employed by Desai et. al. [34], and the database of components we use for all our results consists of servomotors, 3D printable connectors and three types of end effectors: *actuated wheels* whose angular speed is controlled by motors, *passive wheels* that can spin freely about their rotation axis, and *welded wheels* that afford no motion relative to the body part they are attached to. Welded wheels are used to model feet that roll on the ground as the robots are moving, and when their radii are set to 0, they become equivalent to the point foot model commonly used by motion planning algorithms.

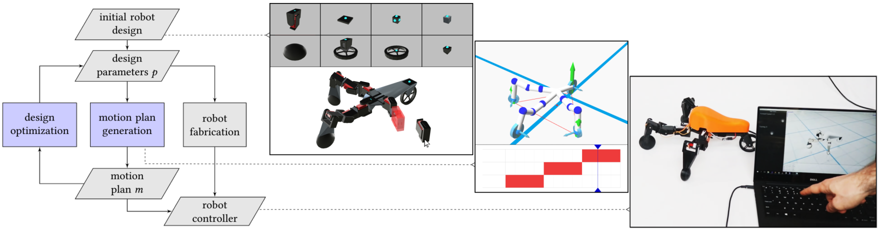


FIGURE 2.2: *High-level overview of our design system:* through a simple drag-and-drop interface, designers can interactively generate a vast array of robotic creatures. Once designs are finished, our system automatically generates physically-valid motions that are skilled and agile. We also present computational solutions for user-driven or automatic optimization of the robot's physical dimensions. The resulting designs can be easily fabricated, leading to compelling physical robots.

The *morphological design* of each robot is generated from a user-specified hierarchical arrangement of components: servomotors correspond to actuated joints, connectors define the geometric shape of each rigid link of the robot, and end-effectors specify the mechanical behavior of the components that will come into contact with the environment as the robot moves. This input directly defines the inertial parameters for each body part, 3D models for fabrication, as well as the rotation axes for each wheel and each joint actuator. With the resulting robot design as input, our trajectory optimization method (Sec. 2.4) generates physically-valid walking, rolling, gliding or skating motions. These motions are automatically tailored according to the morphological characteristics of each individual design. We leverage the particular structure of our motion synthesis formulation to significantly accelerate the underlying numerical solver (Sec. 2.4.3). This allows designers to interactively choreograph motions that are stable, agile and compelling.

To ensure that a robot functions as envisioned by its designer, we also develop computational solutions for manual, semi-automatic and fully automatic optimization of the robot's physical dimensions (Sec. 2.5). These tools leverage sensitivity analysis and allow even non-experts to explore the often unintuitive relationship between design parameters and motor capabilities, as a change in design may lead to a hard-to-predict effect in motion capabilities. Prior to fabrication, designs are validated through off-the-shelf physics simulators. We use proportional-derivative controllers to generate torques for every actuated joint of the simulated robot, and feed-forward velocity controllers for its actuated wheels. Time-varying joint angle and wheel speed targets for these low-level controllers are directly generated from the optimized motions. We use the Open Dynamics Engine [35] as a black-box simulator for all our results.

2.4 MOTION GENERATION

Our motion generation model builds on trajectory optimization techniques that reason in terms of a robot's *centroidal dynamics* [36]. This class of methods exploits the fact that modeling the evolution of the robot's aggregate linear and angular momenta over time is much simpler than considering its full-body dynamics. Nevertheless, this simplified dynamics representation can be easily complemented by geometric constraints to ensure that the generated motions are consistent with the robot's kinematics [37]. Because they strike a favorable balance between predictive power, simplicity and computational efficiency, models based on centroidal dynamics are quickly gaining in popularity.

In this Section, we describe a new mathematical formulation that leverages the concept of centroidal dynamics to efficiently generate dynamic motions for a diverse array of hybrid robotic creatures. The general nature of our formulation allows physically-valid walking, rolling, gliding and skating motions to emerge naturally as a function of the morphological design of each individual robot.

2.4.1 *Optimization Model*

The input to our motion optimization model consists of robots with arbitrary user-generated morphology. The robots interact with the environment through their end effectors, which can be located on any of their body parts. Without loss of generality, each end effector is assumed to be a wheel described by its radius r, mounting location $\hat{\mathbf{l}}$ on body part b, and rotation axis $\hat{\mathbf{a}}$ expressed in the local coordinate frame of b. Our motion optimization model supports passive and actuated wheels. Depending on the type of wheel, different constraints are instantiated as discussed in below.

Using a direct transcription approach, we turn to a time-discretized setting and represent a *motion plan* $\mathbf{m} = \{\mathbf{m}_1, \ldots, \mathbf{m}_T\}$ as a set of vectors $\mathbf{m}_i$ that span a planning horizon with length of time hT, where h is the amount of time between consecutive time samples. The subscript indexes specific samples in time, and $\mathbf{m}_i$ is defined as:

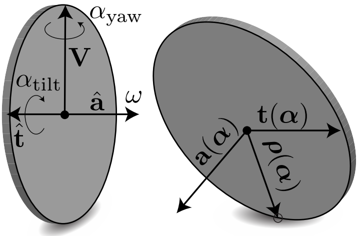
$$\mathbf{m}_i = \{\mathbf{q}_i, \mathbf{c}_i, \mathbf{e}_i^1, \dots \mathbf{e}_i^n, \mathbf{f}_i^1, \dots \mathbf{f}_i^n, \boldsymbol{\omega}_i^1, \dots \boldsymbol{\omega}_i^n, \boldsymbol{\alpha}_i^1, \dots \boldsymbol{\alpha}_i^n\} \tag{2.1}$$

For every time index i, the vector $\mathbf{q}_i = \{q_i^1, \dots, q_i^K\}$ denotes the pose of the robot, which consists of the position and orientation of the root link, as well as the joint angles that describe the relative orientation between articulated body parts; $\mathbf{q}_i$ is thus a vector of size $K = 6 + N$, where N is the number of joints. The corresponding *centroidal coordinate frame* $\mathbf{c} = \{\mathbf{x}, \boldsymbol{\theta}\}$ represents the robot's center-of-mass (COM) position $\mathbf{x}$ and global orientation $\boldsymbol{\theta}$. Each of the robot's n end effectors are referenced by superscript $j, 1 \leq j \leq n$. The points $\mathbf{e}$ define the location of the end effectors in a global coordinate frame (e.g. the points where each wheel should make contact with the environment) at different moments in time and $\mathbf{f}$ represents the force imparted by the robot onto the environment through each end effector. For each wheel in the robot's design we also store its instantaneous, world-relative angular speed $\boldsymbol{\omega}$, and two rotation angles $\boldsymbol{\alpha} = \{\alpha_{\text{tilt}}, \alpha_{\text{yaw}}\}$ that define the global orientation of its rotation axis:

$$\mathbf{a}(\boldsymbol{\alpha}) = W(\boldsymbol{\alpha}, \hat{\mathbf{a}}) = R_{\mathbf{v}}(\alpha_{\text{yaw}}) R_{\hat{\mathbf{t}}}(\alpha_{\text{tilt}}) \hat{\mathbf{a}}, \tag{2.2}$$

where R denotes a typical rotation operator. Note that we use the $\hat{\cdot}$ accent to denote quantities expressed in local coordinates. In the equation above, $\hat{\mathbf{a}}$ and $\mathbf{a}$ therefore represent the rotation axis of the wheel expressed in local and global coordinate frames, respectively.

As illustrated in the inset figure, the tilt axis $\hat{\mathbf{t}}$ is defined to lie at the intersection of the ground plane with the wheel plane. It is computed in the local coordinate frame of the wheel as $\hat{\mathbf{t}} = \hat{\mathbf{a}} \times \mathbf{v} / |\hat{\mathbf{a}} \times \mathbf{v}|$, where $\mathbf{v}$ is the vertical axis in global coordinates. The tilt axis in global coordinates, $\mathbf{t}(\boldsymbol{\alpha})$, is computed as $W(\boldsymbol{\alpha}, \hat{\mathbf{t}})$, and carries special meaning: it represents the only valid direction of movement for the wheel at any specific moment in time. Another important quantity, $\hat{\boldsymbol{\rho}} = \hat{\mathbf{t}} \times \hat{\mathbf{a}} r$, where r is the radius of the wheel, is the vector from the center of the wheel to the point where the end effector will make contact with environment (assuming locomotion on flat ground).


α_{yaw}
V
α_{tilt}
$\hat{\mathbf{a}}$
ω
$\hat{\mathbf{t}}$
$\mathbf{t}(\boldsymbol{\alpha})$
$\mathbf{a}(\boldsymbol{\alpha})$
$\boldsymbol{\rho}(\boldsymbol{\alpha})$

Our decision to model and parameterize wheels as separate entities warrants a brief discussion. We initially considered treating wheels as additional rigid bodies in the robot's morphological structure (i.e. their motion would be stored as part of $\mathbf{q}$, as for any other joint). While this modeling choice would reduce the overall number of parameters we need to optimize for, it presents two major downsides. First, even simple operations such as determining the point on a wheel that contacts the environment would require complex kinematic computations, because in the coordinate frame of a rigid body, this point does not remain fixed. In contrast, it is easily seen that upon transformation to global coordinates, $\boldsymbol{\rho} = W(\boldsymbol{\alpha}, \hat{\boldsymbol{\rho}})$ maintains its meaning, i.e. it still represents the vector from the center of the wheel to the point that is closest to the ground. As a result, the objectives and constraints formulated below take on much simpler forms and become faster to evaluate. Second, as discussed in Sec 2.4.3, optimization terms that include the full kinematic model of the robot are highly non-convex, and if left untreated they negatively affect convergence rates. The auxiliary variables we introduce for wheels allow us to localize these numerical issues to just two consistency constraints, which we can therefore analyze and address in isolation.

With the parameterization of the motion plan in place, we now turn our attention to the constraints and objectives that govern the motions of wheeled/legged hybrid robots.

KINEMATICS AND DYNAMICS: Leveraging the centroidal dynamics representation, the global motion of the robot is governed by the familiar Newton-Euler equations. For our discrete setting, these equations take the form:

$$\sum_{j=1}^{n} \mathbf{f}_i^j + M\mathbf{g} = M\ddot{\mathbf{x}}_i\,, \forall i \tag{2.3}$$

$$\sum_{j=1}^{n} (\mathbf{e}_i^j - \mathbf{x}_i) \times \mathbf{f}_i^j = \mathbf{I}\ddot{\boldsymbol{\theta}} + \dot{\boldsymbol{\theta}} \times \mathbf{I}\dot{\boldsymbol{\theta}}\,, \forall i \tag{2.4}$$

As before, the subscript i denotes a specific time index and the superscript j refers to individual end effectors. The total mass of the robot is M and its moment of inertia I is computed about the robot's COM. The center of mass acceleration, $\ddot{\mathbf{x}}_i$, is estimated using finite differences: $\ddot{\mathbf{x}}_i = (\mathbf{x}_{i-1} - 2\mathbf{x}_i + \mathbf{x}_{i+1})/h^2$, where h is the time step. Similarly, $\ddot{\boldsymbol{\theta}}$ is computed using finite differences that operate on axis-angle representations of the centroidal coordinate frame's change in orientation between time steps.

The motion of the centroidal coordinate frame is a function of the forces that the robot's end effectors impart onto the environment. To ensure their physical feasibility, these forces are subjected to constraints imposed by a typical Coulomb friction model:

$$\mathbf{f}_n \geq 0\,, |\mathbf{f}_t| \leq \mu \mathbf{f}_n\,, \tag{2.5}$$

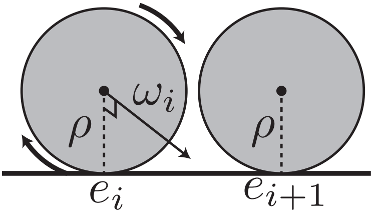
where $\mathbf{f}_t$ and $\mathbf{f}_n$ denote the tangential and normal component of $\mathbf{f}$, and μ is the coefficient of friction. Forces generated by unactuated wheels are further constrained to have a vanishing component in the direction along which the wheel is free to move (i.e. they can only push on the ground in an orthogonal direction):

$$\mathbf{f} \cdot \mathbf{t}(\boldsymbol{\alpha}) = 0\,, \tag{2.6}$$

Furthermore, end effectors can only generate ground reaction forces when they are in contact with the environment. This behavior is enforced through constraints of the form:

$$(1 - c)\mathbf{f} = 0\,, \tag{2.7}$$

where the binary contact flags c are specified by a foot fall pattern per end effector, per time step [26]. These flags represent the schedule of contacts that characterize different locomotion gaits; $c = 1$ indicates that an end effector must be in contact with the ground, while $c = 0$ denotes swing phases during which end effectors cannot generate ground reaction forces. Constraints 2.5-2.7 are applied for all time samples i and all end effectors j.



When end effectors are in contact with the ground, their motion must be subject to kinematic no-slip constraints. Recall that $\mathbf{e}_i^j$ represents the location where end effector j makes contact with the ground at time index i. We wish the evolution of these points to be consistent with the motion of the wheel, as shown in the inset figure. If at some moment in time the wheel has angular velocity $\boldsymbol{\omega}$ and the relative velocity between the wheel and the ground at the contact point is $\mathbf{0}$, then the center of the wheel must have velocity $\boldsymbol{\omega} \times \boldsymbol{\rho}$. The time derivative of $\mathbf{e}_i^j$ must take on the same value, so the no-slip constraint is formulated as:

$$\left(\frac{\mathbf{e}_{i+1}^j - \mathbf{e}_i^j}{h} + \boldsymbol{\omega}_i^j \mathbf{a}(\boldsymbol{\alpha}_i^j) \times \boldsymbol{\rho}(\boldsymbol{\alpha}_i^j) \right) c_i^j = 0 \tag{2.8}$$

While the constraint above bounds end effector velocities such that their motions model rolling wheels, the vertical component of their motion must also be prescribed. We accomplish this through a simple constraint that asks that the normal component of the end effector position is 0 in stance (i.e. $c = 1$), or attain a user-specified value w_h otherwise:

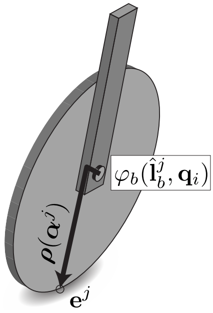
$$c_i^j \mathbf{e}_i^j \cdot \mathbf{n} + (1 - c_i^j)(\mathbf{e}_i^j \cdot \mathbf{n} - w_h) = 0 \tag{2.9}$$

The motion of the end effectors is further optimized to ensure that the motions that are generated are collision-free. For this purpose, we use inequality constraints applied to every pair of end-effectors:

$$||\mathbf{e}_i^j - \mathbf{e}_i^k||_2^2 \geq (r^j + r^k + \beta)^2, \forall j,k \leq n \tag{2.10}$$

where $\beta = 2cm$ is a safety factor.

We note that the constraints imposed on the motion of end effectors and the forces they generate are a function of the types of wheels employed in each design. These constraints, while each simple in formulation, lead to interesting and often surprising motions.


$\varphi_b(\hat{\mathbf{l}}_b^j, \mathbf{q}_i)$
$\boldsymbol{\rho}(\boldsymbol{\alpha}^j)$
$\mathbf{e}^j$

CONSISTENCY CONSTRAINTS: The terms described above operate on the centroidal coordinate frame and the set of auxiliary end effector variables introduced by our model. We ensure that the motion of the robot is in sync with these quantities through a set of consistency constraints. First, we ask that the trajectory of the robot's center of mass matches the linear motion of the centroidal coordinate frame:

$$\varphi_{CoM}(\mathbf{q}_i) - \mathbf{x}_i = 0\,, \tag{2.11}$$

where $\varphi_{CoM}(\mathbf{q})$ outputs the robot's center of mass given pose $\mathbf{q}$. Likewise, the orientation of the robot's body, $\varphi_\theta(\mathbf{q})$, must match that of the centroidal coordinate frame:

$$\varphi_\theta(\mathbf{q}_i) R(\boldsymbol{\theta}_i)^{-1} = I \tag{2.12}$$

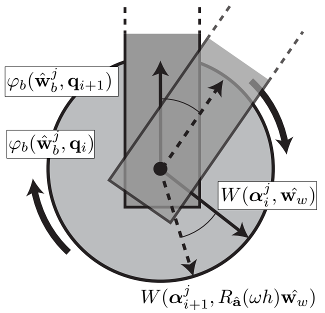
To mirror the motion prescribed through the auxiliary variables for wheels, two constraints must be satisfied:

$$\varphi_b(\hat{\mathbf{a}}^j, \mathbf{q}_i) - \mathbf{a}(\boldsymbol{\alpha}_i^j) = 0\,, \tag{2.13}$$

$$\varphi_b(\hat{\mathbf{l}}^j, \mathbf{q}_i) + \boldsymbol{\rho}(\boldsymbol{\alpha}_i^j) - \mathbf{e}_i^j = 0\,, \tag{2.14}$$

Here, φ_b is the forward kinematics function that computes world coordinates of points or vectors defined in the local coordinate frame of rigid body b which wheel j is mounted on. The first constraint therefore demands that the wheel axis, as seen from the coordinate frame of b, is aligned with the wheel axis computed through the auxiliary variables. The second constraint further requires that the position of the wheel's center of rotation satisfies the kinematic relationship illustrated in the inset figure.

The last consistency constraint is instantiated only for welded wheels that must have zero velocity relative to the robot body part they are mounted on. Before we provide the formulation of the constraints that model *welded wheels*, recall that the wheel speed ω is specified in a global coordinate frame, and not relative to b. Consequently, we must establish a correspondence between the global motion of body part b, and the motion represented by the auxiliary parameters of the wheel. We can do this efficiently by requiring that vectors lying in the plane of the wheel, as seen from the two different coordinate frames, follow equivalent movement patterns. The constraint therefore becomes:


$\varphi_b(\hat{\mathbf{w}}_b^j, \mathbf{q}_{i+1})$
$\varphi_b(\hat{\mathbf{w}}_b^j, \mathbf{q}_i)$
$W(\boldsymbol{\alpha}_i^j, \hat{\mathbf{w}}_w)$
$W(\boldsymbol{\alpha}_{i+1}^j, R_{\hat{\mathbf{a}}}(\omega h)\hat{\mathbf{w}}_w)$

$$\varphi_b(\hat{\mathbf{w}}_b^j, \mathbf{q}_i) \times \varphi_b(\hat{\mathbf{w}}_b^j, \mathbf{q}_{i+1}) \;=\; W(\boldsymbol{\alpha}_i^j, \hat{\mathbf{w}}_w) \times W(\boldsymbol{\alpha}_{i+1}^j, R_{\hat{\mathbf{a}}}(\omega h)\hat{\mathbf{w}}_w) \quad (2.15)$$

where $\hat{\mathbf{w}}_b$ and $\hat{\mathbf{w}}_w$ are two *reference vectors* that lie in the plane of the wheel, specified in the local coordinate frame of the parent body b and the auxiliary wheel frame, respectively. We note that the end effector parameterization does not explicitly store the orientation of the wheel about its rotation axis. For this reason, for the second term on the right hand side, we first rotate the vector $\hat{\mathbf{w}}_w$ according to the angular speed of the wheel, and then compute its world coordinates, as illustrated in the inset figure. This formulation does not require $\hat{\mathbf{w}}_b$ and $\hat{\mathbf{w}}_w$ to be the same vector, as the cross product operator outputs the same result for any pair of reference vectors we choose.

PHYSICAL HARDWARE CONSTRAINTS: With the motion of the robot consistent with its centroidal dynamics, it is important to ensure that the limitations of physical actuators are also respected. We therefore implement bound constraints for the range of motion of each joint angle $\mathbf{q}_k$, its rate of change $\dot{\mathbf{q}}_k$ and the angular speed ω of active wheels. The values for the bound constraints are hardware-specific, but they otherwise take on standard forms, which we omit for brevity.

BOUNDARY CONDITIONS: Our trajectory optimization model supports the generation of periodic motions as well as motions with prescribed starting and end states. For periodic motions, we implement constraints that ask that the poses of the robot at the start and end of the motion plan to be identical, i.e. $\mathbf{q}_1 = \mathbf{q}_T$. Constraints for prescribed starting or end states take on similar forms, and the target values can be poses generated for any other motion. In this way, our method can easily generate transitions between periodic motions to create motion graphs [38].

FUNCTIONAL OBJECTIVES AND MOTION REGULARIZERS Complementing the constraints detailed above, we also implement several objectives that provide interactive control over the generated motions. The walking speed, for example, is controlled by specifying a target offset between the first and last configuration of the centroidal coordinate frame: $||(\mathbf{x}_T - \mathbf{x}_1) - \mathbf{t}||_2$. The turning rate is similarly controlled by specifying a target yaw angle between $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_T$. To promote the generation of smooth motions, we also include a regularizing term defined as $(\mathbf{q}_{i-1} - 2\mathbf{q}_i + \mathbf{q}_{i+1})$.

To provide further control over the generated motions, we allow users to interactively specify target positions for the robot's end effectors or COM trajectory. This interaction mode, which is demonstrated in the accompanying video [33], is supported by simple objectives of the form $||\mathbf{e}_i^j - \mathbf{e}_{\text{target}}||_2$ or $||\mathbf{x}_i^j - \mathbf{x}_{\text{target}}||_2$. We call these *choreography* objectives, and they are directly instantiated or removed by the user as desired. Supported by the real-time feedback enabled by the efficient solver described in Sec. 2.4.3, we found this interactive motion synthesis mode to be very effective. This user-in-the-loop optimization scheme can be used both to help the optimization overcome undesirable local minima when they occur, as well as to shape the style of the resulting motions.

2.4.2 *Numerical Solution*

One common approach to solving constrained optimization problems is through Sequential Quadratic Programming. We initially pursued this approach but, arguably due to the highly non-linear constraints required for our formulation, we could not find a strategy (i.e., merit function and line search parameters) that would reliably lead to good convergence. We therefore resorted to a penalty-based approach that allows us to isolate the most challenging constraints into specific objectives that can then be treated in a numerically stable way. To transform equality constraints into penalty terms, it suffices to minimize their inner product: constraint $\mathbf{f}(\mathbf{x}) = \mathbf{b}$ becomes $(\mathbf{f}(\mathbf{x}) - \mathbf{b})^T(\mathbf{f}(\mathbf{x}) - \mathbf{b})$. Inequality constraints are slightly more involved. As in [7], we first define a $\mathcal{C}^2$, piece-wise polynomial function $\psi(x)$ as:

$$\psi(x) = \begin{cases} 0 & x \leq -\varepsilon \\ \frac{1}{6\varepsilon}x^3 + \frac{1}{2}x^2 + \frac{\varepsilon}{2}x + \frac{\varepsilon^2}{6} & -\varepsilon \leq x < \varepsilon \\ x^2 + \frac{\varepsilon^2}{3} & \text{otherwise} \end{cases} \tag{2.16}$$

Each scalar inequality constraint $f(\mathbf{x}) \leq b$ is then modeled as $\psi(f(\mathbf{x}) - b)$. The function $\psi(x)$ is a quadratic penalty term when $x \geq \epsilon$, it is zero if $x \leq -\epsilon$, and it behaves as a smooth interpolant otherwise. The constant ϵ often affords an intuitive interpretation. For example, when placing a bound on the velocity of a motor, we set ϵ to 10% of its maximum speed.

To solve the resulting optimization problem, we define a function $E(\mathbf{m})$ as a weighted sum of all objectives and penalty terms associated with the equality and inequality constraints introduced earlier. We use a weight of 10000 for all penalty terms. The weights for the functional objectives are set to 50, while the motion regularizers are assigned a weight of 0.1.

To minimize $E(\mathbf{m})$, we use Newton's Method coupled with a backtracking line search method. Once the optimization process converges, we alert the user if the residual of any constraint penalty term is above an acceptable threshold. This could be an indication of an invalid robot design, as discussed in Sec. 2.5.

2.4.3 *Further Analysis and Optimization Speedup*

Although all gradients and Hessians are computed analytically in our framework (we verified our implementation against numerical differentiation estimates), the trajectory optimization algorithm is still quite slow to converge, as illustrated in Fig. 2.3. This behavior is often caused by objectives with an indefinite Hessian. If the Hessian is not positive semi-definite (PSD), the Newton step may result in a non-descent search direction, which explains the "jittery" nature of the convergence plot. To avoid this artifact, indefinite Hessians must be modified such as to remove negative eigenvalues. In many cases, one might opt to simply *regularize* the Hessian by adding a scaled identity matrix to it. However, determining the optimal coefficient value is costly. If it is too low, the Hessian will remain indefinite, and too high a value will slow down progress. Dynamic regularization schemes, which we also tested, start out with a small coefficient that is progressively increased if the search direction is invalid. Nevertheless, every iteration requires an attempt to solve the underlying linear system, so a different strategy is needed.

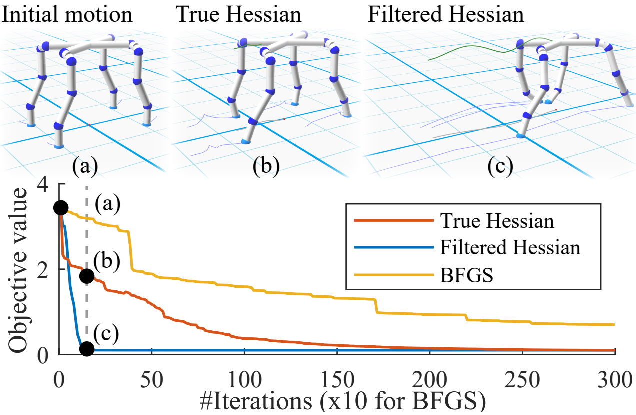


Figure 2.3: Comparison of convergence rates for the motion optimization process using the True Hessian and the Filtered Hessian approximation (Sec 2.4.3). The configurations of the robot in (b) and (c) correspond to the motion plan after 10 optimization steps. The non-smooth end effector trajectories shown in (b) provide a visual indication that the motion has not converged. L-BFGS, a common quasi-Newton method, features very poor convergence for our problem.

Furthermore, for multi-objective problems, while summing up the contributions from different objectives might result in an overall PSD Hessian, individual objectives that are badly behaved might still hinder progress. To pinpoint the root cause of the problem, we examined the Hessians of all our objectives. We found that the culprit was hidden in the consistency constraints (2.11)-(2.14). For example, when written as a penalty term, (2.13) has the form

$$E(\mathbf{q}_i, \boldsymbol{\alpha}_i^j) = \|\varphi_b(\hat{\mathbf{a}}^j, \mathbf{q}_i) - \mathbf{a}(\boldsymbol{\alpha}_i^j)\|^2 = \|\varphi_b - \mathbf{a}(\boldsymbol{\alpha}_i^j)\|^2 \tag{2.17}$$

where we use $\varphi_b := \varphi_b(\hat{\mathbf{a}}^j, \mathbf{q}_i)$ for brevity. Its gradient w.r.t $\mathbf{q}_i$ is

$$\nabla_{\mathbf{q}_i} E(\mathbf{q}_i, \boldsymbol{\alpha}_i^j) = J_{\mathbf{q}_i}\varphi_b^T(\varphi_b - \mathbf{a}(\boldsymbol{\alpha}_i^j)) \tag{2.18}$$

where $J_{\mathbf{q}_i}\varphi_b$ is the Jacobian of φ_b w.r.t. $\mathbf{q}_i$. The Hessian with respect to $\mathbf{q}_i$ is therefore

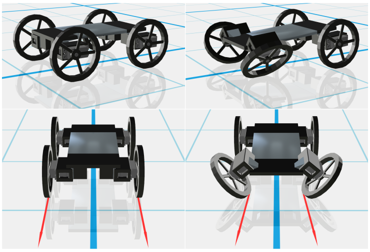
$$\nabla^2_{\mathbf{q}_i} E(\mathbf{q}_i, \boldsymbol{\alpha}_i^j) = \sum_{k=1}^{3} (J_{\mathbf{q}_i}\varphi_b)_{(k)} (J_{\mathbf{q}_i}\varphi_b)^T_{(k)} + \left[\frac{(\partial J_{\mathbf{q}_i}\varphi_b)^T}{\partial q_i^k} (\varphi_b - \mathbf{a}(\boldsymbol{\alpha}_i^j)) \right]_{k=1}^{K}, \qquad (2.19)$$

where we use $A_{(k)}$ for the k'th column of a matrix A, and $[\mathbf{v}_k]_{k=1}^{k}$ for the concatenation of K vectors $\mathbf{v}_k$. The first term is always PSD since it is a sum of outer products of vectors. However, we found that the second term is often indefinite. We therefore simply exclude it when computing the Hessian. We apply the same modification to the other consistency terms and call the result the *Filtered Hessian*. This simplification is akin to a Gauss-Newton approximation, but we note we only remove second derivatives of (2.17) with respect to $\mathbf{q}_i$. A standard Gauss-Newton approximation would also remove second derivatives with respect to other parameters (e.g. $\boldsymbol{\alpha}_i^j$)), as well as mixed derivative terms. Furthermore, Gauss-Newton is traditionally used on the entire objective when second-order derivatives are too difficult or too expensive to evaluate. We use our approximation on only a select few objectives. This simple filtering operation results in remarkably faster and smoother convergence, as can be seen in Fig. 2.3. This result also highlights another benefit of our formulation that employs auxiliary variables to model end effectors: it allows these types of numerical problems to be isolated and addressed in a targeted way.

2.5 DESIGN OPTIMIZATION

2.5.1 *Motivation*

Our early experiments with the motion optimization model described in the previous Section exposed an interesting challenge: the relationship between the morphological design of a robot and the motions it can generate can be very unintuitive. We illustrate this challenge with the example shown in the inset figure. The design on the right is a simple car with four parallel wheels, all of which are actuated. When asked to move forward at constant



speed, the trajectory optimization method generated the trivial motion plan that one would expect. The design shown on the right is the same car, but with the front wheels tilted by 45 degrees. This seemingly innocent editing operation resulted in the robot no longer being able to move forward as expected.

This is because tilting the front wheels has the unintended consequence of lifting them off the ground. The pitch angle of the robot's body must therefore be adjusted such that all four wheels are once again in contact with the environment. In this new configuration, however, as a result of the combined tilt and pitch of the front wheels, the directions along which they can move are no longer parallel – recall that these directions, which are visualized as red arrows, are given by the intersection of the ground plane with the plane that the wheels lie in. Slip-free motion is therefore no longer possible. With this design flaw identified, the robotic car is easy to fix: lowering the front wheels by just the right amount eliminates the need for the body pitch, ensuring the directions of movement for all wheels are once again in agreement.

2.5.2 *Technical solution*

The simple example described above highlights the subtleties and potential pitfalls inherent to the task of creating mobile robots. Needless to say, analyzing and finding problems with designs becomes much more difficult as they increase in complexity. Computational approaches to correcting flaws and improving user-generated designs is therefore of utmost importance. We explore solutions to this technical challenge through a suite of computational tools that enable system-guided manual, semi-automatic and fully automatic design methodologies. These tools leverage the fact that the sensitivities of optimal motions with respect to morphological design parameters encode very valuable information.

We treat each user-generated design as a *parameterized morphological template* Ψ which takes as input a vector $\mathbf{p}$. For our implementation, elements of $\mathbf{p}$ encode the location of each joint and each end effector in the local coordinate frame of their parent rigid bodies. $\Psi(\mathbf{p})$ therefore outputs a specific robot design, and different vectors $\mathbf{p}$ result in robots that have the same morphology but different body proportions. Robots generated with our graphical design system provide both a morphology, which remains fixed, and an initial set of parameters $\mathbf{p}_0$. The motions generated through trajectory optimization and the robot's morphological parameters are deeply intertwined through the forward kinematics functions φ_b, φ_{CoM} and φ_θ that appear in Eq. 2.11-2.15. Without loss of generality, we can therefore express the motion of the robot as a function of its design parameters,

$$\mathbf{m}(\mathbf{p}) = \arg\min_{\tilde{\mathbf{m}}} \ E(\tilde{\mathbf{m}}, \mathbf{p}) \ , \tag{2.20}$$

where the optimization energy E was defined in Sec. 2.4.2.

Although the function $\mathbf{m}(\mathbf{p})$ does not afford an analytic solution, we leverage the fact that $\mathbf{m}$ and $\mathbf{p}$ are coupled through E to compute an explicit map relating them. To derive this map, we note that as $\mathbf{p}$ changes, we can always compute a new motion such that $\mathbf{G}(\mathbf{m}, \mathbf{p}) = \partial E / \partial \mathbf{m} = 0$, i.e. $\mathbf{m}$ is the minimizer of E for the new design parameters. Consequently, the total derivative of $\mathbf{G}(\mathbf{m}, \mathbf{p})$ with respect to $\mathbf{p}$, $d\mathbf{G}/d\mathbf{p}$, vanishes always. Applying the chain rule, we obtain

$$\frac{d\mathbf{G}}{d\mathbf{p}} = \frac{\partial \mathbf{G}}{\partial \mathbf{m}} \frac{\partial \mathbf{m}}{\partial \mathbf{p}} + \frac{\partial \mathbf{G}}{\partial \mathbf{p}} = \mathbf{0} \ . \tag{2.21}$$

This expression exposes the Jacobian $\mathbf{J} = \frac{\partial \mathbf{m}}{\partial \mathbf{p}}$, which captures to first order how motion parameters $\mathbf{m}$ need to change as the design parameters $\mathbf{p}$ change such that the solution remains on the manifold of optimal motions, i.e., $\mathbf{G}(\mathbf{m}, \mathbf{p}) = \mathbf{G}(\mathbf{m} + \partial \mathbf{m}, \mathbf{p} + \partial \mathbf{p}) = \mathbf{0}$. Computing this Jacobian requires the Hessian $\partial \mathbf{G} / \partial \mathbf{m}$, which we compute analytically, and the term $\partial \mathbf{G} / \partial \mathbf{p}$, which we estimate numerically. With $\mathbf{J}$ at hand, we describe three system-guided design editing modes supported by our computational framework.

2.5.3 *Manual Design Mode*

As shown in the accompanying video [33], our computational framework provides an intuitive interface to directly edit the morphological parameters of a robot design. The goal of the *manual design mode* is to enable users to freely explore the design space. As the body proportions of robotic creatures are interactively adjusted, our computational system provides near-instantaneous feedback by generating and displaying corresponding optimal motions. We achieve this performance by coupling the efficient numerical treatment described in Sec. 2.4.3 with a simple warm-starting scheme. Briefly, through the Jacobian $\partial \mathbf{m} / \partial \mathbf{p}$, user-provided edits $\Delta \mathbf{p}$ are explicitly mapped to the changes in motion $\Delta \mathbf{m}$ that they induce, $\Delta \mathbf{m} = \partial \mathbf{m} / \partial \mathbf{p} \Delta \mathbf{p}$. We therefore update the current motion plan by $\Delta \mathbf{m}$ and then proceed with numerical optimization.

2.5.4 *Semi-Automatic Design Mode*

While manual editing enables free-form exploration of the relationship between robot designs and corresponding optimal motions, it is also important to have the ability to optimize designs according to specific functional goals. In general, these goals can be defined in a variety of ways. The option we explore for our *semi-automatic design mode* is the following: starting from an initial robot $\Psi(\mathbf{p}_0)$ and associated motion $\mathbf{m}(\mathbf{p}_0)$, we enable user-driven generation of design variations that ensure specific features of the motion $\mathbf{m}^f \subseteq \mathbf{m}$ are affected as little as possible. In other words, we allow the user to navigate the null space of their robot design. The desired set of motion features is selected by the user from a dropdown menu, and it can consist of the linear or angular motion of the body, the robot's joint angle trajectories, or the paths that its end effectors move along. As before, designers can manually edit the robot's body proportions as they desire. These user-provided morphological edits, $\Delta\mathbf{p}^u$, are complemented by synergistic changes to all other design parameters, $\Delta\mathbf{p}^s$, that are automatically computed such that changes to the motion features, $\Delta\mathbf{m}^f$, are minimal. Noting that $\Delta\mathbf{m}^f = \mathbf{J}^f(\Delta\mathbf{p}^u + \Delta\mathbf{p}^s)$, where $\mathbf{J}^f = \frac{\partial\mathbf{m}^f}{\partial\mathbf{p}}$, $\Delta\mathbf{p}_s$ is the optimum of the following quadratic program:

$$\begin{aligned} &\min_{\Delta\mathbf{p}^s} \frac{1}{2}\left|\left|\frac{\partial\mathbf{m}^f}{\partial\mathbf{p}}(\Delta\mathbf{p}^u + \Delta\mathbf{p}^s) - \Delta\mathbf{m}^f\right|\right|_2^2 \\ &\qquad\qquad\qquad\qquad \text{subject to} \quad \Delta\mathbf{p}_i^s = 0, \forall i : \Delta\mathbf{p}_i^u \neq 0 \end{aligned} \tag{2.22}$$

The constraints ensure that the changes to design parameters that are automatically computed are 0 for all components of $\mathbf{p}$ that are user-specified, and $\Delta\mathbf{m}^f = \mathbf{m}^f(\mathbf{p}) - \mathbf{m}^f(\mathbf{p}_0)$ helps to combat drift. With the solution to this optimization problem computed, the design parameters are updated as $\mathbf{p} = \mathbf{p} + (\Delta\mathbf{p}^u + \Delta\mathbf{p}^s)$, and the corresponding optimal motion is recomputed. The updated robot design respects the morphological edits specified by the user while minimally changing the selected set of motion features. The process, of course, can repeat as desired.

2.5.5 *Automatic Design Optimization*

As a more general solution, our mathematical framework also supports the optimization of designs according to arbitrary functions defined in terms of the robot's motion parameters $\mathbf{m}$. The choreography and functional objectives described in the previous Section, as well as the overall energy

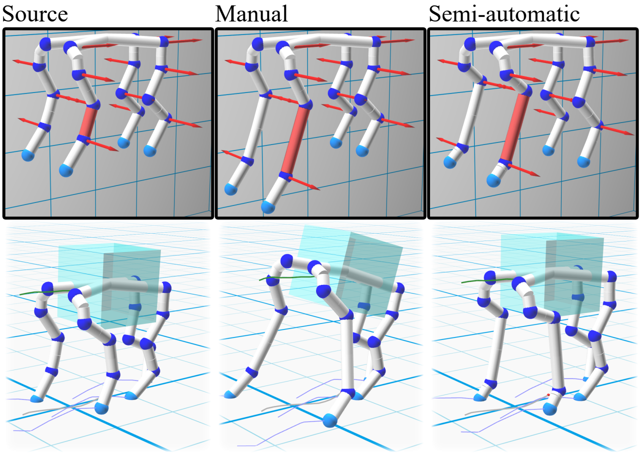


FIGURE 2.4: Comparison of manual and semi-automatic design editing modes. In this example, we increase the length of the highlighted link and apply symmetric edits to the right side of the body. In manual mode, the lengthening of the leg results in a noticeable body pitch – the transparent boxes are added to help better visualize body orientations. In semi-automatic mode, we ask that the orientation of the body does not change. As a result, when the highlighted link is lengthened, synergistic adjustments to other design parameters are automatically applied as well. In both cases, the weights of the objectives governing the motion of the body are exactly the same, so its orientation is only influenced by the robot's design.

E minimized during the trajectory optimization process are examples of such functions, which we denote as $\mathcal{O}(\mathbf{m})$. Through the chain rule, we can easily compute the gradient of $\mathcal{O}(\mathbf{m})$ with respect to the design parameters: $\frac{\partial \mathcal{O}}{\partial \mathbf{p}} = \frac{\partial \mathbf{m}}{\partial \mathbf{p}}^T \frac{\partial \mathcal{O}}{\partial \mathbf{m}}$. Updating the robot's design parameters $\mathbf{p}$ using a step along this gradient (i.e. $\mathbf{p} = \mathbf{p} + \beta \frac{\partial \mathcal{O}}{\partial \mathbf{p}}$) induces the change in the robot's optimal motion that most improves $\mathcal{O}$. We have used this optimization scheme to confirm that the design of the simple car described at the start of this Section can be fixed automatically. In this case, $\mathcal{O}(\mathbf{m})$ was simply $E(\mathbf{m})$. As we demonstrate in the results video [33], based on the choreography objectives, the design of the robot and its motions can be concurrently generated in a co-optimization process that is guided by the designer.

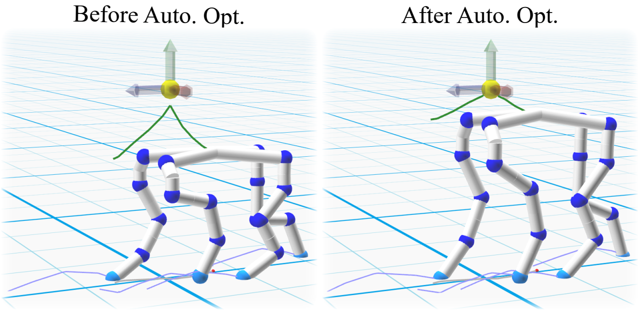


FIGURE 2.5: Result of design optimization. The robot is asked to reach the high target position as shown, but its initial design makes it unable to comply. By optimizing the design parameters, the robot's physical dimensions automatically change such that it can achive the user-specified motion goal.

2.6 RESULTS

Here we discuss our results, which showcase a variety of robots and corresponding motions generated with our method. Thanks to our interactive design system, all of these results were very easy to create. We emphasize that all motions emerged automatically as a function of the morphological design of each robot (i.e. number of limbs, types of wheels, etc). Users provided only high-level guidance in the form of a desired moving speed, or optionally, in the form of a sparse set of targets for the robot's body over time. The accompanying results video [33] shows real-time screen captures of our design system, including the motion optimization and physics simulation steps.

2.6.1 *Wheeled robots*

ACTUATED WHEELS Active wheels are highly successful locomotive devices in their own right, as evidenced by the abundance of vehicles found on our roads. This begs the question: what can be gained by placing motorized wheels on robotic legs? In addition to providing the option to switch to legged locomotion whenever convenient, the extra flexibility afforded by combining legs and wheels enables increased agility. We demonstrate this in the results video [33] by asking one of our robots, Agilebot, to first accelerate and then quickly slow to a halt. In order to not lose balance and topple over, the robot extends its front legs forward to maintain the center of pressure within the support polygon. It is worth noting that the optimization process *discovers* this strategy by itself, without any intervention from the user. We further demonstrate automatically generated step-and-drive and turn-in-place motions that are enabled by the robot's hybrid wheeled/legged design.

PASSIVE WHEELS Passive wheels also provide ample opportunities for efficient locomotion, *skating* being the prime example. Skating is an elegant and highly efficient form of human locomotion. Even at high speeds, muscles move slowly and can thus exert a large amount of force for propulsion. However, as anyone who has ever tried to roller-skate will attest, mastering this skill requires a high degree of motor coordination, and one that is radically different from other, more native, forms of movement. Unlike walking, skating does not produce ground reaction forces aligned with the desired direction of motion; it is precisely the absence of friction in this direction that makes skating so elegant and efficient. We demonstrate two types of natural skating motions that our optimization automatically generates: swizzling and stroking.

Swizzling is a skating technique where wheels remain in constant contact with the ground. This technique is based on wave-like motions of the feet which exploit the directional friction of the wheels and cause the whole body to be propelled forward. Our motion optimization process discovered swizzling motions automatically for robot designs that feature only one passive wheel per leg, as shown in Fig. 2.6 and in the accompanying video [33].

Stroking is a slightly more advanced technique that requires the legs to be lifted off the ground periodically. This is necessary when several wheels are attached to a foot, since this type of design makes it impossible for end effectors to rotate about the vertical axis without sliding while they are touching the ground. This mode of locomotion requires the robot to place its back leg on the ground, orienting the wheels in a direction almost orthogonal to the direction of motion, while the front leg glides forward. A stroke of the back leg pushes the body forward. Next, the legs switch placement and the motion is repeated. Fig. 2.1(left) depicts a robot that automatically learns to perform this type of movement. We note that to obtain this motion, it is necessary to provide an input gait (e.g. a walk or trot) that then defines the stroking pattern.

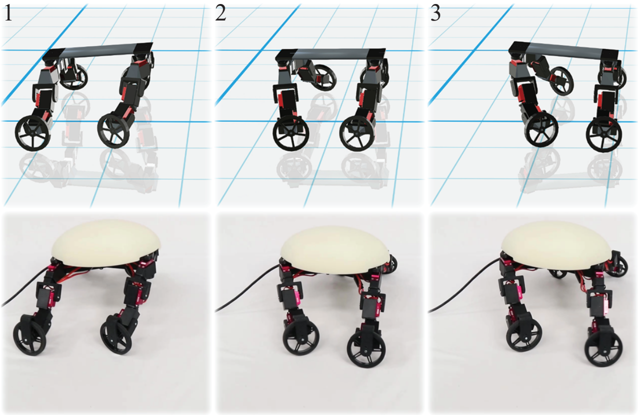


FIGURE 2.6: Swizzle.

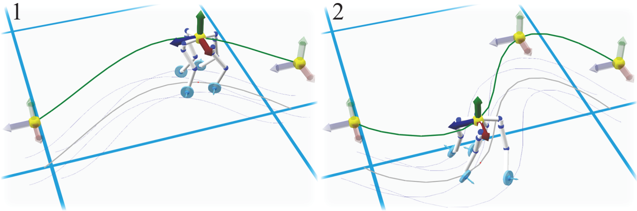


FIGURE 2.7: A demonstration of a slalom motion design. The user can quickly place position objectives and observe the result in real-time.

WELDED WHEELS Welded wheels, which are supported by our design system, enable a more general representation of physical robot feet as compared to the widely used point foot model. Wheels can either be welded permanently to a body part of the robot (e.g. they are printed together as one piece), or they can be actuated wheels that are actively blocked. We use the former option to demonstrate a design with three legs that end in welded wheels and one passive wheel, as shown in Fig. 2.1(middle). The results video [33] includes a real-time demonstration of the motion optimization process for this robot. As can be seen, optimal motions are generated in response to the user changing the desired speed and turning rate.

2.6.2 *Interactive Design*

The improved performance of our motion optimization algorithm allows the user to interactively specify motion choreography objectives. A wide variety of motions can therfore be created in a very short time-span, and the user may fine-tune the movements of the robot to any desired degree. One example, shown in Fig. 2.7, is a slalom motion, which also benefits greatly from the versatility of wheeled legs. For this example, the user simply specifies a desired speed and provides several lateral target positions for the center of mass at different moments in time. The optimal motion is generated almost instantaneously.

2.6.3 *Design optimization*

As discussed in the previous Section, the three different design editing modes assist users with the non trivial and often unintuitive task of optimizing their robots' physical dimensions. These editing modes are supported by our efficient motion optimization method which allows the user to vary design parameters and observe the change in motion immediately.

Semi-automatic optimization allows the user to keep certain aspects of the motion unchanged while manually tweaking their design, as shown in Fig. 2.4. Fully automatic design optimization is used both to fix subtle flaws, as discussed in Sec. 2.5, or to make more drastic changes that enhance a robot's ability to generate the motions envisioned by the designer. Fig. 2.8 presents convergence plots for both of these use cases. For both plots we show the energy of the optimal motion corresponding to the morphological parameters at each design optimization step. For the car example, shown on the left, the initial design was unable to move forward due to the flaw we identified earlier. After three design optimization steps, the flaw was successfully fixed. The example on the right corresponds to the use case shown in Fig. 2.5. Here, user-specified choreography objectives asked the robot to move in ways that were incompatible with its initial design. After one design optimization step, this robot could already reach its target, while subsequent iterations further reduced the total energy of its motion.

2.6.4 *Fabrication*

Fig. 2.10 shows a collection of robots designed with our system. For all these robots we also designed a variety of motion plans that were validated in a

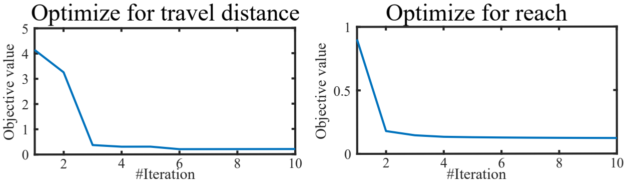


Figure 2.8: Convergence graph for automatic design optimization.

physically-simulated environment. Table 2.1 provides statistics regarding the complexity of all the robots shown in the results video [33]. As can be seen, generating each motion takes only a few seconds of compute time.

We validate our designs through three physical prototypes which are shown in Fig. 2.1. To create these robotic creatures, we used off-the-shelf micro-sized servo motors (e.g. *Turnigy TGY-306*), the *Maestro* USB controller board from *Pololu*, a standard 7.4V battery and 3D printed connectors (e.g. limb segments). Fig. 2.9 shows all the components used for the leg of one of our robot prototypes. For each of our designs, the connectors took less than 24h to 3D print on a Stratasys F370 machine. Assembling, calibration and testing took an additional 2-4h.

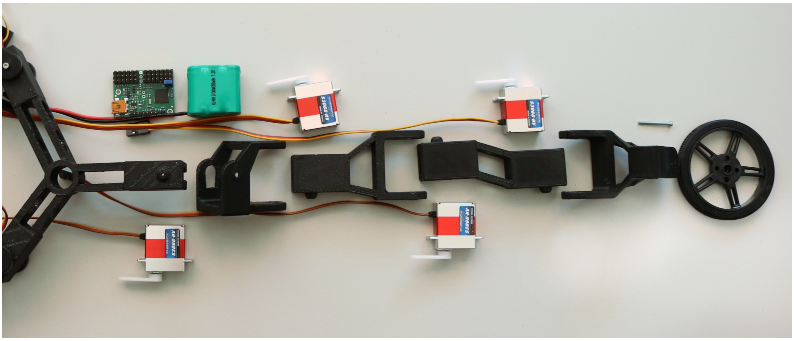
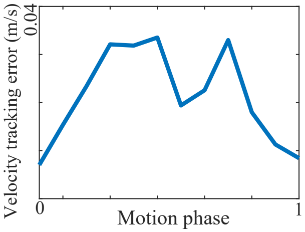


Figure 2.9: A robot leg before assembly.



Micro-sized servos are small, light and easy to work with. However, they are also limited in terms of the torque they can produce, which is particularly problematic for robots with long legs, and thus large moment arms. For this reason, while we noticed that our smaller robots were able to reproduce the motions generated

through optimization quite well, the *Skatebot*
design was noticeably slower as compared to the simulation results. Our initial wheel design, which relied largely on 3D printed parts (e.g. no ball bearings), further contributed to discrepancies between the motions of the physical robot and its simulated counterpart. To quantify the degree to which our robots were able to follow the optimized motion plans, we measured the relative difference in speed for Agilebot as it tracked the slalom motion (Fig. 2.7). As can be seen in the inset figure, the relative error in the velocity of the COM is less than $4cm/s$ (about 6% of the speed of the motion plan). We note that this tracking error is recorded as the robot operates in a physically-simulated environment, which bypasses mismatches arising from sub-optimal hardware components.

Robot	# joints	# end-effectors	motion	motion goals	# time samples	# motion parameters	planning horizon	opt. time
Swizzlebot	12	4	swizzle forward	$d = 0.5\text{m}, \alpha = 0°$	12	648	1.2s	4.5s
			swizzle turning	$d = \text{any}, \alpha = 20°$	12	648	1.2s	3.4s
Creature3	9	4	walk	$d = 0.1\text{m}, \alpha = 0°$	12	504	0.8s	1.3s
			fast walk	$d = 0.3\text{m}, \alpha = 0°$	12	504	0.8s	2.5s
			turning	$d = \text{any}, \alpha = 20°$	12	504	0.8s	1.5s
Brainbot	9	3	swizzle forward	$d = 0.2\text{m}, \alpha = 0°$	24	1008	2s	5.7s
			swizzle forward & turn	$d = \text{any}, \alpha = 20°$	12	504	2s	5.9s
Agilebot	16	4	start & stop	$d = 1.0\text{m}, \alpha = 0°$	12	696	0.8s	5.2s
			turn in place	$d = 0.0\text{m}, \alpha = 0°$	12	696	1.6s	1.1s
			slalom	choreography	24	1392	1.6s	-^{1}
			side-step and roll	choreography	36	2088	3s	-^{1}
Skatebot	16	8	two-leg skating	$d = 0.3\text{m}, \alpha = 0°$	12	696	1.2s	11.3s
Dinobot	17	8	skating	$d = 0.8\text{m}, \alpha = 0°$	12	1140	2s	11.0s

TABLE 2.1: With our system, the user can create various robot designs and easily generate motions satisfying different motion goals.In the fifth column, d defines the desired distance travelled, and α the desired turning angle. Column 7 displays the number of motion parameters which equals the size of $\mathbf{m}$.
^{1}: Motion is generated at interactive rates, while user edits choreography objectives.

2.7 DISCUSSION

We presented a novel design system for a rich class of hybrid legged/wheeled robotic creatures. Thanks to our versatile trajectory optimization formulation, physically-valid walking, rolling, gliding and skating motions arise

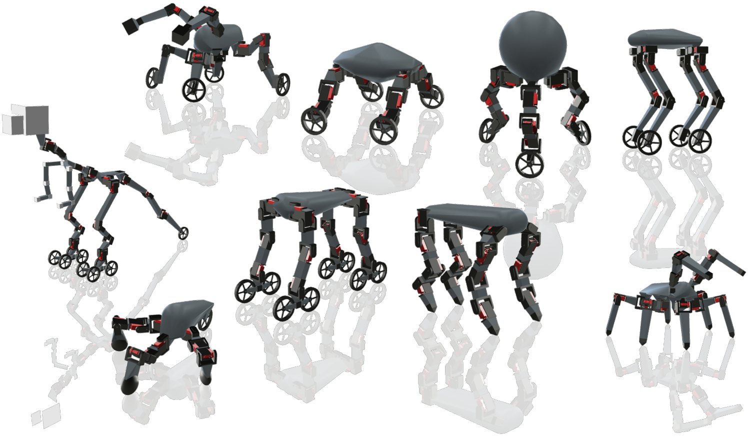


FIGURE 2.10: A variety of unique legged/wheeled robots designed with our system.

naturally as a function of the design characteristics of different robots. We further showed how to leverage the structure of our formulation to significantly accelerate the underlying numerical solver. This, in turn, allows designers to interactively choreograph compelling motions. Given that motor capabilities and the design of a robot are inseparably intertwined, we also developed a suite of user-guided computational tools that support manual, semi-automatic and fully automatic optimization of the robot's physical dimensions. We demonstrated the effectiveness of our method by creating a variety of unique robot designs, three of which we fabricated.

Our results highlight exciting avenues for future work. For example, we are encouraged by the significant improvements in convergence rates achieved through the Filtered Hessian strategy described in Sec. 2.4.3. Our ultimate goal here is to achieve faster than real time performance for the trajectory optimization process. This will allow motion plans to be computed on-line, taking into account dynamic environmental obstacles, and capable of providing full-body feedback strategies in response to unplanned disturbances.

The initial physical prototype for our SkateBot robot moves noticeably slower than the simulation model. Our hope was that better engineered physical wheels (i.e. equipped with professional-grade ball bearings rather than 3D printed parts) and higher performance actuators would help close this gap. This turned out to be the case: The improved hardware of AgileBot can skate much faster, and perform other compelling motions at higher speeds compared to Skatebots and its siblings. The control of all of these robots is however still done in an open-loop fashion. A closed-loop controller could make these robots much more robust against external disturbances. It may also allow them to traverse rough and unknown terrain, where legged-wheeled robots might have a particular advantage as they combine the agility of legs and speed of wheels.

There are also interesting opportunities to improve our motion optimization model to allow the modeling of compliance and slipping, which is currently outside the capabilities of our model. This would allow for a new breed of agile robots that move with the grace of their biological counterparts.

3

DESIGN AND FABRICATION OF LEGGED-WHEELED ROBOTS

In the previous Chapter, we introduced a computational framework for designing robotic creatures and generating motions. In this Chapter, we will use this framework in an attempt to unleash the full potential of legged-wheeled robots. The combination of legs and wheels requires novel modes of locomotion, and robots must therefore be designed to support these complex whole-body motions. Our computational framework can accurately predict how different design decisions will affect the robot's ability to move, and thus serves as an important tool to explore new morphologies for mobile robots. We introduce a novel warm-start method that is critical to our rapid prototyping design approach. It leverages ideas from numerical continuation to dramatically improve the convergence rates for the trajectory optimization routine. This allows us to efficiently explore different robot morphologies and their locomotion modes. The result is AgileBot, a four-legged robot that can walk, roll, and skate. Compared to the fabricated prototypes from the previous Chapter, this robot has servomotors with higher torque and an untethered lightweight design. In addition, we apply our computational method to generate motions for a robot with a feedback control system, Anymal, which is equipped with actuated wheels.

Adapted versions of this Chapter have been published in a peer-reviewed academic journal as Geilinger, M., Winberg, S., and Coros, S. (2020). A computational framework for designing skilled legged-wheeled robots. *IEEE Robotics and Automation Letters*, 5(2), 3674-3681.

3.1 INTRODUCTION

Legged-wheeled robots hold the promise to deliver dexterous mobility while at the same time being fast and efficient. Combining legs and wheels, however, poses interesting new challenges: a robot's morphology determines the range of motion it can perform, and different types of end-effectors govern the way in which it can move. This relationship between mechanical design and locomotion capabilities becomes particularly complex when we want to exploit the combined advantages of legs and wheels.

Because of these advantages and challenges, legged-wheeled locomotion has become an increasingly active area of research in recent years. While most advances in legged-wheeled locomotion have focused on the study of predefined robot designs and their locomotion modes in isolation, our long-term goal is to develop a general computational framework that can serve as a powerful tool for the development of new types of hybrid mobile robots. To this end, we are evaluating the predictive power of our design and motion synthesis system. At the technical level, we introduce a novel warm start technique to dramatically improve the reliability and convergence of the trajectory optimization routine used to synthesize locomotion behavior. This technique builds on ideas from numerical continuation and is particularly effective for motions that combine walking and rolling behaviors. We also build and analyze a novel legged-wheeled robot to validate our computational framework. Equipped with different types of end effectors (i.e., actuated or unactuated wheels), this robot is capable of performing a variety of interesting locomotion modes that take advantage of the combined benefits of limbs and wheels.

3.1.1 *Contribution*

We demonstrate the advantages of our computational system as a predictive tool for the design of legged-wheeled robots by highlighting the important relationship between the design of a robot and its locomotion. Our main contributions are:

- A numerical continuation method for efficiently warm-starting the motion synthesis process for robot designs created with our interactive program. This warm start procedure drastically improves the robustness and convergence of the trajectory optimization routine used by our framework.

- As a result, different robot morphologies can be tested very quickly in simulation, leading to an efficient exploration of the space of locomotion modes that different configurations of legged-wheeled robots are capable of.

- To validate our simulation results, we created two legged robots shown in Fig. 3.1, which have five actuated degrees of freedom per limb. Equipped with different types of end effectors-active or passive wheels and ice-skates-these robots are capable of performing a range of interesting motions, including walking, roll-walking, roller-, and ice-skating.

Our results suggest that a robot engineering system such as the one used in our work can become an invaluable tool in discovering robot designs specifically engineered for different tasks.

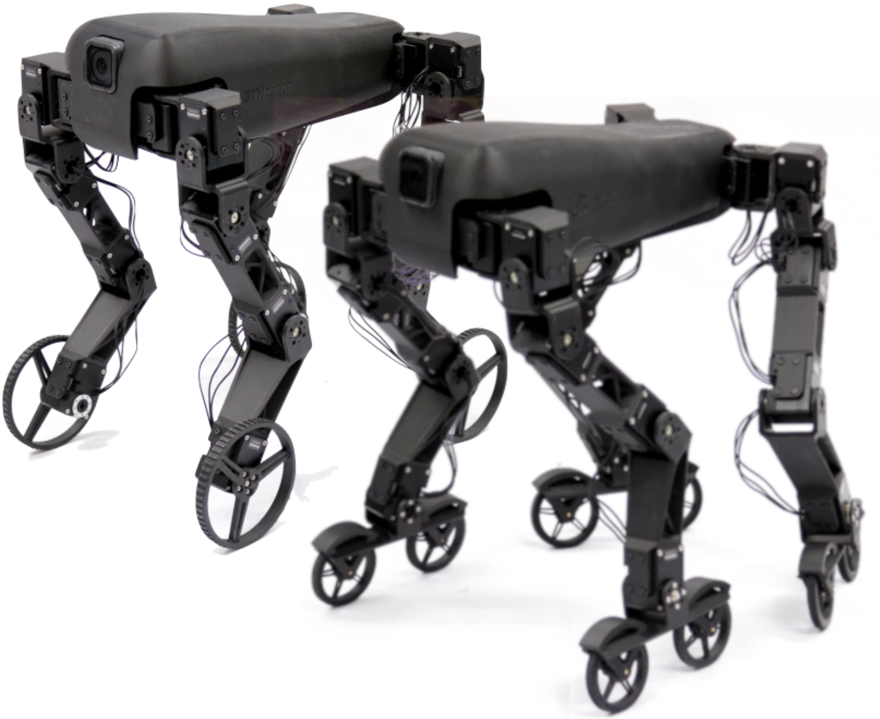


FIGURE 3.1: AgileBot and SkaterBot, two robots created with the help of our interactive design system.

3.2 RELATED WORK

Motion planning and control of legged robots has a rich history in the field of robotics, as these machines show promise in navigating unstructured environments. Nevertheless, wheeled and tracked robotic systems enjoy widespread use in industry as they are more efficient, stable, and easier to control reliably. It comes as no surprise, though, that researchers are also focused their efforts on the challenge of creating hybrid robots that combine the advantages of wheels and legs. For example, the robot presented in [3] uses a manually designed gait that exploits limb articulation and unactuated wheels to create a very particular type of wheeled locomotion. Another interesting robot design that combines legs and wheels is presented in [39]. Here, the robot uses its legs to actively control its support polygon while driving with motorized wheels, allowing it to make aggressive start-stop and turning motions. Boston Dynamics ’ Handle robot [40], one of the most recent examples of a hybrid legged-wheeled system, displays an incredibly rich repertoire of agile behavior, although the technical details behind this platform and its control system remain unpublished.

Traditional legged robot designs are well-studied. However, equipping them with different types of end effectors opens up interesting avenues for further investigations. For example, the quadrupedal robot Anymal was recently retrofitted with different types of wheels [4, 41], and even ice skates [42]. The Anymal team also developed customized motion planning and whole-body control techniques for the emerging mechatronic systems. Their efforts in this area point to an important challenge. While Anymal was able to move using motorized wheels reasonably well [41], it had very limited mobility with skates [42]. We hypothesize that this important limitation is due to Anymal’s morphoholgic design, which was intended for traditional legged locomotion: the types of end-effector trajectories required for skating may be difficult, if not impossible, to generate with Anymal’s current limb design. This observation underscores the need for an interactive design system such as the one we describe in our work.

The design of hybrid robots with wheels and legs requires special considerations, as do the techniques used to control them. For conventional robots with legs, there are already established techniques for motion planning. In general, existing techniques must make a trade-off between computational efficiency and the accuracy of the models used for motion planning. For example, the framework FROST developed in [43] considers the whole-body dynamics of a robot, but requires about 5 to 10 minutes of computation to generate optimal motions. At the other end of the speed/accuracy spectrum, state-of-the-art model predictive control was developed for the MIT cheetah robots [44], which requires about one millisecond of computation time. To achieve this computational efficiency, a linearized model of the centroidal dynamics is used, and the robot’s root and end-effector trajectories are computed using heuristics rather than being treated as parameters in a trajectory optimization routine. The technique we use to generate motions for hybrid robots lies between these two extremes. We use a centroidal model to capture the dynamics of the robots, and simultaneously optimize the body and end-effector trajectories, ground reaction forces, and whole-body poses, similar to the method described in [45]. Our trajectory optimization formulation is complemented by several constraints that model the way different types of end effectors can move when in contact with the ground. As we describe in this Chapter, our trajectory optimization process converges in a few seconds when initialized with the solution of a novel warm start strategy, and allows for an interactive exploration of the motion capabilities that different robot designs will possess.

Through a series of sim-to-real experiments, we have found that our motion optimization model is sufficiently powerful to generate a variety of dynamic maneuvers, such as rolling under obstacles, roller skating, and ice-skating. Nevertheless, we note that for motor skills that are less dynamic in nature, motion planning techniques that only consider only the kinematic design of a robot [46–49], or approaches that use a simplified dynamics model based on the zero moment point [41, 50] have also been proposed. Although these methods are potentially faster, they have not been shown to be able to produce the types of motor skills that we demonstrate in our work.

3.3 WARM-START ROUTINE FOR MOTION GENERATION

The main goal of our computational framework is to provide an intuitive interactive approach to creating, evaluating, and improving robot designs based on a desired set of locomotion skills. This is only possible if the entire design and motion generation pipeline is reliable and efficient. The system presented in Section 2 generates full-body robot motions $\mathbf{m}$ by finding a local minimum of $E(\mathbf{m})$, a function that encodes various motion goals and constraints, as described in Section 2.4. This function is characterized by a strongly nonlinear and nonconvex landscape, so it has many undesirable local minima in which standard gradient-based optimization methods are prone to getting stuck. This challenge is particularly pronounced for motions that exhibit intermittent contacts with the environment, and the initial guess $\mathbf{m}_0$ from which the optimization routine starts plays a crucial role in determining the local minimum to which the final solution $\mathbf{m}$ will converge. In this Section, we therefore turn to the formalization of a principled method for generating good initial solutions $\mathbf{m}_0$ for arbitrary robot morphologies moving with arbitrary gaits.

#	motion parameters	constraints and objectives	goal
1	$\mathbf{f}$	dynamics	robot stands on ground with all end effectors in stance
2	$\mathbf{f}, \mathbf{c}$	dynamics, zero $\mathbf{f}$ in swing*	robot's body moves according to footfall pattern
3	$\mathbf{q}, \mathbf{e}$	kinematics, vertical end effector position trajectories*	robot poses that satisfy kinematics and swing trajectories
4	$\mathbf{q}, \mathbf{c}, \mathbf{e}, \mathbf{f}, \omega, \boldsymbol{\alpha}$	all constraints, motion targets*	motion plan with full set of DOFs and achieving motion targets

* introduced iteratively

Table 3.1: Outline of warm-start strategy

To gain intuition into the method we propose, consider the simple case of a quadrupedal robot that must move with a periodic walking gait. The motion of the robot body must be appropriately synchronized with the corresponding footfall pattern. If the robot's center of mass is inadequately positioned at a certain point in the motion cycle, the robot will lose its balance when lifting the next foot and eventually fall. Such an initial guess for a motion plan corresponds to a very large objective value for the trajectory optimization objective $E(\mathbf{m})$, and a very large number of iterations may be required before a dynamically stable motion is found. In contrast, it is trivial to generate a motion where the robot simply stands in place. Our observation is that we can start from this trivial solution and formulate regularizing objectives that ask the robot to gradually unload its feet according to the desired gait. Since the robot does not need to lift its feet during this initial optimization process, it can maintain balance while it figures out how to coordinate the motion of its body with the underlying footfall pattern. Essentially, the robot first learns how to walk in place. We now show how to mathematically formalize this intuitive concept using a numerical continuation approach [51].

STEP 1 To generate a favorable initial starting state $\mathbf{m}_0$ for the trajectory optimization routine described in the previous Section, our warm start method starts with a subset of the motion plan parameters and gradually introduces new degrees of freedom and constraints. In the first step, the footfall pattern and the corresponding constraints on the ground reaction forces are ignored. Then, the ground reaction forces satisfying the constraints of the centroidal dynamics $\mathbf{C}_d$ given by the equations (2.3) - (2.5) for a quiet standing behavior:

$$\mathbf{f} = \text{argmin}_{\mathbf{f}^*} \frac{1}{2} \mathbf{C}_d^T \mathbf{C}_d$$

STEP 2 The next step of the warm start routine aims to synchronize the motion of the robot body with the constraints imposed by the footfall pattern on the set of feasible ground reaction forces. In particular, the end effectors are not allowed to apply ground reaction forces during the prescribed swing phases. To introduce these constraints incrementally, we solve a sequence of N_1 optimization problems of the form:

$$\{\mathbf{f}, \mathbf{c}\} = \text{argmin}_{\mathbf{f}_i^*, \mathbf{c}^*} \frac{1}{2} \mathbf{C}_d^T \mathbf{C}_d \text{ s.t. } (1 - c_i) \cdot \mathbf{f}_{\mathbf{i}}^* \frac{N_1 - k}{N_1 - 1} \mathbf{f}_i^1 = 0$$

where i denotes the index of end effectors, N_1 is the total number of iterations, $k \in \{0, N_1 - 1\}$ is the index of the current iteration, and $\mathbf{f}^1$ is the set of ground reaction forces calculated in step 1. At the beginning of this iterative process, the robot has the greatest control authority because the end effectors that are in swing (i.e., $c = 0$) can still generate ground reaction forces that affect the robot's full-body motion. As k approaches N_1, these forces are progressively driven toward 0, resulting in body trajectories that are smoothly optimized to anticipate the swing phases before they actually occur. We have experimentally found that $N_1 = 100$ works reliably for a variety of robot designs and footfall patterns, and we use this setting for all of our results.

STEP 3 The intermediate motion plan created in Step 2 satisfies the constraints imposed by the centroidal dynamics and the footfall pattern. Next, our goal is to find full body robot motions $\mathbf{q}$ that match the trajectories of the body and end effectors to satisfy the kinematic consistency constraints $\mathbf{C}_k$ defined in equations $(2.11) - (2.14)$. To smooth out this optimization problem, we gradually increase the swing height from 0 to the user-defined value e_y^{des} and solve a sequence of subproblems:

$$\{\mathbf{q}, \mathbf{e}\} = \text{argmin}_{\mathbf{q}^*, \mathbf{e}^*} \frac{1}{2} \mathbf{C}_k^T \mathbf{C}_k + \frac{1}{2} || e_y - \frac{k}{N_2 - 1} e_y^{\text{des}} ||_2^2$$

Where N_2 is the number of iterations, $k \in \{0, N_2 - 1\}$ is the current iteration, e_y is the vertical component of the contact point, and is the desired swing height trajectory.

STEP 4 The steps leading up to here have found a subset of motion parameters $\mathbf{f}, \mathbf{c}, \mathbf{q}, \mathbf{e}$ that satisfy the most challenging constraints. The final step of the warm start routine now solves the full set of motion parameters and considers all constraints. The values for all motion objectives (i.e., travel or turning speed) are set to 0 in this step, so that the resulting motion corresponds to the robot locomoting in place. This motion constitutes the initial guess $\mathbf{m}_0$, which is fine-tuned according to the user-specific inputs by the trajectory optimization described in the previous Section.

3.4 ANALYSIS

3.4.1 *Evaluation of Warm-Start Routine*

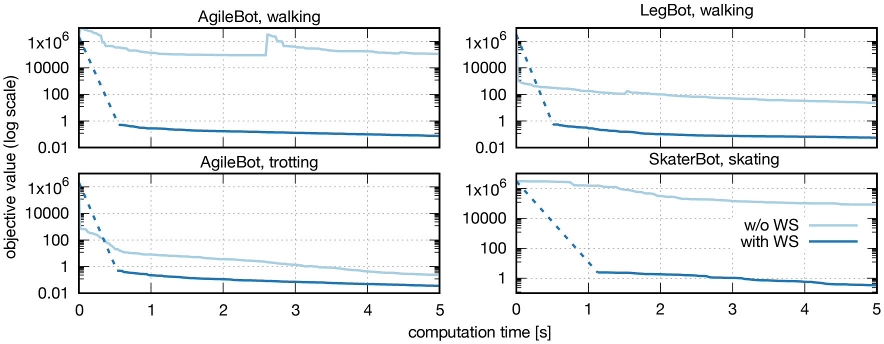


FIGURE 3.2: Convergence improvement when using the warm start routine. The dashed lines denote the warm start routine, which greatly improves convergence speed.

The goal of the warm-start strategy we introduced in Section 3.3 is to appropriately initialize the non-linear program used to generate optimal robot motions. As shown in Fig. 3.2, the convergence rates for the trajectory optimization process are greatly improved when using the initial solution obtained through warm-starting. We note that the trajectory optimization process can converge without warm-starting as well, but it is typically much slower. For AgileBot (quadruped robot, 16 joints, 4 actuated wheels) performing a trotting gait, for example, the trajectory optimization process takes about 9.5 seconds of compute time to reach the same objective value as the one obtained with warm-starting after just 5 seconds. However, the walking gait for the same robot fails to converge within 1 minute when warm-starting is omitted.

3.4.2 *Morphology exploration for hybrid legged/wheeled robots*

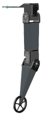
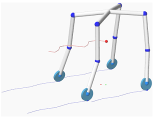
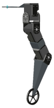
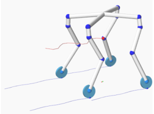
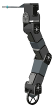
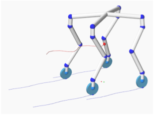
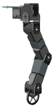
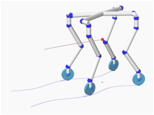
leg design		motion trajectories	avg. joint angle velocity $[\frac{\text{rad}}{s}]$	avg. joint angle accel. $[\frac{\text{rad}}{s^2}]$
2m			4.87	4408.22
3m			3.26	2572.32
4m			3.07	1870.08
5m			0.68	23.93

TABLE 3.2: Differences in SwizzleBot designs and motions

Traditional designs for legged robots with point feet are optimized to provide a sufficiently large reachable space for its end effectors. Adding passive or active wheels to a robot's feet demands further considerations in terms of the space of reachable end effector orientations, as this directly governs the functional capabilities of the final robot design. A robot whose wheels are actuated, for example, will need to move its limbs as it locomotes in a very different manner than one that propels itself using passive wheels or ice skates. Indeed, a robot design that is not compatible with a specific set of end effectors and desired motion skills may function poorly [42], or it may fail to locomote altogether.

Naturally, the question then arises: how does the morphological design of a robot shape its motor capabilities? We begin to study this question by using our computational framework to analyze a quadrupedal robot design that is equipped with one unactuated wheel per end effector – the SwizzleBot. Thanks to our warm start routine (Section 3.3) we can quickly explore the space of eventual robot morphologies. In Table 3.2 we compare four different leg designs for SwizzleBot, with 2, 3, 4 and 5 motors per limb respectively. For each design we used our warm-starting and trajectory optimization routines to generate full-body motions using a desired speed of $0.3m/s$ as a target. All four robot designs successfully reached this target forward speed. However, the effort required to move as desired is vastly different. This can be seen in the last two columns of Table 3.2 which show the joint angle velocity and acceleration averaged over a motion cycle.

Indeed, the robot designed with five joints per limb stands out by being able to locomote with much smoother motion trajectories compared to the other three robot designs. To further clarify table 3.2, we refer the interested readers to the video [52] for a side-by-side comparison of all four designs. We note that creating these designs and their corresponding motions took only a matter of a few minutes with our computational framework.

3.5 FABRICATED RESULTS

3.5.1 *From conceptual designs to physical prototypes*

In order to test the results generated with our computational framework and motion optimization process, we built two quadrupedal robots that can be equipped with actuated or unactuated wheels as end effectors. In our computational framework, Figure 3.3, the user can (1) create arbitrary robot designs, (2) generate and interactively edit motion plans in the motion editor, and (3) try the resulting control trajectories in a physics simulator or transfer them wirelessly to a physical prototype. This process can be seen in the video [52].

Both of our robots are made using 3d printed parts and off-the-shelf servomotors and electronic components. For 3D printing we used a Markforged Two machine that reinforces digitally manufactured parts with long strands of carbon-fiber. For actuation we used Dynamixel servomotors (XM430-W210T and XM540-W150T), both for the robot's joints and to power active

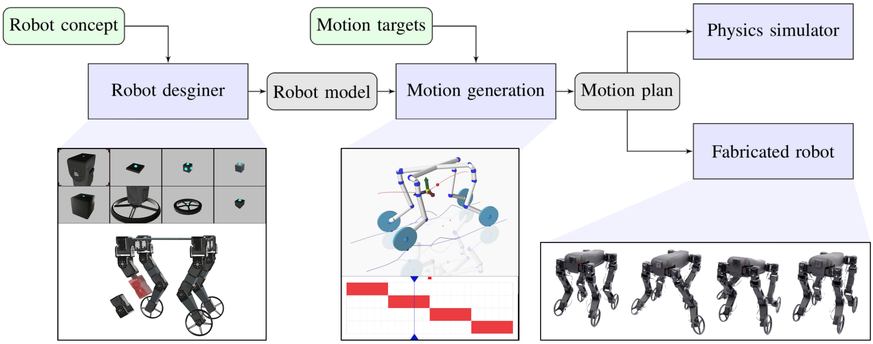


FIGURE 3.3: This figure depicts the computational framework we use to design hybrid legged/wheeled robots: Left: Users can generate arbitrary robot designs by mixing together different building blocks using an interactive graphical interface. Middle: Physically-correct locomotion behaviors are generated through trajectory optimization. Right: These motions can then either be tested in simulation, or they can be transferred wirelessly to the physical robot.

wheels. An onboard Raspberry Pi 3B+ was used to send control signals to the servomotors. Fig. 3.4 shows some of the components we used to fabricate the physical robot prototypes. 3d-printing the structural components of the robot designs took about one week, while the assembly process took an additional half day.

Motion plans are generated offline on a desktop PC, sampled at 60Hz and then transferred to the Raspberry Pi wirelessly. A touch screen is mounted on the robot to provide an interactive interface that allows the robot operator to load new motion plans, combine multiple motions into a motion graph, and to set various servomotor parameters (e.g. the proportional and damping gains of the low-level PD controllers). We further use an XBox gamepad controller to give high-level commands to the robot (i.e. which generated motion to execute). This intuitive and streamlined process makes it possible to try out the motions and behaviors generated with our framework in a matter of seconds.

3.5.2 *Robot Designs*

AGILEBOT - A LEGGED ROBOT WITH ACTUATED WHEELS The first robot we designed is a quadruped that has four joints and one actuated wheel per

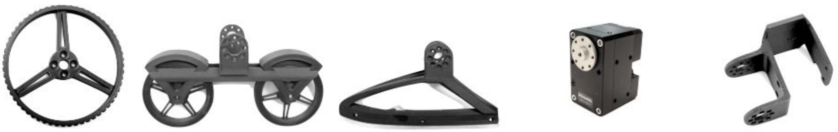


Figure 3.4: Our robots are made from 3d-printed parts and off-shelf electronic components for actuation and control. From left to right: 3D printed wheels that can be attached to a motor or left to spin freely, a roller blade with passive wheels and an ice skate, Dynamixel XM430-W210T are used to actuation joints and active wheels, and 3d-printed brackets.

limb, Figure 3.1. The joint degrees of freedom allow it to move sideways, turn in-place, move forward or sideways while ducking, and perform other agile motions; the accompanying video [52] shows several motor skills where AgileBot makes efficient use of all its degrees of freedom. Note that all these motions emerge from the trajectory optimization process given only high-level targets as input. For example, in the *forward-duck* motion, we use a simple objective to ask the height of the robot's body to be below a user-specified threshold. Additional objectives can ask, for example, that the relative motion between the wheel and the body part it is attached to vanishes, essentially resulting in a design where the virtually welded wheel becomes a circular arc-shaped foot. We refer to this design as LegBot.

skaterbot - a legged robot with passive wheels As this design illustrates, equipping a legged robot with passive wheels can lead to fast, energetically favorable locomotion modes. To create SkaterBot, Figure 3.1, we used the same overall morphology as for AgileBot, but we exchanged the actuated wheels with roller blades – pairs of passive inline wheels. When both wheels are on the ground, the corresponding end effector can not rotate around the vertical axis without slipping. Skating motions therefore require the robot to lift its feet as it moves. To create an additional locomotion mode, the ankle joints can be rotated by 45 degrees, resulting in only one wheel of the roller blade being in contact with the ground. We call this design SwizzleBot, and we note that it is functionally equivalent to the ice skating robot that we also experimented with, given that the ice skates we employed have non-zero curvature. This robot design can propel itself without ever having to lift its feet.

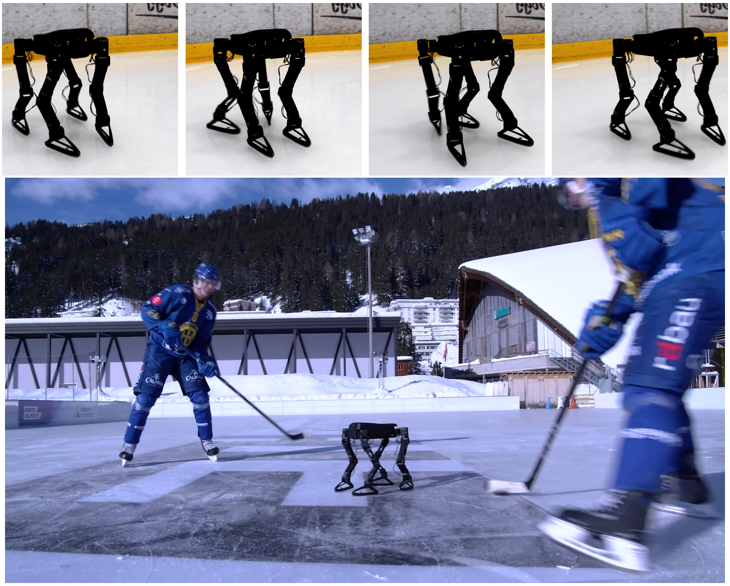


FIGURE 3.5: Top: Skaterbot on ice performing a swizzle motion. Bottom: Experiment at high altitude, 1,560 meters aboce sea.

Instead of roller-blades, Skaterbot can also be equipped with ice-skates. The blades are slightly curved, which results in a single contact point with the ice. Since friction perpendicular to the blade is much higher than in the direction of the blade, we thus model ice-skates as unactuated wheels. In Figure 3.5, we demonstrate that the swizzle motion leads to successful locomotion on ice, and combined with left- and right-turning motions, Skaterbot can enjoy the pleasures of effortlessly gliding on frozen water.

3.5.3 *Performance Evaluation*

We used our trajectory optimization framework to generate a variety of locomotion modes for AgileBot, SkaterBot, LegBot and SwizzleBot. For each of the motions summarized in Table 3.3, which can also be seen in the accompanying video [52], our goal was to maximize speed. For the AgileBot's walking gait, we enforced the constraint that ensures the speed of the wheels is always zero, and we set the footfall pattern accordingly. For the rolling motion, the feet were set in stance mode for the entire duration of the gait cycle. For the walking and rolling behavior, the wheels were left

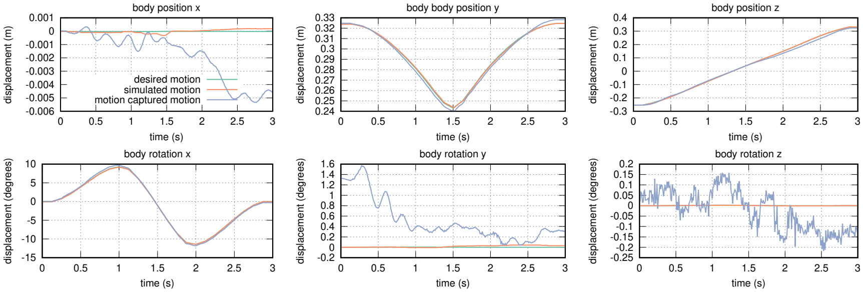


Figure 3.6: Comparison of generated, physically simulated and motion-captured trajectories for AgileBot. The motor skill we evaluate here is the "forward-duck" motion, which can be seen in the accompanying video [52]. The simulated and motion-captured trajectories match the motions generated through our optimization model very well. (Note the difference in scales of the vertical axes.)

free to rotate as best determined by the motion generator, and a walking footfall pattern was provided as input. We note that the speed limit of the wheels imposes strict constraints on the overall movement speed of the robot.

As our results show, the designs that feature passive wheels are much faster than AgileBot. This is particularly noteworthy because we used exactly the same types of servomotors to power the joints of the two robots. The skating motion, in particular, is more than twice as fast as AgileBot's roll-walking locomotion mode.

Fig. 3.6 compares the motions generated by our trajectory optimizer against the motions executed by the physical robot. For this sim-to-real experiment, we measure differences in the motion of AgileBot's body as it performs the "forward duck" maneuver that can be seen in the accompanying video [52]. We use an OptiTrack motion capture system to record the motion trajectories executed by physical robot prototype. As can be seen, the planned position and orientation body trajectories for this relatively complex maneuver are tracked effectively both in simulation, and in the real.

3.6 MOTION GENERATION FOR ANYMAL ON WHEELS

Due to its particular morphology, AgileBot can perform a variety of compelling and dynamic motions. However, the joint angle trajectories are

Robot	measured max. speed
AgileBot walking	0.1483 m/s
AgileBot rolling	0.3380 m/s
AgileBot walking & rolling	0.4762 m/s
SwizzleBot	0.6897 m/s
SkaterBot	0.9677 m/s

TABLE 3.3: Maximum Speed for Different Designs

executed in open loop and tracked by the proportional-derivative controller integrated in each servomotor. The robot's ability to respond to perturbations is therefore severely limited, which has a particular impact on dynamically demanding motions. In this Section, we investigate whether the control trajectories generated by our trajectory optimization can be used as reference trajectories in a closed-loop feedback control system.

To this end, we combine our motion generation system with the locomotion controller developed by Bjelonic et.al. [53, 54], which is capable of tracking reference trajectories for legged-wheeled robots.

3.6.1 *Overview of Feedback Control System*

We give a brief overview of the control system developed in [54]. The locomotion control system has a hierarchical structure. First, a motion planner generates dynamically feasible trajectories guided by higher-level motion goals. This process is controlled by a user and executed offline. The resulting trajectories are stored in a *motion library*. The *trajectory composer* then concatenates the trajectories and feeds them into the *model predictive control* algorithm (MPC), which tracks the trajectories based on the current robot state. Finally, the *inverse dynamics* issues torque commands that are sent to the robot. A state estimator predicts the state of the robot used by the MPC and inverse dynamics.

3.6.1.1 *Model Predictive Control*

The basic idea of MPC is to repeatedly generate controls over a longer time horizon and then apply only the current set of control inputs. This allows anticipation of future events while responding to immediate state disturbances.

The MPC problem is formulated as an optimization problem over a cost term that penalizes deviations from the reference trajectories. Various constraints ensure that physically feasible trajectories are generated. The dynamics of the robot is based on a kinodynamic model. Just as in the case of centroidal dynamics, the kinodynamic model defines a single rigid body together with the kinematics to determine the position of the end- effectors. However, the mass properties of the rigid body are fixed and assumed to be independent of the joint configuration. Other constraints enforce the kinematics of the robot, the initial conditions given by the state estimator, and that the trajectories of the end effectors are compatible with the wheel's speed and orientation. The MPC receives the desired robot state, joint angle, and ground reaction force trajectories from the trajectory composer. Based on the current estimated robot state, it then computes optimal controls consisting of ground reaction forces and joint angle velocities.

3.6.1.2 *Inverse Dynamics*

Inverse Dynamics takes as input the state of the robot measured by the state estimator, as well as the ground reaction forces and joint angular velocities coming from the MPC. Forward simulation is used to calculate the desired joint accelerations, which are then converted to torques. The inverse dynamics only considers the current time step, while the MPC has a fixed planning horizon of 1 second.

For more details on the locomotion controller, we refer to [54].

3.6.2 *Evaluation*

We now want to test whether the reference motions generated by our interactive trajectory optimization are a suitable input for the locomotion controller briefly described above. The robotic platform used for the real experiments is ANYmal [55] equipped with non-steerable wheels, similar to [53].

Unlike AgileBot, the space of possible movements that ANYmal on wheels can perform is more restricted due to the combination of joint configuration and wheels. In Section 3.4.2, we explored different morphologies for a four-legged robot with unactuated wheels. ANYmal's leg configuration is similar to that of the robot with 3 joints per leg seen in Table 3.2, where the orientation of the joint axes is forward, sideways, and sideways

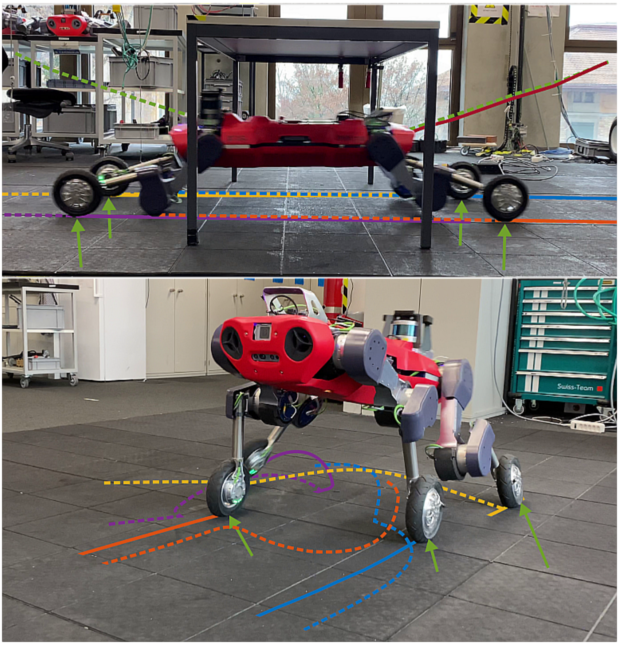


FIGURE 3.7: ANYmal on wheels performing dynamic motions generated with our trajectory optimization. The top image shows a fast ducking motion. The bottom image shows a dynamic turning-in-place motion. The green arrows indicate ground reaction forces. The dashed and solid lines show the reference and the actual trajectories, respectively.

from top to bottom. Our results suggest that this joint configuration is unsuitable for a swizzling motion. It is therefore not surprising that the same morphology does not allow a turning motion while keeping all end-effectors on the ground. Thus, to perform a turning motion, ANYmal on wheels must lift at least two wheels off the ground per motion cycle.

We generate various dynamic motions using our motion generation system, feed them to the motion library, and execute the motions on the real robot. Figure 3.7 shows a ducking motion and a spinning motion. As shown in Figure 3.9, we also create a longer motion where ANYmal ducks under a table, rotates 90^{circ}, and then ducks under another table. The control system and the robot are able to perform these dynamic and fast movements without losing balance. In the figure 3.8, we compare the reference trajectories from our trajectory optimization with the trajectories

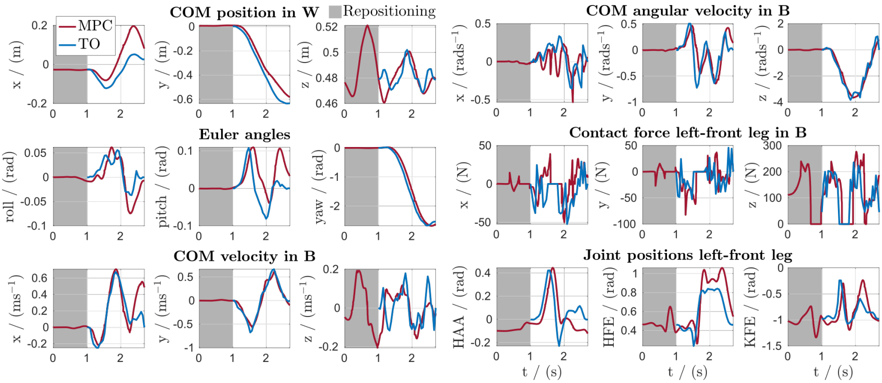


Figure 3.8: Results of the interactive TO in combination with the MPC while executing a repositioning motion ([0-1]s) and the turning motion in Figure 3.7 ([1-2.7]s). The plots compare the optimized whole-body trajectory of both algorithms. Here, the MPC solution is represented by the robot's measured state, which is the equivalent of the initial state vector $\mathbf{x}(t = 0)$ and initial control input vector $\mathbf{u}(t = 0)$. This equivalency is due to the MPC's fast update rate and the reinitialization of its optimization problem after every iteration with the robot's measured state.

tracked by the MPC. The comparison shows that the MPC is able to track the reference trajectories well, even over longer time horizons. This indicates that our trajectory optimization indeed produces physically feasible motions that can be used as input for a feedback control system.

3.7 DISCUSSION

We introduced a novel warm-starting method that drastically improves the robustness and computational efficiency of the trajectory optimization routine that we use to generate optimal motions for legged robots. Integrated into our computational design framework, this warm-starting strategy promotes a systematic, interactive study of the ways in which the morphological design of a legged robot shapes its motor capabilities.

We leveraged our computational design framework to create two physical robot prototypes that are capable of a rich array of motions and locomotory behaviors. The predictions generated by our trajectory optimization are evaluated by real-world experiments. In addition, we showed that our

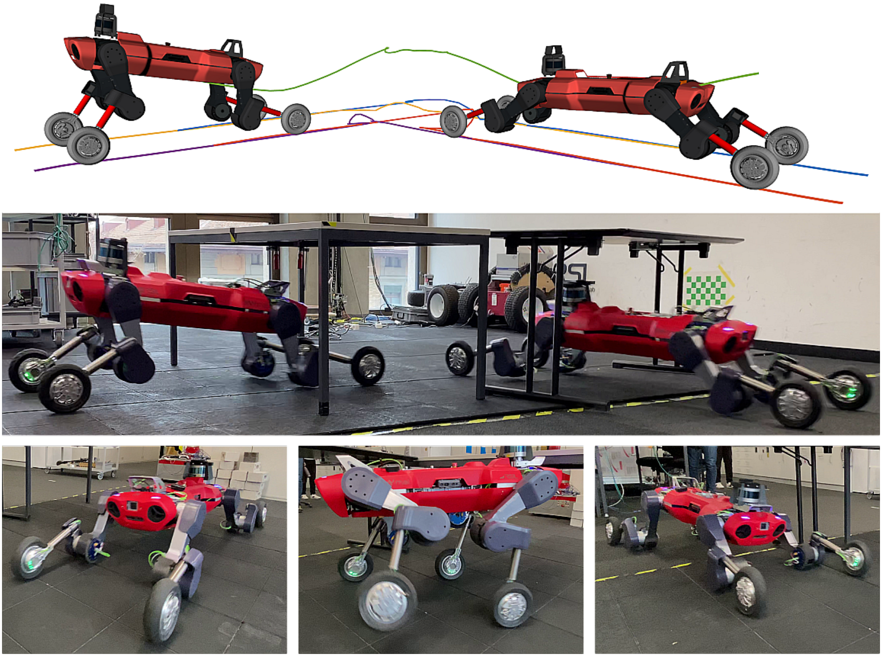


FIGURE 3.9: Anymal on wheels performing a concatenated motion: It ducks under a table, turns in place and ducks again under the other table.

trajectory optimization can be combined with a state-of-the-art closed-loop control system, which allows for even more dynamic and fast motions.

Our current efforts in establishing a computational workflow for robot engineering are not without limitations. We would like to extend our computational design and motion generation framework to support robot morphologies that include mechanical transmission elements (e.g. linkages and drive belts), increasingly complex design features (e.g. kinematic loops that allow a robot's body to reshape as needed for different tasks), as well as soft materials (e.g. rubber feet, flexible links or springs used to store and release energy) that are appropriately accounted for within the trajectory optimization process. These extensions would not only allow for new robot designs, but also benefit the motion generation of existing robots as motors, links and interactions with the ground could be modeled more accurately.

4

DIFFERENTIABLE PHYSICS SIMULATION FOR MOTION SYNTHESIS

Computational tools are becoming increasingly important in robotics. They rely on physics simulation models of the robot and its interactions with the environment. The computational tool presented in Chapters 2 and 3 uses a physics model based on centroidal dynamics and kinematic constraints to model the legs. This model has two main benefits. First, it allows for interactive rates in the motion planning tool. Second, it is expressive enough to capture the dynamics of the robot such that the generated motions transfer to the real world.

However, as we pushed our system to produce more dynamic and agile motions, the limitations of the physical model became more apparent. Most notably, the model is unable to capture the effects of compliant motors or soft feet on the robot's dynamics. Similarly, the capabilities of the motor, such as torque limits, cannot be taken into account when planning a motion. And due to the constraints of non-slip friction, the generation of sliding motions is not possible. In addition, the model does not allow for kinematic loops or deformable components.

In our quest to develop tools that enable the creation of more agile and lifelike robotic creatures, we believe that a more expressive physical model is needed to take advantage of linkages, compliant motors, and hybrid rigid-soft robotic morphologies in both the design and motion planning processes. To this end, we are developing a differentiable simulator for multibody systems with frictional contact. The simulator is targeted at robot applications, such as motion planning for legged robots, parameter estimation, or manipulation of deformable objects.

An adapted version of this Chapter has been published as Geilinger, M.[1], Hahn, D.[1], Zehnder, J., Bächer, M., Thomaszewski, B. & Coros, S. ADD: analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG) 39, 1 (2020)*. David contributed by implementing the simulation model for deformables, conducting the experiments, and writing parts of the manuscript. I developed the multi-

---

[1] joint first authors

body differentiable simulation framework, wrote parts of the manuscript and conducted experiments.

4.1 INTRODUCTION

Simulation tools are crucial to a variety of applications in engineering and robotics, where they can be used to test the performance of a design long before the first prototype is ever built. Based on forward simulation, however, these virtual proving grounds are typically limited to trial-and-error approaches, where the onus is on the user to painstakingly find appropriate control or design parameters. Inverse simulation tools promise a more direct and powerful approach, as they can anticipate and exploit the way in which a change in parameters affects the performance of the design. Unlike forward simulation, this inverse approach hinges on the ability to compute derivatives of simulation runs.

Differentiable simulations have recently seen increasing attention in the vision, graphics, and robotics communities, e.g., in the context of fluid animation [56], soft-body dynamics [57], and light transport [58]. As a characteristic trait of real-world interactions, however, the strong non-linearities and quasi-discontinuities induced by collisions and frictional contact pose substantial challenges for differentiable simulation.

The equations of motion for mechanical systems with frictional contact can be cast as a non-linear complementarity problem that couples the tangential forces to the magnitude of the normal forces. While the exact form depends on the friction model being used, this formulation typically postulates different regimes—approaching or separating and stick or slip—with different bounds on the admissible forces. These dichotomies induce discontinuities in forces and their derivatives, which, already for the forward problem, necessitate complex numerical solvers that are computationally demanding. For inverse simulations that rely on gradient-based optimization, however, this lack of smoothness is a major stumbling block.

To overcome the difficulties of the non-smooth problem setting, we propose a differentiable simulator that combines fully implicit time-stepping with a principled mollification of normal and tangential contact forces. These contact forces, as well as the coupled system mechanics of rigid and soft objects, are handled through a soft constraint formulation that is simple, numerically robust and very effective. This formulation also allows us to analytically compute derivatives of simulation outcomes through adjoint sensitivity analysis. Our simulation model enables an easy-to-tune trade-off between accuracy and smoothness of objective function landscapes, and we showcase its potential by applying it to a variety of application domains, which can be summarized as follows:

REAL2SIM We perform data-driven parameter estimation to find material constants that characterize the stiffness, viscous damping, and friction properties of deformable objects. This enables the creation of simulation models that accurately reproduce and predict the behaviour of physical specimens.

CONTROL Our differentiable simulator is ideally suited for control problems that involve coupled dynamical systems and frictional contact. We explore this important problem domain in the context of robotic manipulation, focusing, through simulation and physical experiments, on behaviours that include controlled multi-bounce tossing, dragging, and sliding of different types of objects. We also use our simulator to generate optimal motion trajectories for a simulated legged robot with compliant actuators and soft feet.

SELF-SUPERVISED LEARNING By embedding our simulator as a specialized node in a neural network, we show that control policies that map target outcomes to appropriate control inputs can be easily learned in a self-supervised manner; this is achieved by defining the loss directly as a function of the results generated with our differentiable simulation model. We explore this concept through a simple game where a robot arm learns how to toss a ball such that it lands in an interactively placed cup after exactly one bounce.

4.1.1 *Contributions*

Our main contributions can be summarized as follows:

- A stable, differentiable, two-way coupled simulation framework for both rigid and soft bodies.
- A smooth frictional contact model that can effectively balance differentiability requirements for inverse problems with accurate modelling of real-world behaviour.
- Application of the differentiable model for trajectory optimization for compliant quadruped with soft feet.

- An evaluation of our friction formulations on a diverse set of applications that highlight the benefits of differentiable simulators.

4.2 RELATED WORK

Contact modelling is a well-studied problem in mechanics (see e.g. [59]), and has received a significant amount of attention in graphics due to its importance in physics-based animation. Here, we focus our review on frictional contact dynamics, its differentiability, and its use in solving inverse problems.

4.2.1 *Frictional Contact Dynamics*

For rigid bodies, early work focused on acceleration-level [60] and velocity-level [61, 62] linear complementarity problem (LCP) formulations and isotropic Coulomb friction. Later, iterative LCP approaches were explored [63, 64], and LCP formulations extended to quasi-rigid objects [65], deformable solids [63, 66], and fast simulation of large sets of rigid bodies [67]. Harmon et al. [68] propose an asynchronous treatment of contact mechanics for deformable models, discretizing the contact barrier potential. Unifying fast collision detection and contact force computations, Allard et al. [69] target deformable contact dynamics where deeper penetrations can arise. Recent work [70] has integrated penalty-based friction models with optimization-based time integration. Exact Coulomb friction paired with adaptive node refinement at contact locations is used for accurate cloth simulation [71]. Anisotropic Coulomb friction in the deformable [72] and rigid-body setting [73] has also been considered.

Like Kaufman et al. [74], our frictional contact formulation applies to coupled rigid and deformable bodies. For this setting, Macklin et al. [75] solve a *non-linear* complementarity problem for an exact Coulomb friction formulation with a non-smooth Newton method. A similar technique was applied to fibre assemblies [76]. However, in contrast to the above body of work, we target inverse contact applications [77–81], rather than animation. Interestingly, our hybrid approach can be interpreted as the opposite of staggered projections [74], as we soften contact constraints with penalty forces, but can enforce static friction with hard constraints. Softening contact has also proven useful for the control of physics-based characters [82]. In the context of identifying frictional parameters, we share goals with Pai et al. [83]. However, in contrast to measuring frictional textures with a robotic system, we aim at characterizing material properties and initial conditions from motion data. For parameter estimation, we draw inspiration from physics-guided reconstruction [84], who consider frictionless contact between rigid bodies. Our formulation extends to frictional contact and deformable bodies.

4.2.2 *Inverse Contact*

Ly et al. [85] solve for the rest configuration and friction forces of a shell model such that the corresponding static equilibrium state best approximates a user-specified target shape. In a first pass, contact is handled using frictionless bilateral constraints whereas frictional forces are found in a second pass. While Ly et al. focus on quasi-static inverse problems, our focus is on inverse *dynamics*. To enable design optimization, Chen et al. [86] introduce an implicitly integrated elasto-dynamics solver that can handle complex contact scenarios. We share the goal of making our frictional contact model smooth and invertible with previous work in optimal control [87–89]. However, instead of relying on finite-differencing and explicit time integration [90], our approach builds on analytically-computed derivatives and fully implicit time integration, allowing us to handle both soft and stiff systems.

4.2.3 *Differentiable Simulation*

Emerging differentiable physics-based simulators are increasingly used for applications in machine learning, physics-based control, and inverse design. While common physics engines such as NVIDIA's PhysX or Bullet Physics enjoy widespread use, they prioritize speed over accuracy, especially in the context of dynamics of deformable objects.

The "MuJoCo" framework [90] has been designed as a differentiable simulator for articulated rigid-body dynamics. However, derivatives are computed through finite differences, and the use of an explicit time integration method in conjunction with a penalty-based contact model restricts the size of the time steps that can be used for reasonably accurate results. Targeting end-to-end learning, de Avila Belbute-Peres et al. [91] compute analytical gradients of LCPs for simple rigid-body systems. Other notable differentiable rigid-body simulations include the work of Freeman et al. [92] and Heiden et al. [93]. In contrast, our approach is able to handle multibody systems that couple rigid and deformable objects and is very robust with respect to the coefficients used for the contact model even under large time steps. Learning-based approaches have also been applied to fluids [94], cloth [95], and robotic manipulation [96].

Hu et al. [57] base their differentiable simulation on the material point method and target specific applications in soft robotics for which explicit time integration is sufficient. Degrave et al. [97] rely on auto-differentiation to compute derivatives of rigid-body dynamics. Similarly, for end-to-end learning of controllers, recording and reversing simulations for back-propagation, DiffTaichi [98] also relies on auto-differentiation. While auto-differentiation can work well in some settings, the stiff problems we are considering demand fully implicit integration schemes that must run to convergence. Consequently, our non-linear solver requires a variable number of iterations in the outer loop, accurate and efficient sparse linear solvers in the inner loop, and line-search mechanisms to enforce monotonicity. In this setting, it is very unclear how to apply auto-differentiation, as it would need to act on the precise sequence of numerical operations performed during the solve. Our approach, on the other hand, analytically computes derivatives via sensitivity analysis, similar to recent work in graphics [99–101], which describes differentiable simulators for visco-elastic materials and constrained flexible multi-body dynamics. However, as a significant departure from prior work, frictional contact is at the core of our paper.

4.3 DIFFERENTIABLE MULTI-BODY DYNAMICS: PRELIMINARIES

We begin by deriving the mathematical formulation underlying our analytically differentiable simulation model. While this derivation follows general concepts recently described in the literature [100, 102], we note that our quest for a unified simulation framework for multi-body systems composed of arbitrary arrangements of rigid and soft objects demands a somewhat different methodology, as detailed below.

4.3.1 *Implicit time-stepping for multi-body systems*

We start from a time-continuous dynamics problem described in terms of generalized coordinates $\mathbf{q}(t)$, see also [74]. These generalized coordinates can directly represent the nodal positions of a finite element mesh or the degrees of freedom that describe the position and orientation of a rigid body in space. Without loss of generality, here we consider dynamical systems that are characterized by a set of *time independent* input parameters $\mathbf{p}$, which could represent, for instance, material constants or a fixed sequence of control actions. The dynamics of such a system can be succinctly represented through a differential equation of the form:

$$\mathbf{r}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \mathbf{p}) = \mathbf{0} \quad \forall t, \quad \text{where } \mathbf{r} := \hat{\mathbf{M}}(\mathbf{q}, \mathbf{p})\ddot{\mathbf{q}} - \hat{\mathbf{f}}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{p}). \tag{4.1}$$

For soft bodies, the generalized mass $\hat{\mathbf{M}}$ is identical to the constant FEM mass matrix, whereas for each rigid body, the corresponding 6×6 block is defined as

$$\hat{\mathbf{M}}_{\mathrm{RB}} := \begin{pmatrix} m\mathbf{I} & 0 \\ 0 & \mathbf{I}_q \end{pmatrix}, \tag{4.2}$$

with m and $\mathbf{I}_q$ representing its mass and configuration-dependent moment of inertia tensor in generalized coordinates [103] (see Appendix A.1.3).

The generalized force vector $\hat{\mathbf{f}}$ aggregates all internal and external forces applied to the simulation's degrees of freedom. Sections 4.4 and 4.5 detail the exact mathematical formulation for these forces, but here we note that for each rigid body, forces must be mapped from world to generalized space, and an additional term $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ that captures the effect of fictitious centrifugal and Coriolis forces must also be added.

Turning to a time-discrete setting, we opt to discretize the residual $\mathbf{r}$ using implicit numerical integration schemes. This decision warrants a brief discussion. While implicit integration schemes are standard when it comes to simulation of elastic objects, rigid-body dynamics is typically handled with explicit methods such as symplectic Euler. This misalignment in the choice of integrators often necessitates multi-stage time-stepping schemes for two-way coupling of rigid bodies and elastic objects [104]. One notable exception is "RedMax" [105], which directly couples rigid and deformable objects. However, their system is only efficient for a small number of deformable DOFs, and they employ staggered projections for ground contacts, which is not well suited for differentiable simulation. Although effective, such a specialized time-stepping scheme complicates matters when differentiating simulation results. Furthermore, as discussed in §4.4, the smooth contact model we propose relies on numerically stiff forces arising from a unilateral potential. Robustly resolving contacts in this setting demands the use of implicit integration for both rigid and elastic objects. As an added bonus, two-way coupling between these two classes of objects becomes straightforward and numerically robust even under large simulation time steps.

We begin by approximating the first and second time derivatives of the system's generalized coordinates, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$, according to an implicit time-discretization scheme of our choice. For instance with BDF1 (implicit Euler), they are $\dot{\mathbf{q}}(t_i) \approx \dot{\mathbf{q}}_i = (\mathbf{q}_i - \mathbf{q}_{i-1})/\Delta_t$ and $\ddot{\mathbf{q}}(t_i) \approx \ddot{\mathbf{q}}_i = (\mathbf{q}_i - 2\mathbf{q}_{i-1} + \mathbf{q}_{i-2})/\Delta_t^2$, where Δ_t is a constant finite time step and superscript i refers to the time-step index for time t_i.

For each forward simulation step i we then employ Newton's method to find the end-of-time-step configuration $\mathbf{q}_i$ such that

$$\mathbf{r}_i := \mathbf{r}(\mathbf{q}_i, \dot{\mathbf{q}}_i, \ddot{\mathbf{q}}_i, \mathbf{p}) = \hat{\mathbf{M}}(\mathbf{q}_i, \mathbf{p})\ddot{\mathbf{q}}_i - \hat{\mathbf{f}}(\mathbf{q}_i, \dot{\mathbf{q}}_i, \mathbf{p}) = 0.$$

This process requires the Jacobian $d\mathbf{r}_i/d\mathbf{q}_i$, which combines derivatives of the generalized forces, the discretized generalized velocities and accelerations, as well as the generalized mass matrix w.r.t. $\mathbf{q}_i$. Recall that for rigid bodies the mass matrix is configuration-dependent, see also Appendix A.1.3 for details. We note that in contrast to previous work [100, 102], we cannot treat time-stepping as an energy minimization problem due to this state dependent nature of the mass matrix. A recent alternative formulation for rigid bodies [106] could overcome this limitation. Nevertheless, in conjunction with a line-search routine that monitors the magnitude of the residual, directly finding the root of $\mathbf{r}_i$ works very well in practice.

4.3.2 *Simulation Derivatives*

With the basic procedure for implicit time-stepping in place, we now turn our attention to the task of computing derivatives of simulation outcomes. To keep the exposition brief, we concatenate the generalized coordinates for an entire sequence of time steps computed through forward simulation into a vector $\tilde{\mathbf{q}} := (\mathbf{q}^T, \ldots, \mathbf{q}^{n_t T})^T$, where n_t is the number of time steps. Similarly, we let $\tilde{\mathbf{r}}$ be the vector that concatenates the residuals for all time steps of a simulation run. Differentiating $\tilde{\mathbf{r}}$ with respect to input parameters $\mathbf{p}$ gives:

$$\frac{d\tilde{\mathbf{r}}}{d\mathbf{p}} = \frac{\partial\tilde{\mathbf{r}}}{\partial\mathbf{p}} + \frac{\partial\tilde{\mathbf{r}}}{\partial\tilde{\mathbf{q}}}\frac{d\tilde{\mathbf{q}}}{d\mathbf{p}}. \tag{4.3}$$

Since for *any* $\mathbf{p}$ we compute a motion trajectory such that $\tilde{\mathbf{r}} = \mathbf{0}$ during the forward simulation stage, the total derivative ((4.3)) is also $\mathbf{0}$. This simple observation lets us compute the Jacobian $d\tilde{\mathbf{q}}/d\mathbf{p}$, which is also called the sensitivity matrix, as:

$$\mathbf{S} := \frac{d\tilde{\mathbf{q}}}{d\mathbf{p}} = -\left(\frac{\partial\tilde{\mathbf{r}}}{\partial\tilde{\mathbf{q}}}\right)^{-1}\frac{\partial\tilde{\mathbf{r}}}{\partial\mathbf{p}}. \tag{4.4}$$

The sensitivity matrix **S** captures the way in which an entire trajectory computed through forward simulation changes as the input parameters **p** change. Note that **S** can be computed efficiently by exploiting the sparsity structure of $\partial\tilde{\mathbf{r}}/\partial\tilde{\mathbf{q}}$, a matrix that can be easily evaluated by re-using most of the same ingredients already computed for forward simulation. For example, in the case of BDF1, each sub-block $\mathbf{s}_j^i := d\mathbf{q}_i/d\mathbf{p}_j$ of **S**, which represents the change in the configuration of the dynamical system at time t_i with respect to the j-th input parameter $\mathbf{p}_j$, reduces to:

$$\mathbf{s}_j^i = -\left(\frac{\partial\mathbf{r}_i}{\partial\mathbf{q}^i}\right)^{-1}\left(\frac{\partial\mathbf{r}_i}{\partial\mathbf{p}_j} + \frac{\partial\mathbf{r}_i}{\partial\mathbf{q}_{j-1}}\mathbf{s}_j^{i-1} + \frac{\partial\mathbf{r}_i}{\partial\mathbf{q}_{j-2}}\mathbf{s}_j^{i-2}\right). \qquad (4.5)$$

Note that the dependency on past sensitivities follows the same time-stepping scheme as the forward simulation. We refer the interested reader to [107] for more details on this topic, but for completeness we briefly describe a variant of this formulation, the adjoint method, as it can lead to additional gains in computational efficiency.

4.4 DIFFERENTIABLE FRICTIONAL CONTACT MODEL

With the differentiable simulation framework in place, our challenge now is to formulate a smooth frictional contact model that approaches, in the limit, the discontinuous nature of physical contacts. In this context, we always refer to the smoothness of the resulting trajectory, rather than the contact forces. Specifically, an impulsive force would result in a non-smooth trajectory, which we must avoid. As we show in this Section, a soft-constraint approach to modelling frictional contact fits seamlessly into the theory presented in §4.3, and enables an easy-to-tune trade-off between accuracy (e.g. solutions satisfying contact complementarity constraints and Coulomb's law of friction) and smoother, easier to optimize objective function landscapes.

Our work investigates different friction models from multiple points of view: accuracy of forward simulation results, smoothness of optimization landscapes, and suitability for gradient-based optimization. It is this mission statement that sets our work apart from previous efforts. Our extensive virtual and real-world experiments show that our simulation model can be used for various applications in graphics and robotics.

We begin by formulating the contact conditions and response forces in terms of a single contact point $\mathbf{x}(\mathbf{q})$. For deformable objects, we handle contacts on the nodes of the FEM mesh. For rigid bodies, we implement spherical or point-based collision proxies and then map the resulting world-space contact forces $\mathbf{f}$ into generalized coordinate space $\hat{\mathbf{f}}$, as described in §4.5.

For the normal component of the contact we must find a state of the dynamical system such that

$$g(\mathbf{x}) \geq 0, \tag{4.6}$$

where we refer to g as the *gap* function, measuring the distance from $\mathbf{x}$ to the closest obstacle. Assuming that g is a signed distance function (and negative if $\mathbf{x}$ lies inside of an obstacle), we define the outward unit normal $\mathbf{n} := (\partial g / \partial \mathbf{x})^T$. Note that we use the convention that the derivative of a scalar function w.r.t. a vector argument is a row-vector throughout this work, consequently $\mathbf{n}$ is a column-vector. Furthermore, we assume that g is available in closed form; we mostly use planar obstacles in our examples. We will also use the notation $\mathbf{N} := \mathbf{n}\mathbf{n}^T$ for the matrix projecting to this normal. The force required to maintain non-negative gaps (non-penetration), denoted $\mathbf{f}_n$, must be oriented along $\mathbf{n}$, i.e. $\mathbf{f}_n = f_n \mathbf{n}$. The *normal force magnitude* f_n must be non-negative and can only be non-zero if there is a contact, consequently (Hertz-Signorini-Moreau condition, see also Eq. (2.10) in [108]):

$$f_n \geq 0, \; f_n \, g = 0 \tag{4.7}$$

In the tangential direction, we must determine a friction force $\mathbf{f}_t$ whose magnitude is constrained by the Coulomb limit

$$\|\mathbf{f}_t\| \leq c_f f_n, \tag{4.8}$$

where c_f is the *coefficient of friction*. This inequality distinguishes two regimes: *static* and *dynamic* friction (also referred to as *stick* and *slip* respectively). In the former case, the friction force magnitude is below the Coulomb limit and the tangential velocity vanishes (stick). In the latter case, following the principle of maximum dissipation [109], the friction force is oriented opposite the tangential velocity and Eq. (4.8) is satisfied as an equality (slip).

More formally we have either

$$\mathbf{T}\dot{\mathbf{x}} = 0 \text{ (stick), or} \tag{4.9}$$

$$\mathbf{f}_t = -c_f f_n \mathbf{t} \text{ (slip),} \tag{4.10}$$

where $\mathbf{t} := (\mathbf{T}\dot{\mathbf{x}}) / \, \|\mathbf{T}\dot{\mathbf{x}}\|$ and $\mathbf{T} := \mathbf{I} - \mathbf{N}$ projects to the tangent plane. Note that in either case $\mathbf{f}_t \cdot \mathbf{n} = 0$ must hold.

4.4.1 *Sequential Quadratic Programming*

Here, we briefly outline a hard constrained formulation of frictional contacts, which we employ for comparison. Introducing Lagrange multipliers (representing contact forces) for the contact constraints (4.6) and (4.9), along with the dynamic friction forces, into the residual (4.1) leads to a system of equations representing the Karush-Kuhn-Tucker conditions of an optimization problem:

$$\begin{gathered} \mathbf{r} + \mathbf{n}\lambda + \bar{\mathbf{T}}^T \boldsymbol{\mu} = 0, \\ g(\mathbf{x}) \geq 0, \text{ and} \\ \bar{\mathbf{T}}\dot{\mathbf{x}} = 0, \end{gathered} \tag{4.11}$$

where λ and $\boldsymbol{\mu}$ are the Lagrange multipliers for the normal and static friction constraints respectively. For a single contact point in static friction, ¯ has two components and $\bar{\mathbf{T}}$ contains two orthogonal tangent vectors (corresponding to the two non-zero eigenvalues of $\mathbf{T}$). In the case of dynamic friction (sliding) we remove the constraint on the tangential velocity and replace the friction force $\bar{\mathbf{T}}^T \boldsymbol{\mu}$ with $\mathbf{f}_t$ according to Eq. (4.10).

In principle, the KKT system (4.11) can be linearized and solved as a sequence of quadratic programs (SQP). In this way, each quadratic program takes the role of the linear solve in Newton's method. However, there are some limitations to this approach. In particular, linearizing the dynamic friction force resulting from Eq. (4.10) is not straightforward, as this force depends not only on the simulation state and velocity, but also on the normal force, in this case given by the Lagrange multiplier λ. Furthermore, the direction of the friction force should be opposing the current sliding velocity, which is ill-defined if this velocity approaches zero.

One way to address these issues is to assume that $\mathbf{f}_t$ is constant rather than linear, and compute its value based on the previous iteration. Doing so however reduces the convergence of the SQP iteration. In some cases the ill-conditioned behaviour of the unit vector $\mathbf{t}$ can even cause this approach to not converge at all. To help mitigate this problem, we first approximately solve the problem assuming sticking contacts, and then only update the friction force magnitude according to the Coulomb limit, but maintain its direction. Similarly, the approach of Tan et al. [80] formulates the frictional contact problem as a quadratic program with linear complementarity constraints by linearizing the friction cone. Their solver also iteratively updates a set of active constraints, which effectively selects the direction of the sliding force. Finally, the SQP iteration is not as straightforward to stabilize by a line-search procedure as a Newton iteration would be. Macklin et al. [75] report similar issues and present approaches to improve upon them.

Note that derivatives of the KKT conditions (4.11), once the solver has converged, could still be used to compute simulation state derivatives by direct differentiation. However, it is currently not clear whether this approach would immediately integrate with our *adjoint* formulation. We refer to [110] for further details on derivatives of QP solutions.

In the light of these limitations, we have implemented an SQP forward simulation approach only for the sake of comparisons on simple deformable examples, where stability and convergence is not an issue; see also Fig. 4.1. We show that our hybrid method, Section 4.4.3, yields visually indistinguishable results.

4.4.2 *Penalty methods*

Another approach to solve dynamics with contacts is to convert the constraints (4.6) and (4.9) to *soft* constraints, and introduce penalty forces if the constraints are violated. For the contact constraint, this means we need to add a force that is oriented along the outward normal and vanishes when $g > 0$. There are various types of functions, such as soft-max or truncated log-barriers, that we can use for this purpose. We find that even the simplest choice of a piece-wise *linear penalty function* is fast and sufficiently accurate for our applications:

$$\mathbf{f}_n = \mathbf{n}\, k_n \max(-g,\ 0), \tag{4.12}$$

where k_n is a penalty factor that must be chosen large enough to sufficiently enforce the normal constraint.

Note that k_n effectively controls the steepness of the resulting force and therefore the smoothness of the collision response. Of course the derivative of this force contains a discontinuity due to the max operator. While this could easily be replaced by a soft-max, our experiments indicate that doing so is not necessary. To find itself exactly at the kink in the contact force, a point on the multi-body system would have to be on the $g = 0$ manifold at the end of the time step due only to gravity, inertia, and internal force, which is extremely unlikely. The force Jacobian, which is needed for both forward simulation and to compute derivatives is well-defined everywhere else.

Similarly, we can formulate Coulomb friction as a clamped linear *tangential* penalty force with corresponding penalty factor k_t:

$$\mathbf{f}_t = -\mathbf{t} \min(k_t \left\|\mathbf{T}\dot{\mathbf{x}}\right\|,\ c_f f_n). \tag{4.13}$$

This expression reduces to (4.10) for dynamic friction, but results in the linear penalty force $k_t \mathbf{T}\dot{\mathbf{x}}$ for static friction.

Instead of solving the KKT conditions (4.11), we now only need to solve a formally unconstrained system that has the same form as Eq. (4.1), with additional penalty forces according to (4.12) and (4.13): $\mathbf{r} + \mathbf{f}_t + \mathbf{f}_n = 0$. From now on, we assume that these penalty forces (mapped to generalized coordinates in the case of rigid bodies) are part of the residual itself. Consequently, we can still use a standard Newton method to solve for the dynamic behaviour (after applying an adequate time discretization scheme). Stabilizing the solver with a line-search method that enforces decreasing residuals is sufficient to handle the non-smooth points of the penalty forces introduced by the max and min operators.

We can also smooth out the transition between stick penalty and dynamic friction force and replace $\mathbf{f}_t$ of Eq. (4.13) with

$$\mathbf{f}_t = -\mathbf{t}\, c_f f_n \tanh\left(k_t \left\| \mathbf{T}\dot{\mathbf{x}} \right\| / (c_f f_n) \right). \tag{4.14}$$

Using a hyperbolic tangent function introduces slightly softer friction constraints and can help improve the performance of optimization methods, see Fig. 4.2. Note that for tangential velocities close to zero $\tanh(v) \approx v$, which resolves any numerical issues with computing $\mathbf{t}$ for small velocities.

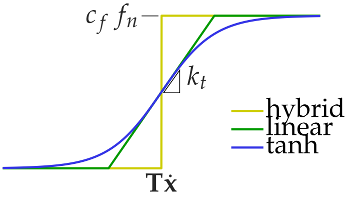
In our examples we always choose $k_t = k_n \Delta_t$, because k_n penalizes a positional error, whereas k_t penalizes a velocity error. Consequently, scaling by a time step admits a similar positional error per time step for both components.

4.4.3 *Hybrid method*

Solving frictional contacts with hard constraints is both computationally expensive in terms of the forward simulation, as well as more challenging to differentiate, especially in an adjoint formulation. Typically, the normal contact constraint is satisfied exactly, but the friction model is simplified to enable more efficient simulation. Even then, the resulting complementarities are technically a combinatorial problem and heuristics are employed to choose the active constraints.

Penalty formulations, on the other hand, are formally unconstrained and therefore relatively straightforward to simulate and differentiate. While the choice of penalty stiffness does affect numerical conditioning, using implicit integration enables stable simulation for a wide range of stiffness. The main drawback of penalties is that one must allow some small constraint violations.

While a small violation of the normal constraint (4.6) is often visually imperceptible, softening the static friction constraint can introduce unacceptable artefacts in some situations. In particular, if a (heavy) object is supposed to *rest* on an inclined surface *under static friction* and the simulation runs for a sufficiently long time, the tangential slipping introduced by softening the stick constraint will inevitably become visually noticeable.



We address this problem by formulating a hybrid method using linear penalty forces, Eq. (4.12) and (4.13), combined with *equality* constraints for the static friction case. In order to take one time step in the forward simulation, we proceed as follows: first, we approximately solve the linear penalty problem to a residual of $\|\mathbf{r}\| < \varepsilon^{1/2}$. Then, we apply hard constraints of the form (4.9) if the friction force according to (4.13) is below the Coulomb limit. Note that these *equality* constraints are analogous to standard Dirichlet boundary conditions. Consequently, we continue with the Newton iteration, including these constraints. However, if enforcing a hard constraint leads to a tangential force exceeding the Coulomb limit, we revert that contact point back to the clamped penalty force (4.13). We only allow this change if the intermediate state satisfies $\|\mathbf{r}\| < \varepsilon^{1/2}$. In rare cases, we eventually fall back to the penalty formulation. These cases typically arise right at the transition point when a sliding object comes to rest; once at rest, the hard constraints keep it in place reliably. Even if we revert to penalties, we know that hard constraints would have violated the Coulomb limit in that time step, so allowing a small sliding velocity is acceptable at this point. Finally, we continue the solver iterations until convergence, i.e. $\|\mathbf{r}\| < \varepsilon$.

In this way, we enforce sticking with hard constraints whenever possible, and guarantee that the Coulomb limit is never violated.

Note that employing a penalty formulation for the *normal* forces means that both, the normal force and the Coulomb limit, depend *directly* on the simulation state (as opposed to representing normal forces as Lagrange multipliers). Consequently, we can easily compute the derivatives of these forces, maintain the quadratic convergence of Newton's method, and stabilize the solver with a line search.

Similarly, when computing sensitivities or adjoint objective function gradients, we can incorporate the additional equality constraints in the

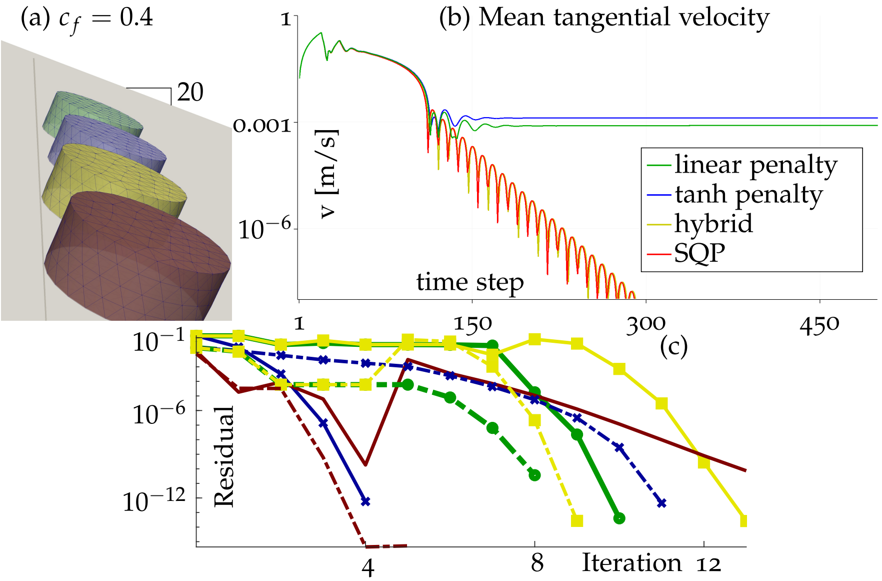
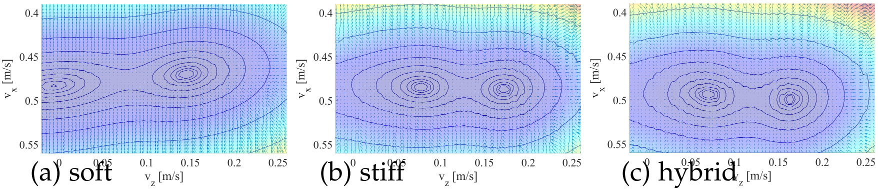


FIGURE 4.1: Dropping a cylinder onto an inclined plane. The resulting motion is generated using different contact and friction methods: linear penalty (green), tanh penalty (blue), hybrid (yellow), SQP (red). The final state of the simulation (a) after 500 time steps ($t = 2.5$ s) is shown on the top left, and the mean tangential velocity over time (b) is shown in the top right sub-figure. Solver convergence for time steps 60 (sliding phase, solid), and 150 (sticking phase, dashed) are shown in sub-figure (c). Note that tangential penalty methods cannot achieve a full stop under static friction. Our hybrid method is very close to the SQP solution.

same way as Dirichlet boundary conditions. That is, the sensitivities or the adjoint state must fulfil analogous boundary conditions to the forward simulation. Specifically, for BDF1, the static friction constraint at time t_i becomes $\bar{\mathbf{T}}_i\mathbf{x}_i = \bar{\mathbf{T}}_i\mathbf{x}_{i-1}$ and consequently, we have $\bar{\mathbf{T}}_i\mathbf{s}_i = \bar{\mathbf{T}}_i\mathbf{s}_{i-1}$ for sensitivities of $\mathbf{x}$, or $\bar{\mathbf{T}}_i\boldsymbol{\lambda}_{i-1} = \bar{\mathbf{T}}_i\boldsymbol{\lambda}_i$ for the adjoint state (note that the earlier state is unknown in the latter case).

4.4.4 *Summary and evaluation*

In this Section we have described various ways of handling contacts with Coulomb friction, focusing our discussion on a single contact point. Now



(a) soft (b) stiff (c) hybrid

FIGURE 4.2: Objective function (shading and isolines) and gradients (arrows) corresponding to the control problem presented in Fig. 4.12a, using soft (a; $k_n = 100$) and stiff (b; $k_n = 10^3$) linear penalty contacts, compared to hybrid contacts (c; $k_n = 10^3$). Note that the overall nature of the objective landscape (i.e. the local minima it exhibits) remains the same, although these minima do shift slightly when softening the contacts. This behaviour promotes the use of continuation methods, whereby solutions obtained with a soft, smoother contact model are used as initial guess for simulations with stiffer, more realistic, parameters.

we briefly summarize how these considerations integrate with our differentiable simulation framework. Recall that for deformable objects the nodes of the FEM mesh directly define the degrees of freedom. In this case, the contact handling extends naturally from a single point to each mesh node independently. For rigid bodies, on the other hand, we handle spheres or point sets defining the collision proxy. The resulting contact forces are then mapped from each contact point to the generalized force vector as described in §4.5.

In Fig. 4.1, we compare the different contact models described above through an experiment where a soft cylindrical puck is dropped onto an inclined plane. We choose a relatively low penalty factor of $k_n = 10^2$ for this example to highlight the differences between soft and hard constrained methods, especially in the static friction phase. The differences in the normal direction are imperceptible, even with a fairly low penalty factor. With soft friction constraints the cylinder slides slightly further (Fig. 4.1a) and does not come to a complete stop (Fig. 4.1b) (though the tangential speed is on the order of 1 mm/s). Also of note is the fact that the SQP solver converges much slower during the sliding phase (Fig. 4.1c), while our hybrid method performs similarly to pure penalty methods in most cases, but delivers equivalent results to the SQP version under static friction. Conversely, tanh penalty forces are highly non-convex around the Coulomb limit, which results in slightly slower convergence during the sticking phase as compared to the linear friction forces. Overall, the linear penalty approach to friction

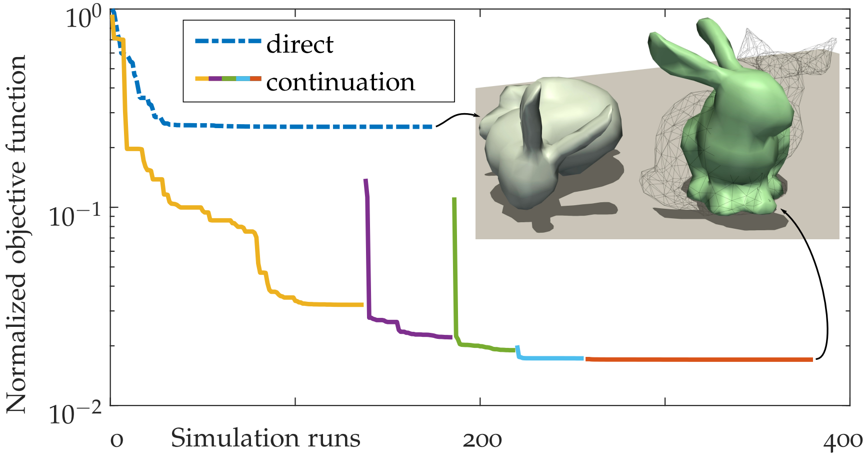


FIGURE 4.3: Optimization convergence for throwing a bunny, similar to Fig. 4.13b but without the wall, such that it lands upright, close to a target pose. A continuation strategy applied to the stiffness of the contact model can drastically improve optimization results for such challenging control problems. Direct optimization uses $k_n = 10^3$; continuation uses k_n: 100 (yellow), 200 (purple), 400 (green), 800 (blue), and 10^3 (red).

forces converges faster than tanh penalties or hybrid contacts, while the SQP method takes about twice as long in terms of total CPU time; see also Table A.1.

In summary, our validation tests show that penalty-based models of frictional contact approach the ground truth defined by complimentary contact constraints and Coulomb's friction law, and they lead to better convergence rates for forward simulation than alternatives based exclusively on hard constraints. In this context, employing an implicit integration method in conjunction with a line-search routine maintains simulation stability at every time step even for very stiff penalties. Furthermore, our treatment of normal and friction forces makes sensitivity analysis (both the direct and adjoint variation) easy to apply to the simulated motion trajectories. We note that this would be much more challenging to achieve if we had to take derivatives of the general KKT conditions of the underlying linear complementarity problem.

To understand how the different contact models affect the types of inverse problems we aim to solve with our differentiable simulator, we perform another experiment. Here we exhaustively sample the objective function on the task of tossing a ball to a specific target location (see also Fig. 4.12a). In particular, we evaluate the objective function value and its gradients on a regular grid in input parameters v_x and v_z (i.e. the initial linear velocity in the forward and upward direction, respectively). Note that the ball is initially spinning with a non-zero angular velocity that is kept fixed for all tosses. Figure 4.2 illustrates these results. The two local minima correspond to one-bounce and two-bounce solutions to this control problem. As can be seen, contacts introduce noise into the objective function (visible as wiggly isolines and somewhat incoherent gradients); this is inevitable as contacts are inherently discontinuous events. Slight changes in the object's initial velocity can lead to a different order in which the nodal degrees of freedom impact the ground. When both the object and the ground are stiff, the noise in the objective function landscape caused by these discretization artefacts can be significant and lead to reduced performance of gradient-based optimization methods. Nevertheless, our experiment shows that the smoothness of the objective function landscape can be effectively controlled through the parameters used by the contact model. This is because smoother contact models enlarge the window of time over which contacts are resolved, and they avoid the use of large impulsive forces. Sensitivities with respect to the exact timing and order of collisions are therefore reduced. This observation, which is supported by the objective function landscapes visualized in Fig. 4.2, can be exploited to improve convergence rates for the inverse problems that leverage our differentiable simulator.

For the example in Fig. 4.3, we evaluate a simple continuation approach. This time around, the task is to throw a geometrically complex object (a bunny) such that it lands upright in a particular location. When the optimization problem is solved using a stiff contact model, an unfavourable local minimum is quickly reached. In contrast, if the optimization problem starts out with a soft contact model which gets progressively stiffer over time, a much better solution is found.

Based on the experimental results described above, we conclude that the linear friction force model offers a favourable trade-off between simplicity, accuracy and practical performance, and as such it is our default choice for the results we present in this Chapter.

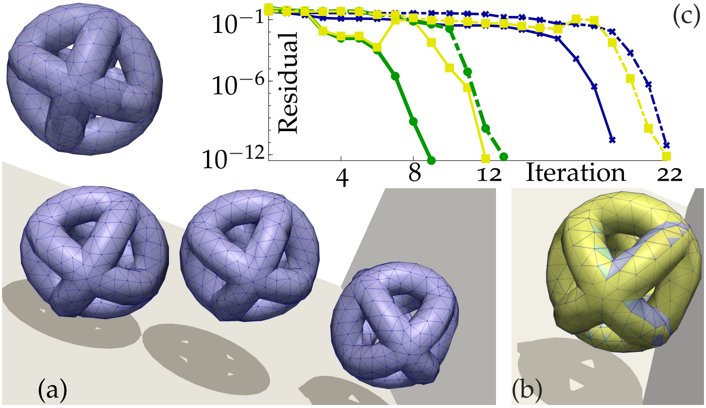


FIGURE 4.4: Dropping an object composed of three tori. Selected frames from the animation (a), and overlay of the wall contact configuration (b) simulated with different contact methods: linear penalty (green), tanh penalty (blue), hybrid (yellow). Solver convergence (c) for the shown time steps with ground contact (solid), and wall contact (dashed).

4.5 INTERNAL AND EXTERNAL FORCES IN GENERALIZED COORDINATES

In this Section, we describe the models used to generate the forces acting on the multi-body systems simulated within our framework. We also present basic validation tests of our forward simulation.

4.5.1 *Soft bodies*

For deformable elastic objects, we employ a standard Neo-Hookean material model, given by homogeneous Lamé parameters (μ, λ) and constant mass density ρ. As is standard, this material model describes the energy density as a function of the deformation gradient $\mathbf{F}$. Internal shape-restoring forces that arise in response to induced deformations are then computed as the negative gradient of the energy density integrated over each element with respect to the nodal degrees of freedom.

To model the behaviour of real-world objects, the elastic forces described above must be complemented by internal damping forces. Most viscosity models, such as the ones described in [100], define the viscous stress (and consequently the damping force) based on the linear strain rate $(\dot{\mathbf{F}} + \dot{\mathbf{F}}^T)$.

One major drawback of these models is that they are not invariant to rotational motion, and therefore damp out the angular velocity of a deformable object during free flight.

Brown et al. [111], on the other hand, describe a family of rotation invariant viscosity models. Here, we employ a quadratic model, similar to their power-law damping, defining the viscous stress as a function of the Green strain rate **D**. In particular, we define the viscous stress as where **F** is the deformation gradient, $\dot{\mathbf{F}}$ is the velocity gradient, and ν is the material's viscosity. We compute this derivative, as well as the corresponding damping matrix entries, using symbolic differentiation. As this viscosity model is based on a quadratic strain rate, it behaves like a power-law model with flow index $h = 2$.

Figure 4.4 shows a deformable object composed of three tori in a drop test. The ground is inclined by 20, while the friction coefficients of the ground and the wall are 0.4 and 0.8 respectively. In this example, we use BDF2 time integration. The rotation invariant viscosity model allows the object to rotate freely in the absence of contacts, but damps out elastic oscillations. Our three contact methods converge reliably to very low residuals; Fig. 4.4c shows convergence for two representative time steps during ground and wall contact respectively. Again, the linear penalty method is fastest, while tanh and hybrid contacts are closely matched. These tests further confirm our conclusion that penalty methods are sufficiently accurate when using implicit integration, which allows a high penalty stiffness.

4.5.2 *Rigid bodies*

Cartesian-space forces **f** and torques $\boldsymbol{\tau}$ applied to a rigid body project to generalized coordinates via the standard transformation

$$\hat{\mathbf{f}} = \begin{pmatrix} \mathbf{I} & 0 \\ 0 & \mathbf{J}_l^T \end{pmatrix} \begin{pmatrix} \mathbf{f} \\ [\hat{\mathbf{x}}]\mathbf{f} + \boldsymbol{\tau} \end{pmatrix},$$

where the Jacobian $\mathbf{J}_l$ maps the rate of change of a rigid body's rotational degrees of freedom to changes in its world-space angular velocity. We use this expression, for example, to apply the contact and friction forces computed in §4.4 to rigid bodies that are in contact with the environment. Note that this operation demands the computation of the world coordinates of a contact point, $\mathbf{x}(\mathbf{q})$, as well as its time derivative $\dot{\mathbf{x}}(\mathbf{q}) = (\partial\mathbf{x}/\partial\mathbf{q})\dot{\mathbf{q}}$. We parameterize rotations with exponential coordinates $\boldsymbol{\theta}$ and compute derivatives as in Gallego and Yezzi [112]. A more detailed description of

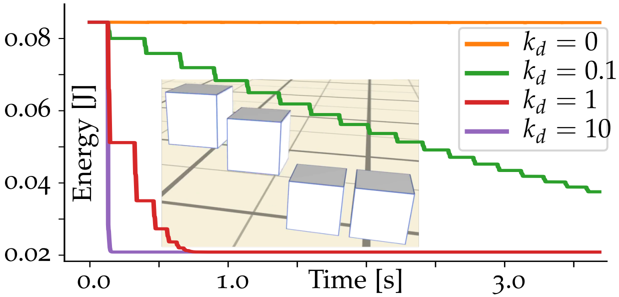


FIGURE 4.5: Total energy over time for bouncing rigid cubes (BDF2) with various ground contact damping coefficients (increasing left-to-right in the inset image).

how we use exponential coordinates and how we avoid the singularity at $|\theta| = 2\pi$ can be found in Chapter 4.6. Force Jacobians, which are needed for both forward simulation and sensitivity analysis, can be easily computed analytically by using the chain rule in conjunction with the derivatives presented in our supplements.

One important concept that must still be modelled is the restitution behaviour of rigid-body contacts. While post-impact velocities for deformable objects are governed by the material's elastic parameters and internal viscosity, for rigid bodies we must explicitly include a damping force in the normal direction in the event of a contact:

$$\mathbf{f}_d = -k_d \mathbf{N}\dot{\mathbf{x}} \quad \text{if } g(\mathbf{x}) \leq 0, \quad \mathbf{f}_d = 0 \text{ otherwise},$$

where k_d is the damping coefficient. For our implicit soft contacts, this contact damping model replaces the common Moreau impact law used in explicit rigid-body engines to model restitution behaviour. In the absence of external forces, the contact phase for a single one-dimensional point mass x against a wall at $x = 0$ can be described as a damped harmonic oscillator $m\ddot{x} + k_d\dot{x} + k_n x = 0$. Analysing the exact solution for this oscillator, with initial conditions $x_0 = 0$ and $\dot{x}_0 = -v_{\text{in}}$, we find the following relation between the damping coefficient and the restitution ratio:

$$\frac{v_{\text{out}}}{v_{\text{in}}} = \exp\left(\frac{-\pi k_d}{\sqrt{4k_n m - k_d^2}}\right),$$

where v_{out} is the outgoing velocity measured after the first half-period of oscillation. Restitution occurs only below the critical damping factor,

$k_d^2 < 4k_n m$. This relation can be useful to either determine the restitution coefficient having estimated k_d using our system, or to set k_d manually for a desired restitution behaviour.

Figure 4.5 shows a basic test case for our fully implicit rigid-body system using BDF2 time integration. Without additional damping, numerical damping is barely noticeable when time-stepping at $\Delta_t = 1/60$ s. This corresponds to almost perfectly elastic collisions. We can effectively control the restitution via our linear contact damping model. Note that the symmetry of the contact is maintained over many bounces. In our video [113], we also show that rotating the cube slightly to the left quickly breaks this symmetry for comparison.

4.5.3 *Signed distance functions as collision objects*

Another problem is the generation of the contact point itself, which is usually one of the most complex parts of a physics engine. For simple geometries, such as planes or spheres. However, for arbitrarily complex geometries, computing contact points can be computationally challenging. A common approach is to use *explicit* representations of the collision geometries, such as triangle meshes, and compute contact points and normals geometrically. Finding the contact points of intersecting triangle meshes usually involves clipping polygons or traversing graph structures, which are operations not well suited for differentiation.

If a collision occurs, our contact model requires the distance to an obstacle, referred to as the gap function $g(\mathbf{x})$, the normal $\mathbf{n}(\mathbf{x})$, and the curvature $\boldsymbol{\kappa}(\mathbf{x})$, which is used to compute derivatives of the contact model. A promising approach is to resort to *implicit* representations of the collision geometries, namely signed distance functions

$$s(\mathbf{x}) = \begin{cases} g(\mathbf{x}, \partial\Omega) & \text{if } \mathbf{x} \in \Omega \\ -g(\mathbf{x}, \partial\Omega) & \text{if } \mathbf{x} \in \Omega^c \end{cases} \tag{4.15}$$

where Ω, Ω^c, and $\partial\Omega$ denote the interior, exterior, and boundary of the object, respectively. The first and second derivatives of $s(\mathbf{x})$ are the normal $\mathbf{n}(\mathbf{x}) = -\nabla_{\mathbf{x}} s(\mathbf{x})$ and the curvature $\boldsymbol{\kappa}(\mathbf{x}) = -\nabla_{\mathbf{x}}^2 s(\mathbf{x})$. This makes signed distance functions a well-suited level of abstraction to combine our contact model with more complex geometries, and gives us several options for how the signed distance field is created.

The first way is to manually find an analytic expression for the signed distance function. For example, a sphere can be described as $s(\mathbf{x}) = ||\mathbf{c} - \mathbf{x}||_2^2 - r$, where $\mathbf{c}$ and r are the center and the radius. Distance functions for primitive shapes can be composed, smoothed, and modified in many ways, as shown in [114].

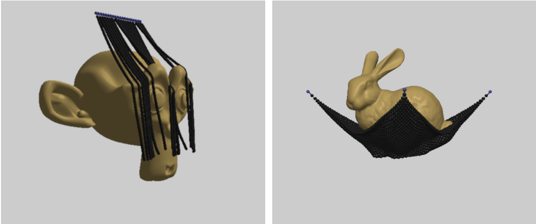


FIGURE 4.6: Forward simulation for different collision types

Second, we can compute signed distance fields using distance samples at discrete points in space. To this end, we can leverage *VDB*, an efficient sparse datastructure, specifically developed for processing volumetric data in computer graphics applications [115]. Similar to a B+-Tree it hierarchically partitions the space into smaller and smaller volumes until a certain leaf volume size is reached. The leaf cubes store the distance values and represent voxels in 3D space. We use an OpenSource version, OpenVDB, of the VDB architecture and make use of its ability to create narrow-band signed distance fields around the zero levelset. Smoothing of the distance field can naturally be applied with a gaussian filter of variable kernel size. This allows an easy way to smooth out high frequencies in the distance field. Figure 4.6 shows strings of point masses and cloth colliding with collision objects represented using VDB.

Finally, neural networks can be used to represent signed distance fields, which has recently attracted much interest in the machine learning community. In recent work, auto-encoder-decoder frameworks are used to learn classes of shapes [116, 117]. In *IGR* [118], signed distance functions are directly learned from raw point cloud data with multilayer perceptrons. Another approach is employed in *Siren* [119], where periodic activation functions are used to solve constraint level set equations. Our initial tests with *Siren* and *IGR* are very promising and we are eager to present more details and results in the near future.

4.5.4 *Multi-body systems*

As we use implicit integration schemes for time-stepping, we employ (stiff) generalized springs and damping to couple the individual constituents of a multi-body system to each other. This is a simple, general, and drift-free technique that can be shown to be closely related to Baumgarte-stabilized velocity-level constraints for rigid-body dynamics [120]. In general, for non-dissipative coupling elements, we define a potential energy as a function of points or vectors anchored on the multi-body system. Taking the gradient of this potential energy with respect to the system's generalized coordinates directly outputs the resulting generalized forces. More formally, constraints $\mathbf{c}(\mathbf{q})$ are enforced through potentials of the form $E(\mathbf{q}) = (k_c/2)\,\mathbf{c}(\mathbf{q})^T\mathbf{c}(\mathbf{q})$.

For instance, the constraint

$$\mathbf{c}_s := \|\mathbf{x_1}(\mathbf{q}) - \mathbf{x_2}(\mathbf{q})\| - l_0 = 0,$$

asks that a specific distance is maintained between two points on the multi-body system. Its resulting potential models a stiff linear spring of rest length l_0. We use zero-length springs to formulate ball-and-socket joints. Furthermore, unilateral springs (of non-zero length), which do not produce a force under compression, model cables and elastic strings (similar to Bern et al. [121]).

Hinge joints (i.e. 1-DOF revolute joints) connecting two rigid bodies are defined through attachment points $(\hat{\mathbf{x}}_1,\ \hat{\mathbf{x}}_2)$ and rotation axes $(\hat{\mathbf{a}}_1,\ \hat{\mathbf{a}}_2)$, specified in the local coordinate frame of each rigid body respectively. We model hinge joints with two constraints: a zero-length spring connecting the attachment points $(\hat{\mathbf{x}}_1,\ \hat{\mathbf{x}}_2)$, and a second constraint that aligns the rotation axes $(\hat{\mathbf{a}}_1,\ \hat{\mathbf{a}}_2)$:

$$\mathbf{c}_h := \mathbf{w}(\hat{\mathbf{a}}_2) - \mathbf{w}(\hat{\mathbf{a}}_1) = 0,$$

where $\mathbf{w}(\hat{\mathbf{a}})$ denotes the mapping from local to world coordinates.

We model active motors by extending hinge joints with a second set of local axes $(\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2)$ that are orthogonal to the rotation axes $\hat{\mathbf{a}}_1$ and $\hat{\mathbf{a}}_2$ respectively. A motor constraint, $\mathbf{c}_m$, enforces a specific relative angle α between $\mathbf{w}(\hat{\mathbf{b}}_1)$ and $\mathbf{w}(\hat{\mathbf{b}}_2)$:

$$\mathbf{c}_m := \mathbf{w}(\hat{\mathbf{b}}_1) - \mathbf{R}(\alpha)\mathbf{w}(\hat{\mathbf{b}}_2).$$

To model position-controlled motors that are driven by typical Proportional-Derivative controllers, it is also important to add a damping component to the torques these motors generate. To this end we directly define a world-space torque as a function of end-of-time-step angular velocities:

$$\boldsymbol{\tau}_{md} := k_{md}(\boldsymbol{\omega}_1(\mathbf{q}, \dot{\mathbf{q}}) - \boldsymbol{\omega}_2(\mathbf{q}, \dot{\mathbf{q}})),$$

which we then project into generalized coordinates using (4.5.2).

4.6 PARAMETERIZATION OF RIGID BODY ROTATION

One of the challenges in simulating the dynamics of rigid bodies in three dimensional space is representing the rotation and angular velocity of rigid bodies. Parameterizations of the rotation matrix either suffer from singularities or impose additional constraints arising from the identities of the rotation matrix. In both cases, a reparameterization of the rotational degrees of freedom is required: either to avoid a singularity in the parameter space, or to project back to the constraint manifold.

We choose exponential coordinates as parameterizations of rotation matrices. Unlike unit quaternions, they do not require normalization after a simulation step. However, they suffer from a singularity when their magnitude approaches 2π, which requires an occasional reparameterization. In this Section, we propose a reparametrization strategy for exponential coordinates that preserves the angular velocity. We also show how this reparametrization fits seamlessly into a differentiable simulator and how the sensitivities can be computed across a reparameterization step.

4.6.1 *Exponential Coordinates and Angular Velocity*

We now introduce exponential coordinates as our choice of parametrization of rotations. While this is in no way a complete description, we hope the reader will gain some intution on how exponential coordinates fit into our framework. We refer the reader to [122] for a more detailed exposition of the exponential map, $SO(3)$-group and associated Lie algebra.

Rotation can be viewed as a transformation of a vector $\mathbf{a}$ with $\hat{\mathbf{a}} = \mathbf{R}\mathbf{a}$, where the identities of a rotation matrix must hold. These are the orthogonality identity $\mathbf{R}^T\mathbf{R} = \mathbf{I}$ and the identity $\det(\mathbf{R}) = 1$ which preserves the right-handedness of the coordinate system. One can think of the elements of a rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ as inhabiting a 9-dimensional space. Not all matrices $\mathbf{R} \in \mathbb{R}^{3\ \times3}$ are valid rotation matrices. The rotation identities are constraints on the elements of $\mathbf{R}$ that form a 3-dimensional manifold in 9-dimensional space. A parametrization of the rotation matrix maps from the parameter space to the manifold formed by the identity constraints in 9-dimensional space.

We can derive exponential coordinates and angular velocity by taking the rate of change of the orthogonality terms $\mathbf{R}^T\dot{\mathbf{R}} + \dot{\mathbf{R}}^T\mathbf{R} = 0$. By rearranging the terms we obtain

$$\mathbf{R}^T\dot{\mathbf{R}} = -\dot{\mathbf{R}}^T\mathbf{R} = -(\mathbf{R}^T\dot{\mathbf{R}})^T = [\boldsymbol{\omega}]_\times,$$

we note that the above matrix is skew-symmetric, denoted $[\dots]_x$, and define $\boldsymbol{\omega}$ to be the corresponding vector. Note that $\dot{\mathbf{R}} = [\boldsymbol{\omega}]$ at $\mathbf{R} = \mathbf{I}$, and we therefore call it the *angular velocity*. We can now solve the ordinary differential equation $\dot{\mathbf{R}} = \mathbf{R}[\boldsymbol{\omega}]_x$ and find $\mathbf{R}(t) = \mathbf{R}_0 \exp([\boldsymbol{\omega}]_\times t)$. At the origin $\mathbf{R}_0 = \mathbf{I}$, *exponential coordinates* are then defined as the vector $\boldsymbol{\theta} = \mathbf{u}\theta = \boldsymbol{\omega}t$ integrating the rotation in terms of an angular axis-axis representation, with angle θ and unit axis $\mathbf{u}$. A closed expression for the exponential mapping $\mathbf{R}(\boldsymbol{\theta}) = \exp([\boldsymbol{\theta}]_\times) = \sum_i [\boldsymbol{\theta}]^i_\times / i!$ is given by the Rodrigues rotation formula

$$\mathbf{R}(\boldsymbol{\theta}) = \mathbf{I} + \sin(\theta)[\mathbf{u}]_x + \cos(\theta)[\mathbf{u}]^2_x.$$

Another useful relation is that of the time derivative of the rotation parameters and the angular velocity

$$\boldsymbol{\omega}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \mathbf{J}_l(\boldsymbol{\theta})\dot{\boldsymbol{\theta}},$$

where $\mathbf{J}_l(\boldsymbol{\theta})$ is the left Jacobian, see Appendix A.1.1.1.

4.6.2 *Reparameterization*

Unlike unit quaternions, exponential coordinates introduce no additional constraints. Numerical integration is therefore very straightforward, since we don't have to worry about projecting back onto the manifold of valid rotation matrices. However, exponential coordinates suffer from a singularity at $\theta = 2n\pi$, where n is an integer.

The common solution is to reparameterize to keep θ within the half-sphere $\theta \leq \pi$. In what follows, the symbols marked $'$ denote the quantities after the reparametrization step. If $\theta \geq \pi$, we compute a new set of parameters $\boldsymbol{\theta}$ such that $\mathbf{R}(\theta) = \mathbf{R}(\theta')$ and $\theta' < \pi$ with

$$\boldsymbol{\theta}' = (1 - \frac{2\pi}{\theta})\boldsymbol{\theta}. \tag{4.16}$$

We note that since $\theta \approx \pi$, $\mathbf{u}' = -\mathbf{u}$ and $\theta' = 2\pi - \theta$.

When integrating in time, the integration scheme computes the time derivatives using the states of the previous time steps, e.g., for BDF1 $\dot{\boldsymbol{\theta}} = \frac{1}{h}(\boldsymbol{\theta} - \boldsymbol{\theta}_{-1})$ with $\boldsymbol{\theta}_{-1}$ being the state in the previous time step. Changing $\boldsymbol{\theta}$ to $\boldsymbol{\theta}'$ necessarily requires adjusting all past states used in the integration scheme. In doing so, we attempt to find a new $\dot{\boldsymbol{\theta}}'$ that preserves the angular velocity

$$\boldsymbol{\omega}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \boldsymbol{\omega}(\boldsymbol{\theta}', \dot{\boldsymbol{\theta}}') \Leftrightarrow \dot{\boldsymbol{\theta}} = J_l^{-1}(\boldsymbol{\theta}')J_l(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}. \tag{4.17}$$

We can now substitute the expressions for J_l and J_l^{-1}, and simplify the expression (see Appendix A.1.1.3) to get the formula

$$\dot{\boldsymbol{\theta}}' = \left(\mathbf{I} + [\mathbf{u}]_\times^2 \frac{2\pi}{\theta}\right) \dot{\boldsymbol{\theta}}. \tag{4.18}$$

We find an easier way to the same result if we take the time derivative of the reparameterization formula (4.16):

$$\dot{\boldsymbol{\theta}}' = \frac{d}{d\boldsymbol{\theta}} \left((1 - \frac{2\pi}{\theta})\boldsymbol{\theta}\right) \dot{\boldsymbol{\theta}} \tag{4.19}$$

$$= \frac{d}{d\boldsymbol{\theta}} (\boldsymbol{\theta} - 2\pi\mathbf{u})\, \dot{\boldsymbol{\theta}} \tag{4.20}$$

$$= \left(\mathbf{I} + \frac{2\pi}{\theta}[\mathbf{u}]_\times^2\right) \dot{\boldsymbol{\theta}} \tag{4.21}$$

where in the last step we used $\frac{\partial \mathbf{u}}{\partial \boldsymbol{\theta}} = -\frac{1}{\theta}[\mathbf{u}]_\times^2$ from [112], Appendix B.

We now have formulas (4.16) and (4.18) to compute new parametrizations $\boldsymbol{\theta}'$, $\dot{\boldsymbol{\theta}}'$ of the rotation matrix and angular velocity, which can then be used to adjust past states $\boldsymbol{\theta}'_{-1}, \boldsymbol{\theta}'_{-2}$, ... depending on the integration scheme. When a reparameterization occurs in the forward simulation, it needs to be accounted for in the backwards pass. For this, we refer the reader to the Appendix A.1.1.2 for the necessary derivatives of equations (4.16) and (4.18).

4.7 SOLVING INVERSE PROBLEMS

We now turn our attention to *optimization problems* based on dynamical systems. The original motivation for developing our differentiable simulator was to consider the full dynamics of a legged robot when planning its locomotion. This allows us to more accurately predict the motion trajectories and take advantage of compliance and deformable components in the motion planning process. In this Section, we demonstrate the effectiveness of our differentiable physics model for trajectory optimization of a compliant robot with soft feet. We also present results on several other important robot applications: Estimation of material parameters including contacts, optimization of initial conditions, machine learning with our differentiable simulator directly integrated with the loss function, and trajectory optimization for manipulation. Before devoting ourselves to these exciting applications, we give an overview of how we formulate these problems in an optimization framework.

4.7.1 *Optimization Framework*

The robotics problems we are interested in can all be represented as optimization problems where we want to find parameters $\mathbf{p}^*$ that minimize an objective O defined as a function of the simulation result:

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} O\,(\,\mathbf{p}, \tilde{\mathbf{q}}(\mathbf{p})\,)\,, \tag{4.22}$$

where $\tilde{\mathbf{q}}(\mathbf{p})$ is the entire trajectory of the dynamical system.

In order to solve this optimization problem using gradient-based methods such as ADAM [123, 124] or L-BFGS [125, 126], we need to compute the derivative

$$\frac{dO}{d\mathbf{p}} = \frac{\partial O}{\partial \mathbf{p}} + \frac{\partial O}{\partial \tilde{\mathbf{q}}}\mathbf{S}. \tag{4.23}$$

In our framework, the parameters $\mathbf{p}$ can be initial conditions, material parameters or control variables. Depending on the objective function, we can then solve a variety of interesting problems. For example, if the parameters are the initial throwing velocity of a ball and the objective is the distance between a target location and the position of the ball at the end of the simulation, we perform trajectory optimization. If the parameters are the material properties of a deformable ball and the objective is the distance between a motion-captured reference trajectory and the simulated trajectory, we perform parameter estimation, also called real2sim. The choice of parameters and objective defines the problem we solve while staying within the same optimization framework. A differentiable simulation model that captures a wide range of physical effects thus opens up the possibility of solving many robotics problems using gradient-based techniques.

4.7.1.1 *The adjoint method*

For some applications, it is not necessary to explicitly compute the sensitivity matrix. Instead, we are interested in finding the input parameters $\mathbf{p}^*$ directly. The adjoint method allows us to compute the objective function gradient (4.23) without directly computing $\mathbf{S}$. While Bradley [127] derives the adjoint method for the continuous-time case, here we describe it directly for the discretized-time setting.

We first express the sensitivity matrix as $\mathbf{S} = -\tilde{\mathbf{B}}^{-1}\tilde{\mathbf{A}}$, where $\tilde{\mathbf{A}} := \partial\tilde{\mathbf{r}}/\partial\mathbf{p}$ and $\tilde{\mathbf{B}} := \partial\tilde{\mathbf{r}}/\partial\tilde{\mathbf{q}}$, respectively. Introducing the notation $\tilde{\mathbf{y}} := \partial O/\partial\tilde{\mathbf{q}}$, Eq. (4.23) becomes

$$\frac{dO}{d\mathbf{p}} = \frac{\partial O}{\partial\mathbf{p}} - \tilde{\mathbf{y}}\tilde{\mathbf{B}}^{-1}\tilde{\mathbf{A}}. \tag{4.24}$$

Instead of evaluating this expression directly, we first solve for the *adjoint state* $\tilde{\boldsymbol{\lambda}}$:

$$\tilde{\mathbf{B}}^T\tilde{\boldsymbol{\lambda}} = \tilde{\mathbf{y}}^T, \tag{4.25}$$

and then compute the gradient of the objective function as follows:

$$\frac{dO}{d\mathbf{p}} = \frac{\partial O}{\partial\mathbf{p}} - \tilde{\boldsymbol{\lambda}}^T\tilde{\mathbf{A}}. \tag{4.26}$$

Note that both $\tilde{\mathbf{y}}$ and $\tilde{\boldsymbol{\lambda}}$ are vectors of length $|\mathbf{q}|\,n_t$, while the size of $\tilde{\mathbf{A}}$ is$|\mathbf{q}|\,n_t \times |\mathbf{p}|$, where n_t is the number of time steps, $|\mathbf{q}|$ is the number of degrees of freedom, and $|\mathbf{p}|$ is the number of parameters. Moreover, the sparsity of $\tilde{\mathbf{A}}$ (or lack thereof) depends on the time interval and spatial domain in which each parameter affects the simulation. For instance, homogeneous material parameters of deformable objects generally affect the entire simulation and result in dense (parts of) $\tilde{\mathbf{A}}$, whereas control inputs for specific time steps influence only a small subset of the simulation and result in a much sparser structure. Finally, we note that the block-triangle structure of $\tilde{\mathbf{B}}$ can also be leveraged to speed up the computation of the adjoint state. This block-matrix shape highlights when (and where) data must be stored during the forward simulation, regardless of how the system is solved. During adjoint state computation, information propagates backward in time (a common feature of time-dependent adjoint formulations).

4.7.1.2 *Gauss-Newton Hessian Approximation*

The adjoint method makes computing the gradient of the objective more efficient, and in combination with a quasi-Newton method such as L-BFGS, we can improve convergence compared to gradient-descent. We however found, that using the Gauss-Newton approximation proposed in [102] in combination with Newton's method leads to even faster convergence rates.

Computing the full Hessian

$$\frac{d^2 O}{d\mathbf{p}^2} = \frac{d}{d\mathbf{p}} \left(\frac{\partial O}{\partial \mathbf{q}} \mathbf{S} \right) + \frac{d}{d\mathbf{p}} \frac{\partial O}{\partial \mathbf{p}}$$

would require us to compute second-order sensitivites. Zimmermann et.al. [102] suggest to leave away the terms involving second-order sensitivities, which leads to the Gauss-Newton approximation

$$\mathbf{H} = \mathbf{S}^T \frac{\partial^2 O}{\partial \mathbf{q}^2} + 2\mathbf{S}^T \frac{\partial^2 O}{\partial \mathbf{q} \partial \mathbf{p}} + \frac{\partial^2 O}{\partial \mathbf{p}^2}.$$

In comparison to the adjoint method, the Gauss-Newton approximation does require the explicit computation of the sensitivity matrix **S**. In our experiments, we found that the convergence improvements per optimization iteration cost outweighs the additional computational cost required to compute the approximated Hessian.

4.7.2 *Motion Generation for Compliant Robot*

In our first experiment, we optimize the control inputs for a legged robot actuated by compliant motors. We note that this use case is motivated by real-world challenges. Compliance, whether parasitic (e.g., motors that are not infinitely strong) or intentionally built in (e.g., rubber feet designed to absorb shock), is a defining feature of physical robots. However, most existing motion planning and trajectory optimization algorithms are unable to account for this. Our differentiable simulator, on the other hand, allows us to optimize the robot's motion by directly taking into account its whole-body dynamics, including the compliant nature of its actuators and feet.

For this experiment, we model the robot's actuators as relatively low-gain PD controllers, a reasonable model for position-controlled motors implemented as soft, damped angular constraints between the coordinate frames of adjacent rigid links. We use the motion synthesis tool presented in Chapter 2 to create a nominal motion trajectory for this robot. However, the physical model based centroidal dynamics is a relatively coarse approximation of the robot's dynamics and assumes precise and strong actuators. Unsurprisingly, due to these model simplifications, our compliant

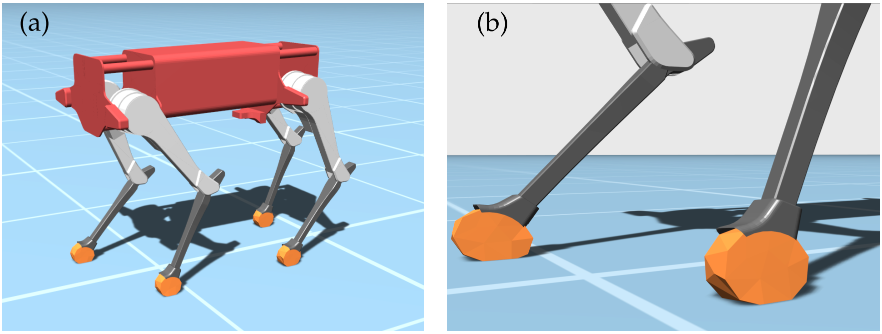


FIGURE 4.7: Coupling soft and rigid bodies allows us to equip this compliant robot with soft feet. We can perform trajectory optimization on the entire robot to find control inputs that account for the compliant motors, as well as the deformable end-effectors.

robot is unable to move effectively when attempting to follow the nominal trajectory.

To recover the target walking speed, we perform trajectory optimization using our differentiable simulation to account for the full dynamics as well as the stiffness and compliance of the actuators. We optimize for control trajectories $\mathbf{p} = [\mathbf{p}_1^T, ..., \mathbf{p}_n^T]^T$, where $\mathbf{p}_i$ represents the target motor angles at time i and n the number of time steps. To generate stable motions, we set the time horizon to $5n\Delta t$, which includes five motion cycles. The same control trajectory is applied for each motion cycle. The initial guess for the control trajectories is set to the motor angle trajectories generated by the motion synthesis tool presented in Chapter 2, and the goal is to achieve the target walking speed. Using our differentiable simulator, we apply trajectory optimization to the fully coupled multibody dynamics and achieve optimized control so that the robot reaches the target walking speed. The result, as shown in Fig. 4.8, is a successful locomotion gait for this robot with compliant actuators and soft feet. Details of the simulation parameters and runtime can be found in Table A.2.

4.7.3 *Material parameter estimation*

To solve problems in robotics, we must be able to accurately predict the dynamics of the physical world. A critical component to this success is parameter estimation, where we attempt to fit the material properties of

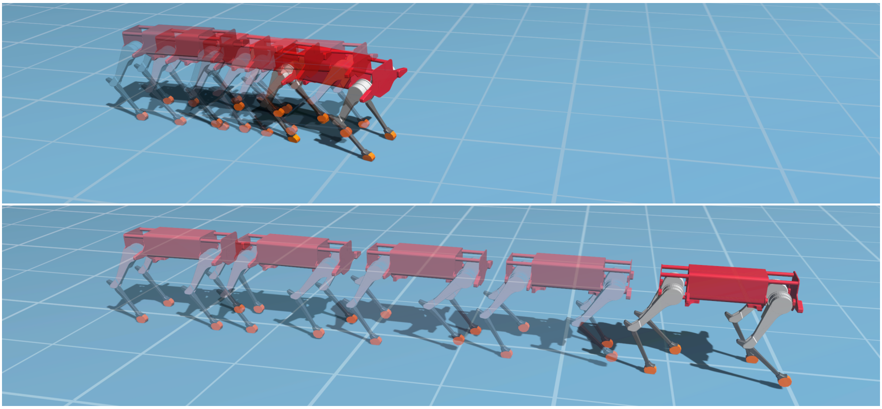


FIGURE 4.8: Control trajectories executed on a robot with compliant motors and soft feet. Top: The control trajectories generated with a simplified dynamics model assuming very stiff motors (chapter 2) fail to achieve the target walking speed. Bottom: Using our multibody differentiable simulation model, we model the compliance of this robot and generate new control trajectories such that the robot reaches the target walking speed.

our simulation model so that the simulation results match the real world. In this Section, we demonstrate this approach by estimating the material parameters of a deformable ball.

Our system allows us to estimate material parameters such as stiffness and damping of deformable objects. We capture the behaviour of our samples in the real world using either an optical motion capture system or an *Kinect v2* depth camera. In the former case, we track up to six labelled optical markers on the sample at a frame rate of 120 Hz using an array of 10 *OptiTrack Prime 13* cameras. System calibration ensures that the world-space coordinates match the ground and wall planes so that these rigid obstacles can be included in the simulation. For motion capture data, the objective function measures the sum of squared distances between the tracked marker position and the corresponding position on the simulated mesh for all time steps. In each time step, we only consider markers that are currently visible to the tracking system are considered. In the latter case, we read 3D point clouds from the *Kinect* at 30 Hz and then identify the ground and wall planes in a manual post-processing step. We apply box filters in both world and colour space to identify the points that correspond to the surface of the object. In this case, since there is no direct correspondence between the tracked points and the mesh positions, the objective function instead measures the distances of all filtered points to their nearest point on the surface of the simulated mesh.

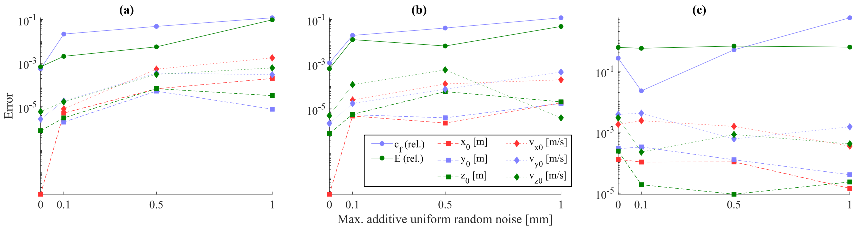


FIGURE 4.9: Errors obtained when optimizing for ground truth initial conditions and material parameters on synthetic data for linear penalty contacts (a), tanh penalties (b), and hybrid contacts (c). We show relative errors for Young's modulus E and the coefficient of friction c_f, and absolute errors for initial conditions (position and velocity).

For parameter estimation, we use a temporal continuation strategy, where we first optimize the initial conditions of the simulation (in terms of position, orientation, velocity, and spin) to match the recorded motion during the ballistic phase (before the first contact). In the second phase, we keep the initial conditions fixed,and optimize the material parameters to most closely match the initial bounce of the recorded motion. Finally, we add a third phase in which all parameters (material and initial conditions) are optimized simultaneously for the entire motion. In our experiments, we found that L-BFGS is well suited for these optimization tasks.

We evaluate the robustness of this approach using synthetic ground truth data containing only a single bounce in Fig. 4.9. Starting from a different initial guess, which is about 10 times softer than the ground truth material, the optimization should recover both the initial conditions and the material parameters while we add uniform random noise to the ground truth data. Both penalty-based approaches find very accurate solutions in the absence of noise and provide good approximations even with higher noise levels. In contrast, the hybrid method, where static friction is enforced by hard constraints, causes the optimization to find unfavourable local minima even in the absence of noise. In particular, the friction coefficient is not correctly recovered in the presence of increased noise. Therefore, we prefer penalty point-based methods for optimization tasks, but also show examples where the hybrid method can be used for optimization.

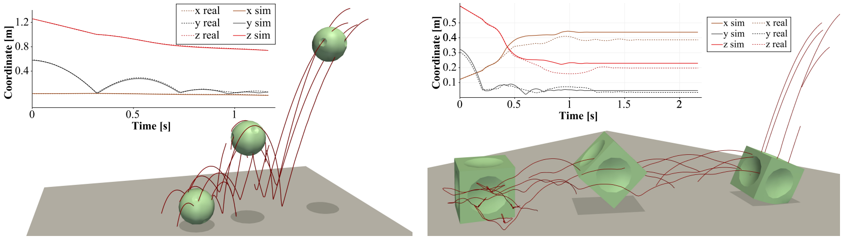


FIGURE 4.10: Parameter estimation for throwing a sphere and a cube. The 3D image shows motion-capture trajectories for all six markers (one in the centre of each face) and snapshots of the best fitting simulation. The inset graph shows captured and simulated trajectories for the front facing marker.

Here, we show three results for material parameter estimation of real-world specimens. We prepare two custom-made elastic foam specimens, a sphere and a cube, for motion capture with six slightly inset motion capture markers each, see Fig. 4.10. The motion capture data provides direct correspondences between the tracked markers and their simulated counterparts, allowing us to find the initial orientation and angular velocity in the first phase. In the first example, Fig. 4.10 left, we then optimize material parameters for the duration of the first bounce, and finally all parameters over the entire recorded trajectory. The second example, Fig. 4.10 right, uses a more automated approach, optimizing all parameters over increasing time horizons. Apart from the material parameter optimization in the first example, which includes a short ADAM phase, we use L-BFGS for all these optimizations. In our video [113] we also show verification tests for both of these results, where we use the material parameters obtained via these optimizations, and then only fit the initial conditions to the ballistic phase of a different recorded motion. Please also refer to Table A.1 for details on material parameters and runtime.

Finally, we show an example for a foam ball without additional markers, where we record the real-world motion with a *Kinect* depth camera, Fig. 4.11. In this case, we minimize the distance from the simulated surface to the recorded point cloud, which means that we do not have any rotational information about the real-world specimen. Nevertheless, by allowing the optimization to change initial conditions during later stages where contacts are taken into account, we find a good match between real and simulated motion. We employ the same approach to estimate parameters of a common

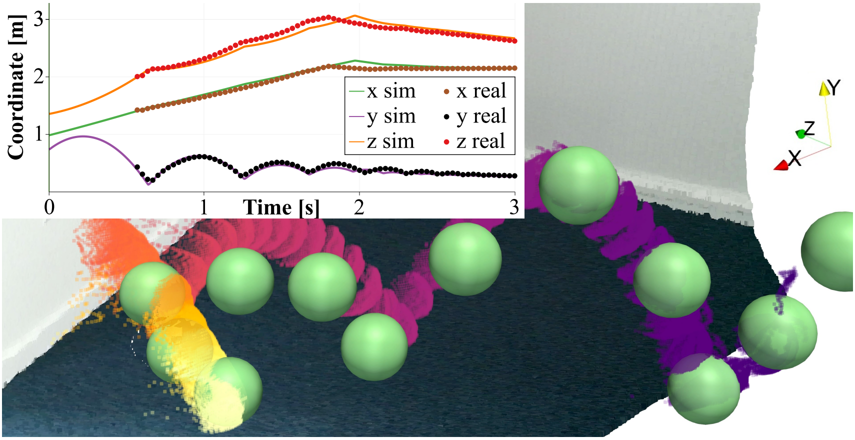


Figure 4.11: Parameter estimation with *Kinect* data. Image shows input point cloud (time colour-coded purple to yellow) and representative time steps of the simulation result (green). Graph shows average coordinates; axes represent the camera orientation (z forward, y up).

tennis ball, which we subsequently throw with a robot as discussed in the next Section.

4.7.4 *Trajectory Optimization*

With optimal parameters found by parameter estimation, we can now use our simulation model to find optimal controls via trajectory optimization. We separate these applications into two groups. First, we find optimal initial conditions to throw objects to hit a target. And second, we find optimal controls for a robot arm which manipulates rigid and soft objects.

4.7.4.1 *Throwing*

Our framework allows us to parameterize, and optimize for, the initial conditions of our simulation, such as in the examples shown in Fig. 4.12. In these cases, we must account for the contribution of the initial conditions to the objective function gradient in Eq. (4.26). While previous work provides an adjoint formulation for general, implicitly defined, initial conditions [127], when directly parameterizing initial conditions we find it more convenient to calculate the corresponding derivatives explicitly.

Parameters that define initial conditions, $\mathbf{p}_0$, are parameter variables that affect only the initialization of the time-integration scheme, but do *not* directly affect any of the *unknown* states $\tilde{\mathbf{q}}$. We can therefore compute the derivatives of the residuals w.r.t. these parameters analytically: $\partial \mathbf{r}_i / \partial \mathbf{p}_0 = (\partial \mathbf{r}_i / \partial \chi)(\partial \chi / \partial \mathbf{p}_0)$, where χ refers to the initial state of the time integrator. The first term follows directly from the choice of time-integration method, while the second term follows from the parameterization of initial conditions. Finally, these derivatives are added to the matrix $\tilde{\mathbf{A}}$ of the sensitivity system, where each block now becomes $\mathbf{A}_i := \partial \mathbf{r}_i / \partial \mathbf{p} + \partial \mathbf{r}_i / \partial \mathbf{p}_0$. Note that only the first few time steps receive a non-zero update, depending on the chosen time-discretization scheme.

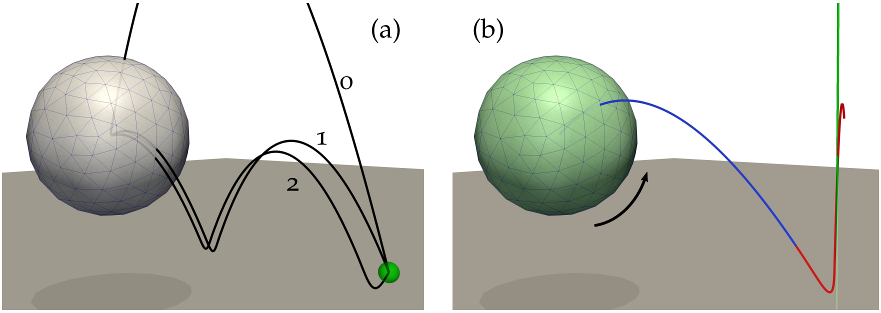


Figure 4.12: Throwing a deformable ball: a point target for the ball's centre of mass (a) admits multiple exact solutions with either zero, one, or two bounces off the floor. Asking the second half of the c.o.m. trajectory (red) to be as close to a vertical line as possible (b) requires a trade-off between forward motion and back-spin (black arrow) such that friction slows the ball down when it bounces off the ground.

In our tests, we optimize initial linear and angular velocities for throwing a deformable ball, Fig. 4.12. The objective function measures the distance from the ball's centre of mass to a specific target point at the end of the simulation (Fig. 4.12a), or to a target line over a specified time range (Fig. 4.12b), respectively.

In order to compare our results to a gradient-free sampling method, we run CMA-ES on the optimization problem in Fig. 4.12a. Qualitatively, gradient-based approaches are less likely to traverse a saddle point, whereas sampling methods explore the parameter space more randomly in the early stages before settling into a local minimum. In this particular case, using L-BFGS and linear penalty contacts requires 28 simulation runs to find an exact solution for the two-bounce motion (relative objective function

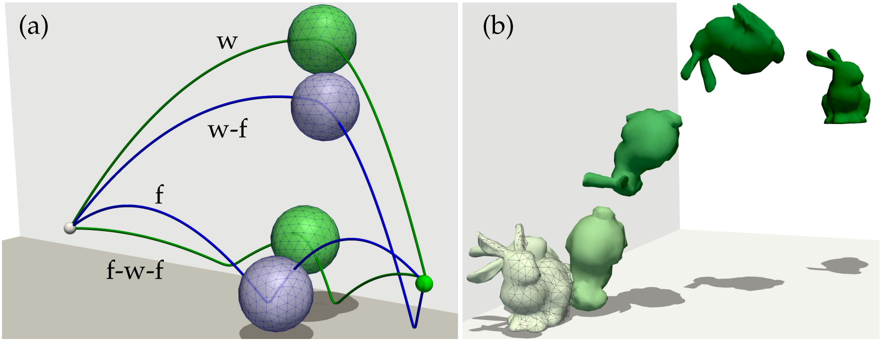


FIGURE 4.13: Throwing with multiple contacts: (a) multiple paths for the ball's c.o.m. to reach the target point, labelled by contact sequence ('w' wall, 'f' floor); (b) throwing the bunny to a specific target pose (wireframe) after bouncing off the floor and the wall (time colour-coded from dark to bright).

value $O/O_0 < 10^{-25}$) in a matter of minutes (Table A.1), while CMA-ES requires 1042 simulations and 2h 15m to find an approximate solution with $O/O_0 \approx 10^{-6}$.

On a more complex example (Fig. 4.13b), L-BFGS finds a better solution ($O/O_0 \approx 4 \cdot 10^{-4}$) after 319 simulations, whereas CMA-ES returns a noticeably worse result ($O/O_0 \approx 10^{-2}$) even after running over 8000 simulations. We observe the same behaviour for a trajectory optimization test, similar to Fig. 4.14, where CMA-ES fails to produce an acceptable solution after multiple hours, whereas our system yields a good result in a few minutes using direct sensitivity analysis and Gauss-Newton optimization; see our supplements for details.

We now show results for artistic control of animations. In particular, we find optimal throwing velocities such that an elastic object hits a specified target after multiple bounces. We first extend the example of Fig. 4.12a by including a wall and increasing the distance to the target, Fig. 4.13a. Depending on the initial conditions, we can now find multiple paths to the target with various bounce patterns, as labelled in the image. In these examples, using smoother tanh friction forces (blue) yields slightly better results than linear ones (green).

We can also throw the Stanford bunny (again including a contact with a wall) such that it lands at a specific target location (Fig. 4.13b). In this example, the objective function measures the squared distance to the target pose for each mesh node, and also includes a regularizer that additionally penalizes solutions where the bunny falls over.

After performing parameter estimation for a tennis ball, as described in the previous Section, we also optimize initial conditions for a new throw such that the tennis ball hits a specific location on the wall after bouncing off a table once. For the resulting initial position and velocity, we then generate a throwing motion (shown in our video [113]) for a UR5 robot using a standard inverse kinematics model.

4.7.4.2 *Manipulation*

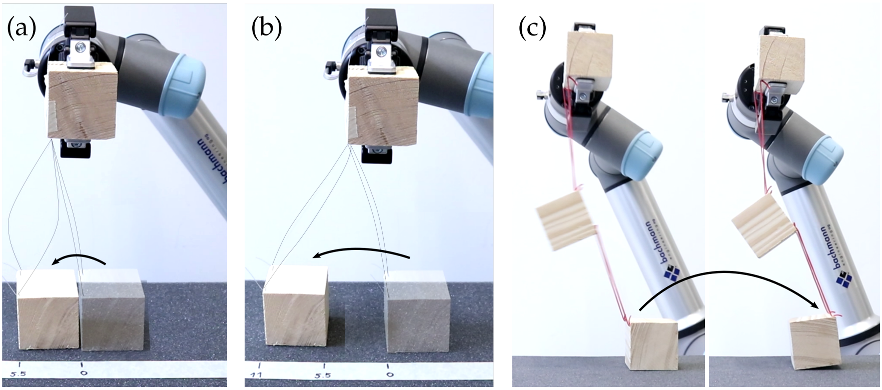


FIGURE 4.14: Optimizing end-effector trajectories for robotic control of coupled dynamic systems. Our robot drags a rigid cube attached with nylon strings over a distance of 5 cm (a) or 11 cm (b) respectively, and actuates a coupled system composed of two rigid cubes and four elastic rubber bands (c) such that the lower cube is tipped over.

We present various applications to robot control using trajectory optimization and show that our contact-aware differentiable simulation is well-suited for these types of problems. We optimize for per-time-step control trajectories $\mathbf{p}$ representing the position and orientation of a robotic end-effector over time.

We manually define targets for rigid objects or point masses at specific moments in time. The objective function O again measures squared distances between the simulated and target state at the specified time. To ensure temporal smoothness of parameters, we add the regularization term $\beta \sum_i \|\mathbf{p}_{i+1} - \mathbf{p}_i\|^2$ to O.

We first optimize the trajectory for a 6-DOF robotic end-effector over a time interval of 1 s, and test our results on a real-world *Universal Robotics UR5* robot running at a controller frequency of 60 Hz, Fig. 4.14. This test shows that we can effectively optimize for a large number of parameters, as each time step ($\Delta_t = 1/60$ s) has its own set of end-effector coordinates. We also demonstrate that our simulated results carry over to the real world by having the robot perform the resulting motion repeatedly for comparison.

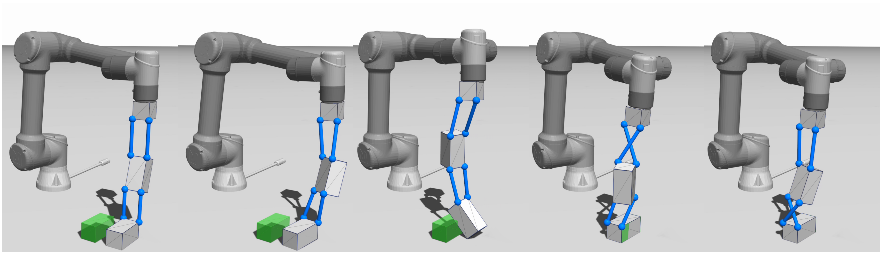


FIGURE 4.15: Trajectory optimization of three rigid bodies linked together by soft springs (blue). The target, green, is to translate and rotate the bottom rigid body by half a turn.

Our system also works for more complex dynamical systems and goals. In Figure 4.15, three rigid bodies are linked together with springs and control trajectories for the end-effector of the robot arm are optimized such that at the end of the time, the bottom rigid body is translated and rotated to mach the state of the greeen target. Note that the bottom rigid body is being turned on of its corners.

Another example of the application of our differentiable simulator in a trajectory optimization setting is the manipulation of a sheet modelled as a mass-spring system (Fig. 4.17). The control parameters specify the positions of two handles, each attached to a corner of the sheet. Initially, the cloth lies on the ground facing upwards. The objective function asks for the following motion of this sheet: After 1 s it is to be flipped over and point downward, and later, at $t = 2$ s, it is to be flipped back to its original orientation, as well as translated to the right. By adding a smoothing regularizer for the control parameters, the trajectory optimization finds a solution after a few Gauss-Newton iterations.

In our last example, the goal is to catapult a bunny to a target location by controlling the corner points of a cloth. The is cloth modeled as a mass-spring system, whereas the bunny is represented as a rigid body and has

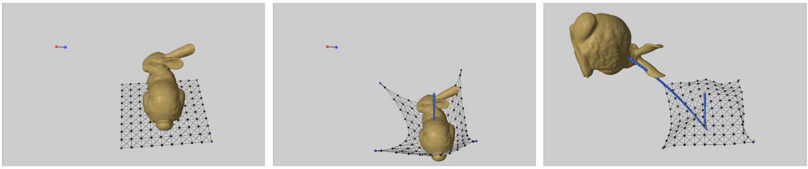


FIGURE 4.16: Trajectory optimization for a rigid body with complex collision geometry.

its collision geometry described as a signed distance field using *OpenVDB*, as laid out in Section 4.5.3.

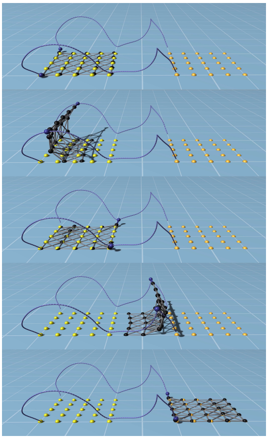


FIGURE 4.17: Trajectory optimization of cloth modeled as mass-spring system. The handles attached attached to the corners of the cloth sheet control the cloth sheet. The first target configuration, yellow dots, is to flip the cloth in place after 1s. The second target configuration, orange dots, is to flip the cloth back to its initial orientation, but translated to the right. The purple lines denote the parameter trajectories that are found using trajectory optimization.

4.7.5 *Self-supervised learning of control policies*

Learning-based methods that leverage neural networks and simulation data to train control policies have achieved impressive results for various control applications. Forward simulation is commonly used as an infinite data

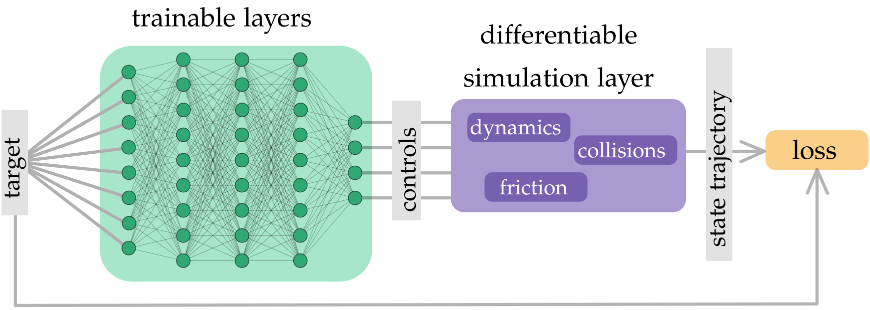


FIGURE 4.18: Differentiable simulation layer in a neural network.

source that is sampled over initial conditions and control parameters to generate training data. While this data-driven approach effectively decouples learning from simulation and thus simplifies implementation, it critically relies on the parameter space sampling reflected in the training data to yield an appropriate coverage in performance space. We pursue an alternative strategy that, rather than using a fixed training set, integrates simulation directly into the loss function, as illustrated in Fig. 4.18, thus enabling the learning algorithm to exploit the map between parameter and performance space provided by our differentiable simulator. Note that this is not our core contribution as similar approaches have been explored in related work, nevertheless, we demonstrate that our framework is well-suited for this important application.

As in the other applications, the objective (or *loss*) function measures the simulation result $\tilde{\mathbf{q}}$ against a desired target behaviour $\mathbf{q}^*$, i.e. $O(\mathbf{q}^*, \tilde{\mathbf{q}})$. We then train the neural network $\phi(\mathbf{q}^*, \mathbf{w})$ to return simulation parameters $\mathbf{p}$ that achieve the given target. The result is a weight vector $\mathbf{w}$ for the network that minimizes the training loss:

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{l=1}^{n} O(\mathbf{q}_l^*, \tilde{\mathbf{q}}(\phi(\mathbf{q}_l^*, \mathbf{w}))). \tag{4.27}$$

Note that the simulated trajectory is a function of the parameters returned by the neural network, i.e. $\tilde{\mathbf{q}}(\phi(\mathbf{q}^*, \mathbf{w}))$. Consequently, a *differentiable* simulation is key to computing gradients during training while avoiding costly finite differencing. We demonstrate this approach on a simple game, where we train a neural network to find the throwing velocity for a rigid ball such that, after a single impact with the ground, it hits a given target position. The objective (or loss) function measures proximity to the target location using a soft minimum over the descending part of the trajectory, and includes a penalty term that discourages solutions without contact. Rather than measuring distance at a specific time, this approach provides more flexibility in terms of timing, allowing the learning algorithm to find better solutions.

We select $n = 1000$ target positions for training and 100 for testing, uniformly sampled in a rectangular region. We train using ADAM [123] with $\beta_1 = 0.95$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$, and a mini-batch size of 5. We start with a learning rate of 10^{-2}, which is reduced by a factor of 0.5 after each epoch. On average, each epoch takes about 450 s of CPU time to compute. The architecture of the network is shown in Fig. 4.19. It is worth noting that, even in this comparatively simple example, accounting for friction in the simulation is crucial for accurate control; a controller trained in a friction-less environment will systematically fail to hit the target.

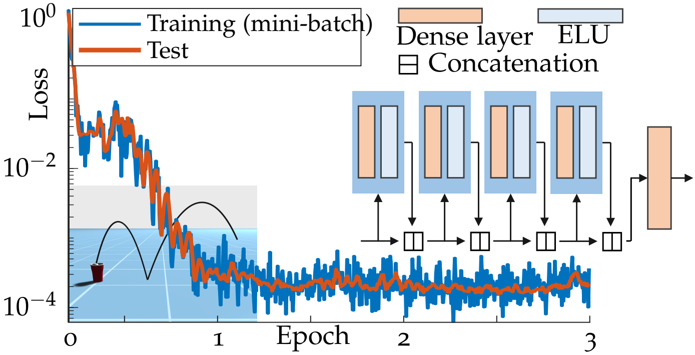


FIGURE 4.19: Convergence of the learning process for the throw controller (left) and the corresponding network architecture (right). The neural network outputs the initial velocity of the simulation such as to hit the given target after one bounce. (ELU: exponential linear unit.)

4.8 DISCUSSION

We present an analytically differentiable dynamics solver that handles frictional contact for soft and rigid bodies. A key aspect of our approach is the use of a soft constraint formulation of frictional contact, which makes our simulation model straightforward to differentiate.

Our results show that penalty-based contact models, especially in the normal component, are sufficiently accurate when combined with implicit time integration, and also enable tuneable, sufficiently smooth contact treatment for gradient-based optimization. We also analyse the effects of penalties against hard constraints with respect to static and dynamic friction. For dynamic motion, where static friction persists only for a short contact duration, penalty-based methods perform adequately and improve the performance of optimization methods built upon these simulations.

In contrast to Chapter 2, we can leverage our differentiable simulator to take into account the compliant nature of motors and feet when generating motion plans for a legged robot. Since our simulation can model the compliant and damped nature of the actuators as well as the deformation of the feet, the trajectory optimization successfully retargets the original control trajectories to regain the target walking speed. We further demonstrate the effectiveness of our approach on a wide range of other robotic applications, including parameter estimation and manipulation tasks, as well as learning-based control. Our optimization examples show that using this framework, gradient-based optimization methods greatly outperform sampling-based methods such as CMA-ES.

All of our examples assume that contact forces obey the isotropic Coulomb model (neglecting viscous contact dynamics, for the time being). While this assumption is valid for a large class of surfaces, anisotropy is an important characteristic of various physical systems such as textiles [72]. Furthermore, in our modelling, we assume a functional representation of the distance metric between different bodies to be readily available, and to be sufficiently smooth. For complex contact scenarios where the geometric representations of the bodies involved in collisions are highly detailed or non-convex, we employ an approach similar [128], where the collision object is represented as a signed distance field.

Our experiment on learning-based control is an initial investigation into combining our differentiable simulator with machine-learning techniques. In the future, we see great promise in leveraging this concept in the context of deep reinforcement learning. By eliminating the need for random exploration, for example, the analytic gradient information that our framework provides is likely to improve sample efficiency.

Similarly, our experiments on performing throwing motions on a real-world robot demonstrate that our simulation results translate to the physical specimens. However, there are still numerous sources of error such as aligning the robot in its environment, unwanted movements of the robot's base, and latency of the hardware controller, which require further investigation. Further, we would like to apply the real2sim2real approach to more complex dynamical systems.

In summary, our experiments demonstrate that our system enables efficient inverse problem solving for various applications in graphics and robotics. For many of these applications, soft constraints with linear penalty forces, combined with implicit integration, lead to physically meaningful and analytically differentiable simulations. Furthermore, we explore options for smoother friction forces, which help reduce noise in the objective function, as well as equality constraints for static friction in cases where physical accuracy is key. In either situation, a soft contact in the normal component enables differentiability and gradient-based optimization.

In the future, we plan to further investigate simulation-driven optimization methods in the context of robotics and control of highly complex multi-body systems that combine rigid and flexible elements.

5

CONCLUSION

5.1 DISCUSSION

Physical models of robots and their environment play a central role in motion planning and the design of new robot morphologies. They can be used to generate motions in an interactive tool, to quickly explore the space of feasible motions when designing a new robot, or to provide parameter estimates and synthesis of control strategies.

This thesis presents different approaches to physical modeling of both legged-wheeled and compliant robots, and leverages the simulation models to generate physically valid and agile motions. In Chapter 2, we introduce a physical model based on centroidal dynamics that focuses on robots with legs and wheels. Using this model, we develop a computational framework for designing and planning motions in an interactive editor for a variety of robot morphologies. Given high-level motion goals, the trajectory optimization method generates physically valid motions taking advantage of the motion capabilities imposed by the leg configurations and type of end-effector, such as point feet, actuated and unactuated wheels. In Chapter 3, we extended our framework with a warm start technique that initiates the trajectory optimization routine, allowing for rapid exploration of different robot designs and their locomotion options. Using this system, we designed a more powerful robot specifically designed to display the potential combining legs and wheels. Equipped with actuated and unactuated wheels or even ice-skates, it can perform different locomotion modes, such as roll- walking, skating, or swizzling, all of which emerge as solutions to a trajectory optimization problem given high-level goals. Furthermore, we demonstrate the versatility of our computational framework by synthesizing control trajectories for Anymal on wheels and using them as reference trajectories for a state-of-the-art feedback controller.

While our computational approach to motion planning is capable of generating various agile and dynamic motions, the underlying physical model does not capture a variety of physical characteristics of the real robot, such as compliance, deformation, or torque limits of actuators. However, most of today's robots do feature deformable objects, such as soft feet, and compliant components, such as PD -controlled actuators. Moreover, we use a fixed footfall pattern and restrict the ground reaction forces to the non-slip region of Coulomb's friction law. In our experiments with the prototypes shown in Chapters 2 and 3, we encountered these limitations as we advanced to increasingly dynamic and complex motions. Furthermore, robot designs that effectively combine rigid and soft structures could be the key to more agile and lifelike motions.

To address these issues, in Chapter 4 we developed a differentiable dynamics model that can simulate both rigid and deformable objects, uses a mollified frictional contact model, and allows modeling of compliant actuators. Our method circumvents the main difficulties associated with the non-smooth nature of frictional contact. We combine this contact model with a unified treatment of rigid and deformable objects and fully-implicit time integration to obtain a robust and efficient dynamics solver that is analytically differentiable. In conjunction with sensitivity analysis, our formulation enables trajectory optimization taking into account the full body dynamics, compliant motors, and soft feet. We further showcase the effectiveness of our differentiable simulation in applications beyond locomotion, such as parameter estimation, object manipulation, and self-supervised learning of control policies.

5.2 CONCLUSION

Motion planning for robots with arbitrary morphologies and design features is a challenging problem, as it amounts to determining a large number of actuation signals to control the high-dimensional state trajectory of a nonlinear dynamical system. In this thesis we have shown that motion generation for robots with legs, wheels, and compliant components becomes a well-solvable problem by using trajectory optimization with an appropriate physical model.

The dynamics model and trajectory optimization of our computational framework allow for interactive rates when editing motions, while generating control trajectories that transfer well to the real world. This human-in-the-loop optimization approach has proven critical to exploiting the full motion capabilities of arbitrary robot design, as the space of physically valid motions can sometimes be unintuitive, especially for designs that combine legs and wheels. Our approach is not limited to toy-like robotic creatures, but can also be used in the design of more powerful legged robots, or in the synthesis of dynamic motions that serve as reference trajectories in a feedback controller.

We have also shown that a differentiable physical model can be used to optimize motion trajectories that take into account the full dynamics of a legged robot as well as the compliance of its actuators and feet. A penalty-based approach to friction and contact fits well in a differentiable simulation model that can have many applications beyond legged locomotion.

5.3 FUTURE WORK

The results presented in this thesis point in exciting directions for future work. Our trajectory optimization runs at interactive rates, but does not converge fast enough to be used in a closed loop controller. We are encouraged by the significant improvements in convergence rates achieved by the Filtered Hessian strategy described in Section 2.4.3. Our ultimate goal is to make the trajectory optimization process faster than real-time. This will enable online computation of motion plans taking into account dynamic obstacles in the environment and providing full-body feedback strategies in response to unplanned perturbations.

There are also interesting opportunities to improve our motion optimization model and the manufacturing process of robots designed using our framework. For example, the physical prototype of our SkaterBot robot moves more slowly than the simulation model. This is not surprising since the wheels in the simulation have no backlash or friction losses and the motors can track the planned movements very accurately. Increasing the accuracy of our predictive model to better account for real-world constraints would also be beneficial and allow us to produce increasingly agile motions. For example, drift maneuvers make very effective use of glide, which is currently beyond the capabilities of our model.

As for differentiable simulation, there are many exciting opportunities to expand its modeling capabilities and performance. We would like to increase the modeling options for deformable bodies, for example, by introducing model reduction of FEM meshes or cloth models that account for bending. This would open up the space of possible applications. Performance improvement in deformation simulation could enable the development of closed-loop systems, such as model predictive control or whole-body controllers for hybrid rigid-soft robots.

Another interesting starting point is to improve contact model and collision detection capabilities. Macklin et al. [128] propose some interesting methods to account for collisions between cloth and sharp features and embed a signed distance field in a deformable cage. This would allow a more accurate prediction of the interactions between deformable and rigid objects. In addition, we intend to further investigate the representation of signed distance fields using neural networks and their use in a differentiable simulator.

We also plan to extend the application of the real2sim2real approach to more complex dynamical systems. For example, we can envision a system in which a robotic arm picks up an object, chooses a simulation model and parameters, and then begins to manipulate that object. The differentiable simulation model could also be useful in evaluating the robustness of a dynamical system. Derivatives of state trajectories with respect to potential perturbations or unknown variables (such as the friction of the floor) could be used to aid in decision making when designing a robot or setting the parameters of a controller.

Finally, our experiment on learning-based control is an initial investigation into combining our differentiable simulator with machine learning techniques. In the future, we see great potential in using this approach in the context of deep reinforcement learning. For example, the analytical gradient information provided by our framework can improve sampling efficiency by eliminating the need for random exploration. Research efforts, such as [129], are very promising.

A

APPENDIX

A.1 RIGID BODY THEORY

A.1.1 *Parameterization of Rotation*

We parameterize the rotation of a rigid body with exponential coordinates $\boldsymbol{\theta}$, using the exponential map $\mathbf{R}(\boldsymbol{\theta}) = \lim_{n\to\infty}(\mathbf{I} + [\boldsymbol{\theta}])^n$. For $||\boldsymbol{\theta}|| > \epsilon$ we compute $\mathbf{R}(\boldsymbol{\theta})$ with the Euler-Rodrigues formula, where we use $\epsilon = 10^{-8}$ in all our examples. For $||\boldsymbol{\theta}|| \leq \epsilon$, we use the first-order approximation to the exponential map $\mathbf{R}(\boldsymbol{\theta}) = \mathbf{I} + [\boldsymbol{\theta}]$. The derivative of $d\mathbf{R}/d\boldsymbol{\theta}$ is computed analytically using the derivation by Gallego and Yezzi [2015], and for $||\boldsymbol{\theta}|| \leq \epsilon$ we use $d\mathbf{R}/d\boldsymbol{\theta} = [\mathbf{I}]$. The second-order derivative of the rotation matrix is computed using symbolic differentiation.

A.1.1.1 *Jacobians*

Closed forms of the left Jacobian of the exponential coordinates and its inverse are

$$\mathbf{J}_l(\boldsymbol{\theta}) = \mathbf{I} + \frac{1-\cos(\theta)}{\theta^2}[\boldsymbol{\theta}]_\times + \frac{\theta - \sin(\theta)}{\theta^3}[\boldsymbol{\theta}]_\times^2$$

$$\mathbf{J}_l^{-1}(\boldsymbol{\theta}) = \mathbf{I} - \frac{1}{2}[\boldsymbol{\theta}]_\times + \left(\frac{1}{\theta^2} - \frac{1+\cos(\theta)}{2\theta\sin(\theta)}\right)[\boldsymbol{\theta}]_\times^2$$

A.1.1.2 *Derivation of Reparameterization Formula*

The formula (4.18) can be obtained by taking the time derivative of (4.16). Another way is to start at equation (4.17) and plug in the definitions of the Jacobian and its inverse. First, we note that $\boldsymbol{\theta}' = -(\theta - 2\pi)\mathbf{u}$ and consequently $\sin\boldsymbol{\theta}' = \sin(2\pi - \theta) = -\sin\theta$ and $\cos\boldsymbol{\theta}' = \cos(2\pi - \theta) = \cos\theta$. We insert these expressions in $\mathbf{J}_l^{-1}(\boldsymbol{\theta}')$ to get

$$\begin{aligned}\mathbf{J}_l^{-1}(\hat{\boldsymbol{\theta}}') &= \mathbf{I} - \frac{1}{2}[\boldsymbol{\theta}]_\times + [\boldsymbol{\theta}']_\times^2 \frac{1}{||\boldsymbol{\theta}'||^2}\left(1 - \frac{||\boldsymbol{\theta}'||}{2}\frac{\sin||\boldsymbol{\theta}'||}{1-\cos||\boldsymbol{\theta}'||}\right) \\ &= \mathbf{I} + [\mathbf{u}]_\times \frac{(2\pi - \theta)}{2} + [\mathbf{u}]_\times^2 \left(1 + \frac{(2\pi-\theta)}{2}\frac{\sin\theta}{1-\cos\theta}\right).\end{aligned}$$

Using the expression $[\mathbf{u}]_\times^3 = -[\mathbf{u}]_\times$ and $[\mathbf{u}]_\times^4 = -[\mathbf{u}]_\times^2$, we obtain

$$
\begin{aligned}
&J_l^{-1}(\boldsymbol{\theta}')J_l(\boldsymbol{\theta})\\
&\qquad = \left[\mathbf{I} + [\mathbf{u}]_\times \frac{(2\pi - \theta)}{2} + [\mathbf{u}]_\times^2 \left(1 + \frac{(2\pi-\theta)}{2}\frac{\sin\theta}{1-\cos\theta}\right)\right]\\
&\qquad\quad \cdot \left[\mathbf{I} + [\mathbf{u}]_\times \left(\frac{1-\cos\theta}{\theta}\right) + [\mathbf{u}]_\times^2 \left(\frac{\theta - \sin\theta}{\theta}\right)\right]\\
&= \mathbf{I} + [\mathbf{u}]_\times^2 \left[1 + \frac{2\pi - \theta}{2\theta(1-\cos\theta)}\left((1-\cos\theta)^2 + \theta\sin\theta - (\theta - \sin\theta)\sin\theta\right)\right]\\
&= \mathbf{I} + [\mathbf{u}]_\times^2 \left[1 + \frac{2\pi - \theta}{2\theta(1-\cos\theta)}\left(1 - 2\cos\theta + \cos^2\theta + \sin^2\theta\right)\right]\\
&= \mathbf{I} + [\mathbf{u}]_\times^2 \left[1 + \frac{2\pi - \theta}{2\theta(1-\cos\theta)}\left(2(1-\cos\theta)\right)\right]\\
&= \mathbf{I} + [\mathbf{u}]_\times^2 \left[1 + \frac{2\pi - \theta}{\theta}\right] = \mathbf{I} + [\mathbf{u}]_\times^2 \frac{2\pi}{\theta}
\end{aligned}
$$

A.1.1.3 *Derivatives of Reparameterization Formulas*

If a reparameterization happens in the forward simulation, we need to account for it when computing derivatives with respect to simulation parameters. Below we provide derivatives of the reparameterization formula (4.16).

$$
\begin{aligned}
\frac{\partial \boldsymbol{\theta}'}{\partial \boldsymbol{\theta}} &= \frac{2\pi}{\theta}\frac{\partial [\mathbf{u}]_\times^2 \dot{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} + [\mathbf{u}]_\times^2 2\pi \frac{\partial}{\partial \boldsymbol{\theta}}\left(\frac{\dot{\boldsymbol{\theta}}}{\theta}\right) && \text{(A.1)}\\
&= -\frac{2\pi}{\theta^2}((\mathbf{u}^T\dot{\boldsymbol{\theta}})\mathbf{I} + \mathbf{u}\dot{\boldsymbol{\theta}}^T)[\mathbf{u}]_\times^2 - [\mathbf{u}]_\times^2 \frac{2\pi}{\theta^2}\dot{\boldsymbol{\theta}}\mathbf{u}^T && \text{(A.2)}\\
&= -\frac{2\pi}{\theta^2}\left[((\mathbf{u}^T\dot{\boldsymbol{\theta}})\mathbf{I} + \mathbf{u}\dot{\boldsymbol{\theta}}^T)[\mathbf{u}]_\times^2 + [\mathbf{u}]_\times^2 \dot{\boldsymbol{\theta}}\mathbf{u}^T\right] && \text{(A.3)}
\end{aligned}
$$

Where we used the following expressions from Gallego et al. [112] to get from the first to the second line: $\frac{\partial [\mathbf{u}]_\times^2 \mathbf{a}}{\partial \boldsymbol{\theta}} = -\frac{1}{\theta}((\mathbf{u}^T\mathbf{a})\mathbf{I} + \mathbf{u}\mathbf{a}^T)[\mathbf{u}]_\times^2$ and $\frac{\partial \frac{\mathbf{a}}{\theta}}{\partial \boldsymbol{\theta}} = -\frac{1}{\theta^3}\mathbf{a}\boldsymbol{\theta}^T = -\frac{1}{\theta^2}\mathbf{a}\mathbf{u}^T$ with $\mathbf{a}$ being a vector independent of $\boldsymbol{\theta}$. The derivative of (4.16) with respect to $\dot{\boldsymbol{\theta}}$ is trivial.

A.1.2 *Rigid bodies*

A rigid-body frame is defined by its centre of mass $\mathbf{c}$ and rotational degrees of freedom $\boldsymbol{\theta}$. A point $\hat{\mathbf{x}}$ in rigid-body coordinates is transformed to world coordinates with $\mathbf{x} = \mathbf{c} + \mathbf{R}(\boldsymbol{\theta})\hat{\mathbf{x}}$. Similarly, $\mathbf{w} = \mathbf{R}(\boldsymbol{\theta})\hat{\mathbf{w}}$ maps a vector $\hat{\mathbf{w}}$ from rigid-body to world coordinates.

The linear velocity $\mathbf{v}$ is equal to the rate of change of the rigid body's centre of mass $\mathbf{v} = \dot{\mathbf{c}}$. The angular velocity $\boldsymbol{\omega}$, however, describes the rate $\|\boldsymbol{\omega}\|$ at which an object rotates around an axis $\boldsymbol{\omega}/\|\boldsymbol{\omega}\|$, and thus $\boldsymbol{\omega} \neq \dot{\boldsymbol{\theta}}$. To relate the angular velocity $\boldsymbol{\omega}$ to the rate of change of rotational degrees of freedom, $\dot{\boldsymbol{\theta}}$, consider a vector $\mathbf{a}$: its rate of change due to an angular velocity $\boldsymbol{\omega}$ is computed as $\dot{\mathbf{a}} = \boldsymbol{\omega} \times \mathbf{a}$. Since the column vectors of $\mathbf{R}$ are just the axes of the rigid body's coordinate frame, $\dot{\mathbf{R}}$ can therefore be computed with

$$\dot{\mathbf{R}} = [\boldsymbol{\omega}]\mathbf{R},$$

where $[\cdot]$ denotes the skew-symmetric matrix corresponding to the left-cross product. $\mathbf{R}$ being an orthogonal matrix, we find

$$[\boldsymbol{\omega}] = \dot{\mathbf{R}}\mathbf{R}^T = \sum_j \frac{\partial \mathbf{R}}{\partial \theta_j}\mathbf{R}^T\dot{\theta}_j = \sum_j [(\mathbf{J}_l)_j]\dot{\theta}_j,$$

where $(\mathbf{J}_l)_j$ is the j-th column vector of the Jacobian $\mathbf{J}_l$, representing the skew-symmetric matrix $(\partial\mathbf{R}/\partial\theta_j)\mathbf{R}^T$.

Note that $\mathbf{J}_l$ is a function of $\boldsymbol{\theta}$. We can consequently write the angular velocity in terms of the generalized coordinates and their time derivative:

$$\boldsymbol{\omega}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \mathbf{J}_l(\boldsymbol{\theta})\dot{\boldsymbol{\theta}}$$

A.1.3 *Rigid body dynamics*

Using the above mapping from $\dot{\boldsymbol{\theta}}$ to $\boldsymbol{\omega}$, we can write the Newton-Euler equations in the form of Eq. (1), where the generalized mass and forces are

$$\hat{\mathbf{M}} = \begin{pmatrix} m\mathbf{I} & 0 \\ 0 & \mathbf{J}_l^T\mathbf{I}_c\mathbf{J}_l \end{pmatrix}$$

and

$$\hat{\mathbf{f}} = \begin{pmatrix} \mathbf{I} \\ \mathbf{J}_l^T[\hat{\mathbf{x}}] \end{pmatrix} \mathbf{f} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}),$$

respectively. The first term in $\hat{\mathbf{f}}$ maps forces from the body's surface to generalized coordinates and the fictitious force term $\mathbf{C}$ is defined as

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} 0 \\ \mathbf{J}_l^T\mathbf{I}_c\dot{\mathbf{J}}_l\dot{\boldsymbol{\theta}} + \mathbf{J}_l^T[\mathbf{J}_l\dot{\boldsymbol{\theta}}]\mathbf{I}_c\mathbf{J}_l\dot{\boldsymbol{\theta}} \end{bmatrix}.$$

This term appears because $\dot{\boldsymbol{\theta}}$ is *not* the same as the angular velocity $\boldsymbol{\omega}$.

To solve rigid-body dynamics implicitly, we need to compute the following derivatives:

$$\left[\frac{\partial(\mathbf{J}_l)_j}{\partial\theta_i}\right]_\times = \frac{\partial^2\mathbf{R}}{\partial\theta_j\partial\theta_i}\mathbf{R}^T + \frac{\partial\mathbf{R}}{\partial\theta_j}\frac{\partial\mathbf{R}}{\partial\theta_i}^T,$$

$$\begin{aligned}\left[\frac{\partial^2(\mathbf{J}_l)_j}{\partial\theta_i\partial\theta_k}\right]_\times =& \frac{\partial^3\mathbf{R}}{\partial\theta_j\partial\theta_i\partial\theta_k}\mathbf{R}^T + \frac{\partial^2\mathbf{R}}{\partial\theta_j\partial\theta_i}\frac{\partial\mathbf{R}}{\partial\theta_k}^T\\ &+ \frac{\partial^2\mathbf{R}}{\partial\theta_j\theta_k}\frac{\partial\mathbf{R}}{\partial\theta_i}^T + \frac{\partial\mathbf{R}}{\partial\theta_j}\frac{\partial^2\mathbf{R}}{\partial\theta_i\partial\theta_k}^T,\end{aligned}$$

and

$$\frac{\partial\dot{\mathbf{J}}_l}{\partial\theta_i} = \sum_j \frac{\partial^2\mathbf{J}_l}{\partial\theta_j\theta_i}\dot{\boldsymbol{\theta}}.$$

The rotated moment of inertia is computed as $\mathbf{I}_c = \mathbf{R}^T\mathbf{I}_0\mathbf{R}$ (where $\mathbf{I}_0$ is the inertia in rigid-body coordinates). Its derivative with respect to $\boldsymbol{\theta}$ is

$$\frac{\partial\mathbf{I}_c}{\partial\theta_i} = \frac{\partial\mathbf{R}}{\partial\theta_i}\mathbf{I}_0\mathbf{R}^T + \mathbf{R}\mathbf{I}_0\frac{\partial\mathbf{R}}{\partial\theta_i}^T.$$

A.2 SUPPLEMENTAL MATERIAL FOR CHAPTER 4

A.2.1 *Comparison to CMA-ES*

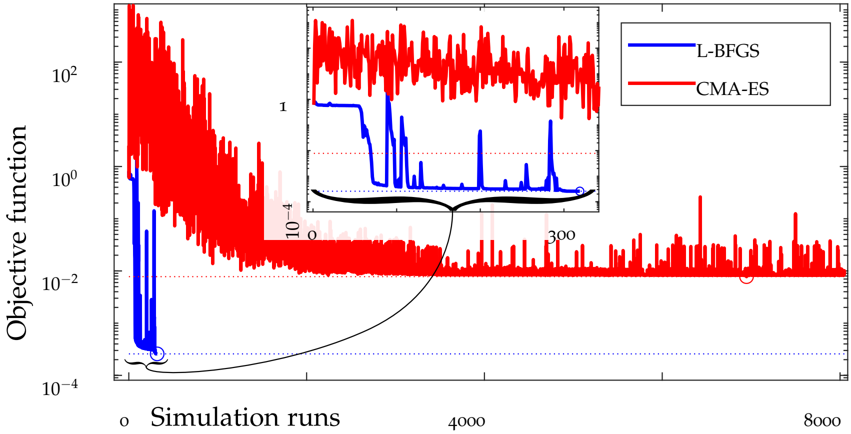


FIGURE A.1: Optimizing throwing the bunny to a specific target. Graph shows all simulation runs, including ones during line search for L-BFGS. Circles mark best result per method.

In this Section we show convergence plots of optimizations using our differentiable simulation framework compared to a gradient-free CMA-ES approach. We use the CMA implementation by Nikolaus Hansen[1] in the first comparison (Fig. A.1), and the implementation by Alexander Fabisch[2] in the second one (Fig. A.2); both with their default parameter values.

A.2.2 *Scaling of computation time*

Here, we test how the computation time scales with increasing number of objects in a simulation scene. In this test case, a chain of N rigid bodies is simulated. The first rigid body is connected to two springs anchored in world space. Similarly, each following rigid body is connected to the previous one by two springs. For all rigid bodies, collision ($k_n = 10^3$, $k_d = 10^{-3}$) and friction ($\mu = 0.5$) with the ground plane is enabled. The total simulated time is $t = 2s$ with a time step of $\Delta t = 1/500$ s.

---

[1] https://github.com/cma-es/c-cmaes

[2] https://github.com/AlexanderFabisch/CMA-ESpp

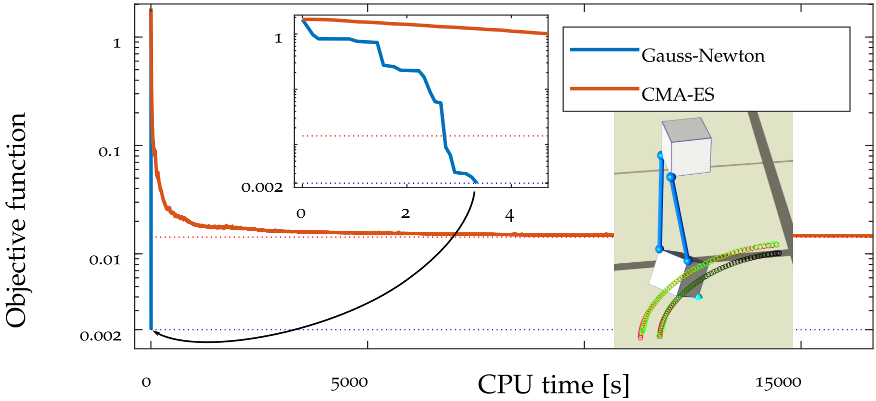


Figure A.2: Optimizing control trajectories for dragging a cube along a circular arc. Graph shows best result per iteration (filtering out line-search evaluations for Gauss-Newton, and sub-optimal samples per generation for CMA).

Figure A.3 shows the resulting computation time for $N = 1..40$, exhibiting fairly linear scaling.

A.2.3 *Number of contact points*

The number of contact points influences the collision response. Increasing the number of contact points, while keeping the contact penalties (k_n, k_d) fixed, results in a stiffer collision response. Conversely, when scaling the penalties by the surface area each contact point represents, we arrive at the same total contact force regardless of the number of contact points (for a planar collision geometry). In Fig. A.4 the restitution is plotted against the number of contact points per edge of a rigid cube. For each sample we scale the penalty factor according to the surface area each contact point represents, e.g. $k_{n,N} = k_{n,0}/N^2$ and $k_{d,N} = k_{d,0}/N^2$ for N contact points per edge, where $k_{n,0}$, $k_{d,0}$ are nominal stiffness and damping coefficients.

A.2.4 *High-stiffness elastic material*

In this example we show an optimization using continuation of the penalty stiffness ($k_n = 100..10^6$) such that a deformable cube made of a stiff elastic material (shear modulus $\mu = 10$ MPa) reaches a given target position. The

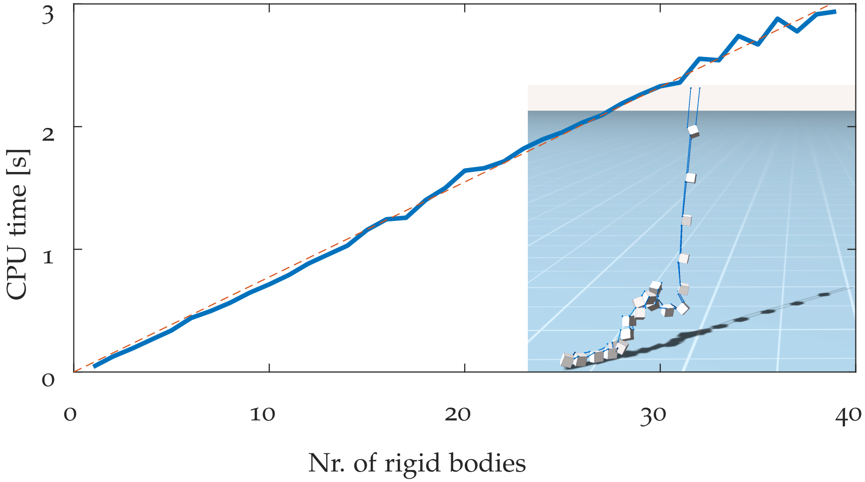


Figure A.3: Computation time per number of rigid bodies in a "chain" setup. The inset image shows a screenshot of this test with $N = 20$.

optimization must find the initial velocity, such that the cube drops to the ground and then slides to the target.

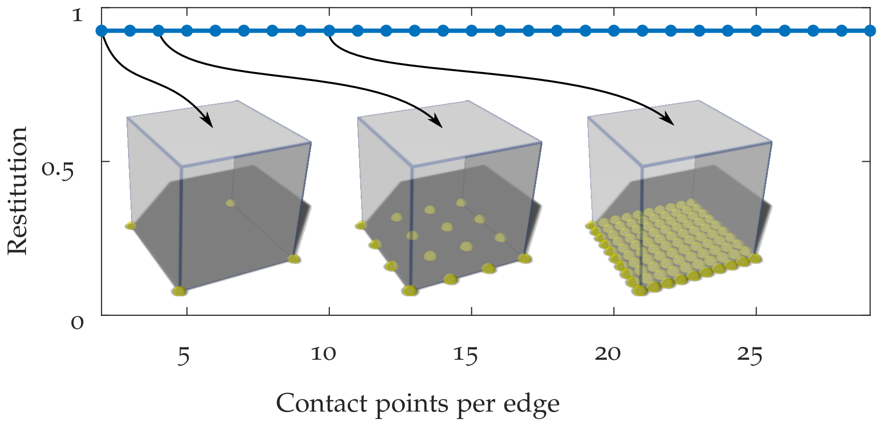


Figure A.4: Restitution remains constant as the number of contact points increases when scaling penalty factors by surface area. Inset images show the contact point distribution for 2, 4, and 10 points per edge respectively.

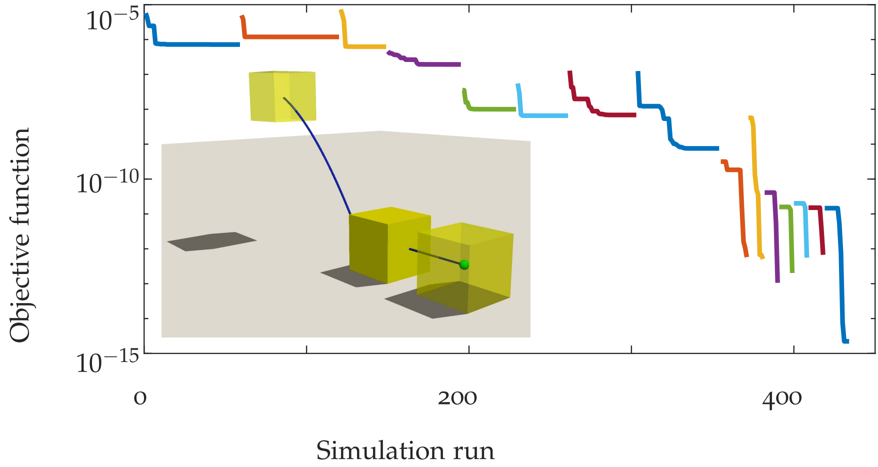


Figure A.5: Best objective function value per simulation run using continuation on the contact penalty factor. In each continuation step (colours), the penalty factor doubles, starting from 100, up to the final value of 10^6. The inset image shows the final result.

A.2.5 *Overview of Simulation Parameters*

Figure	E [Pa]	ν [Pa s]	ρ [kg/m^3]	c_f [1]	m	t_{sim} [s]	t_{opt}	i_{opt}
4.1 lin.	2.1E+03	0 *	150	0.4	879	26.24		
4.1 tanh	2.1E+03	0 *	150	0.4	879	30.86		
4.1 hyb.	2.1E+03	0 *	150	0.4	879	31.2		
4.1 SQP	2.1E+03	0 *	150	0.4	879	66.78		
4.4 lin.	2.1E+04	0.1	150	0.4 (0.8)	1930	119.19		
4.4 tanh	2.1E+04	0.1	150	0.4 (0.8)	1930	178.5		
4.4 hyb.	2.1E+04	0.1	150	0.4 (0.8)	1930	180.75		
4.12a (0)	2.0E+04	0.0125	150	0.4	451	6.92	27.33	4
4.12a (1)	2.0E+04	0.0125	150	0.4	451	8.65	222.54	24
4.12a (2)	2.0E+04	0.0125	150	0.4	451	8.62	265.21	28
4.12b	2.0E+04	0.0125	150	0.4	451	8.21	01:01:11	453
4.13b	2.4E+04	0 *	90	0.2 (0.4)	2729	71.28	05:59:32	322
4.10 left *(initial mat. param.)*	*2.1E+05*	*0.025*	86	*0.4*	1351	(4.13) 38.35	05:08:42	(104) 586
Second motion (video [113])	3.3E+04	6.225	86	0.47	1351	13.38	158.04	105
4.10 right *(initial mat. param.)*	*2.0E+04*	*0.525*	77	*0.4*	1501	(3.08) 421.70	06:58:22.30	(150) 350
Second motion (video [113])	*1.5E+04*	0.7183	77	1.16	1501	1.89	430.35	112

TABLE A.1: Material parameters, performance on our FEM examples. Timings obtained on a 4 × 3.5 GHz CPU with 16 GB RAM. Columns: Young's modulus E, viscosity ν (* BDF1), mass density ρ, coefficient of friction c_f (values in braces refer to walls), number of elements in the mesh m, sim. runtime t_{sim}, runtime of the entire optimization t_{opt} (in sec. or hh:mm:ss), number of sim. runs i_{opt}.

Example	Figure	$\lvert\mathbf{q}\rvert$	$\lvert\mathbf{p}\rvert/n_t$	n_t	Δt [s]	k_n	k_d	k	k_{md}	c_f [1]	β	t_{sim} [s]	t_{opt}	i_{opt}
Cube dragging	4.15	12	6	60	1/60	200	1e-3	5	n.a.	0.5	1e-2	0.044	00:03.3	20
Mass-spring flipping	4.17	75	6	120	1/60	1e3	1e-3	1000	n.a.	0.5	1	0.109	00:54	25
Compliant robot, soft feet	4.8	414	12	144	1/60	1e5	1e-3	5e5 (1e5)	10	0.8	1e-3	1.552	27:54	17

TABLE A.2: Simulation parameters and performance on our trajectory optimization examples. Columns: number of degrees of freedom $|\mathbf{q}|$, parameters per time step $|\mathbf{p}|/n_t$, time steps n_t, time step size Δt, contact stiffness k_n, contact damping k_d, stiffness of constraints k, compliant motors (braces), motor damping k_{md}, coefficient of friction c_f, runtime t_{sim}, runtime for the entire optimization t_{opt}, optimization iterations i_{opt}.

BIBLIOGRAPHY

---

1. Megaro, V., Thomaszewski, B., Nitti, M., Hilliges, O., Gross, M. & Coros, S. Interactive design of 3D-printable robotic creatures. *ACM Transactions on Graphics (TOG)* **34**, 216 (2015).
2. Spröwitz, A., Tuleu, A., Vespignani, M., Ajallooeian, M., Badri, E. & Ijspeert, A. J. Towards dynamic trot gait locomotion: Design, control, and experiments with Cheetah-cub, a compliant quadruped robot. *The International Journal of Robotics Research* **32**, 932 (2013).
3. Hirose, S. & Takeuchi, H. *Study on roller-walk (basic characteristics and its control)* in *Proceedings of IEEE International Conference on Robotics and Automation* **4** (1996), 3265.
4. Bjelonic, M., Bellicoso, C. D., de Viragh, Y., Sako, D., Tresoldi, F. D., Jenelten, F. & Hutter, M. Keep rollin'—whole-body motion control and planning for wheeled quadrupedal robots. *IEEE Robotics and Automation Letters* **4**, 2116 (2019).
5. Du, T., Schulz, A., Zhu, B., Bickel, B. & Matusik, W. Computational Multicopter Design. *ACM Trans. Graph.* **35**, 227:1 (2016).
6. Schulz, A., Sung, C., Spielberg, A., Zhao, W., Cheng, R., Grinspun, E., Rus, D. & Matusik, W. Interactive robogami: An end-to-end system for design of robots with ground locomotion. *The International Journal of Robotics Research* **36**, 1131 (2017).
7. Bern, J. M., Chang, K.-H. & Coros, S. Interactive Design of Animated Plushies. *ACM Trans. Graph.* **36**, 80:1 (2017).
8. Lu, L., Sharf, A., Zhao, H., Wei, Y., Fan, Q., Chen, X., Savoye, Y., Tu, C., Cohen-Or, D. & Chen, B. Build-to-last: strength to weight 3D printed objects. *ACM Transactions on Graphics (TOG)* **33**, 97 (2014).
9. Stava, O., Vanek, J., Benes, B., Carr, N. & Měch, R. Stress Relief: Improving Structural Strength of 3D Printable Objects. *ACM Trans. Graph. (Proc. SIGGRAPH)* **31** (2012).
10. Prévost, R., Whiting, E., Lefebvre, S. & Sorkine-Hornung, O. Make it stand: balancing shapes for 3D fabrication. *ACM Transactions on Graphics (TOG)* **32**, 81 (2013).

11. Musialski, P., Auzinger, T., Birsak, M., Wimmer, M., Kobbelt, L. & Wien, T. Reduced-order shape optimization using offset surfaces. *ACM Transactions on Graphics (TOG)* **34**, 102 (2015).
12. Bächer, M., Whiting, E., Bickel, B. & Sorkine-Hornung, O. Spin-it: optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics (TOG)* **33**, 96 (2014).
13. Bächer, M., Bickel, B., James, D. L. & Pfister, H. Fabricating Articulated Characters from Skinned Meshes. *ACM Trans. Graph.* **31**, 47:1 (2012).
14. Ureta, F. G., Tymms, C. & Zorin, D. *Interactive Modeling of Mechanical Objects* in *Proceedings of the Symposium on Geometry Processing* (Eurographics Association, Berlin, Germany, 2016), 145.
15. Calì, J., Calian, D. A., Amati, C., Kleinberger, R., Steed, A., Kautz, J. & Weyrich, T. 3D-printing of Non-assembly, Articulated Models. *ACM Trans. Graph.* **31**, 130:1 (2012).
16. Coros, S., Thomaszewski, B., Noris, G., Sueda, S., Forberg, M., Sumner, R. W., Matusik, W. & Bickel, B. Computational design of mechanical characters. *ACM Transactions on Graphics (TOG)* **32**, 83 (2013).
17. Ceylan, D., Li, W., Mitra, N. J., Agrawala, M. & Pauly, M. Designing and fabricating mechanical automata from mocap sequences. *ACM Transactions on Graphics (TOG)* **32**, 186 (2013).
18. Song, P., Wang, X., Tang, X., Fu, C.-W., Xu, H., Liu, L. & Mitra, N. J. Computational Design of Wind-up Toys. *ACM Trans. Graph.* **36**, 238:1 (2017).
19. Zhang, R., Auzinger, T., Ceylan, D., Li, W. & Bickel, B. Functionality-aware Retargeting of Mechanisms to 3D Shapes. *ACM Trans. Graph.* **36**, 81:1 (2017).
20. Megaro, V., Zehnder, J., Bächer, M., Coros, S., Gross, M. & Thomaszewski, B. A Computational Design Tool for Compliant Mechanisms. *ACM Trans. Graph.* **36**, 82:1 (2017).
21. Villar, N., Scott, J., Hodges, S., Hammil, K. & Miller, C. in *Pervasive Computing* 216 (Springer, 2012).
22. Weichel, C., Lau, M. & Gellersen, H. *Enclosed: a component-centric interface for designing prototype enclosures* in *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction* (2013), 215.

23. Follmer, S., Savage, V., Li, J. & Hartmann, B. *Makers' Marks: Physical Markup for Designing and Fabricating Functional Objects* in *UIST'15 Proceedings of the 28th annual ACM symposium on User interface software and technology* (2015).
24. Bächer, M., Hepp, B., Pece, F., Kry, P. G., Bickel, B., Thomaszewski, B. & Hilliges, O. *DefSense: Computational Design of Customized Deformable Input Devices* in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (ACM, San Jose, California, USA, 2016), 3806.
25. Bharaj, G., Coros, S., Thomaszewski, B., Tompkin, J., Bickel, B. & Pfister, H. *Computational Design of Walking Automata* in *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (ACM, Los Angeles, California, 2015), 93.
26. Megaro, V., Thomaszewski, B., Nitti, M., Hilliges, O., Gross, M. & Coros, S. Interactive Design of 3D-printable Robotic Creatures. *ACM Trans. Graph.* **34,** 216:1 (2015).
27. Umentani, N., Igarashi, T. & Mitra, N. J. Guided Exploration of Physically Valid Shapes for Furniture Design. *Commun. ACM* **58,** 116 (2015).
28. Pérez, J., Otaduy, M. A. & Thomaszewski, B. Computational Design and Automated Fabrication of Kirchhoff-plateau Surfaces. *ACM Trans. Graph.* **36,** 62:1 (2017).
29. Ha, S., Coros, S., Alspach, A., Kim, J. & Yamane, K. *Joint Optimization of Robot Design and Motion Parameters using the Implicit Function Theorem* in *Robotics: Science and Systems* (2017).
30. Endo, G. & Hirose, S. *Study on Roller-Walker - Adaptation of characteristics of the propulsion by a leg trajectory -* in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2008), 1532.
31. Smith, J. A., Sharf, I. & Trentini, M. *PAW: a hybrid wheeled-leg robot* in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* (2006), 4043.
32. BostonDynamics. *Handle, https://www.bostondynamics.com/handle* 2017.
33. *Accompanying video of Chapter 2.* http://crl.ethz.ch/videos/skaterbots.mp4. 2018.
34. Desai, R., Yuan, Y. & Coros, S. *Computational Abstractions for Interactive Design of Robotic Devices* in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)* (2017).

35. ODE. *Open Dynamics Engine, http://www.ode.org/* 2007.
36. Orin, D. E., Goswami, A. & Lee, S.-H. Centroidal Dynamics of a Humanoid Robot. *Auton. Robots* **35**, 161 (2013).
37. Dai, H., Valenzuela, A. & Tedrake, R. *Whole-body motion planning with centroidal dynamics and full kinematics* in *Humanoids* (2014).
38. Kovar, L., Gleicher, M. & Pighin, F. *Motion Graphs* in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (ACM, San Antonio, Texas, 2002), 473.
39. Smith, J. A., Sharf, I. & Trentini, M. *PAW: a hybrid wheeled-leg robot* in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* (2006), 4043.
40. Handle: Boston Dynamics robot on wheels performs on stage. Youtube. [Online]. Available: https://www.youtube.com/watch?v=-7xvqQeoA8c.
41. De Viragh, Y., Bjelonic, M., Bellicoso, C. D., Jenelten, F. & Hutter, M. Trajectory optimization for wheeled-legged quadrupedal robots using linearized zmp constraints. *IEEE Robotics and Automation Letters* **4**, 1633 (2019).
42. Bjelonic, M., Bellicoso, C. D., Tiryaki, M. E. & Hutter, M. *Skating with a force controlled quadrupedal robot* in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), 7555.
43. Hereid, A. & Ames, A. D. *FROST*: Fast robot optimization and simulation toolkit* in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017), 719.
44. Bledt, G., Powell, M. J., Katz, B., Di Carlo, J., Wensing, P. M. & Kim, S. *MIT Cheetah 3: Design and control of a robust, dynamic quadruped robot* in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), 2245.
45. Dai, H., Valenzuela, A. & Tedrake, R. *Whole-body motion planning with centroidal dynamics and full kinematics* in *14th IEEE-RAS International Conference on Humanoid Robots, Humanoids 2014, Madrid, Spain, November 18-20, 2014* (2014), 295.
46. Reid, W., Pérez-Grau, F. J., Göktogan, A. H. & Sukkarieh, S. *Actively articulated suspension for a wheel-on-leg rover operating on a martian analog surface* in *2016 IEEE International Conference on Robotics and Automation (ICRA)* (2016), 5596.

47. Grand, C., Benamar, F. & Plumet, F. Motion kinematics analysis of wheeled–legged rover over 3D surface with posture adaptation. *Mechanism and Machine Theory* **45**, 477 (2010).

48. Klamt, T. & Behnke, S. *Anytime hybrid driving-stepping locomotion planning* in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017), 4444.

49. Kamedula, M., Kashiri, N. & Tsagarakis, N. G. *On the kinematics of wheeled motion control of a hybrid wheeled-legged centauro robot* in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), 2426.

50. Suzumura, A. & Fujimoto, Y. Real-time motion generation and control systems for high wheel-legged robot mobility. *IEEE Transactions on Industrial Electronics* **61**, 3648 (2013).

51. Krauskopf, B., Osinga, H. M. & Galán-Vioque, J. *Numerical continuation methods for dynamical systems* (Springer, 2007).

52. *Accompanying video of Chapter 3*. http://crl.ethz.ch/videos/skaterbotsRAL.mp4. 2020.

53. Bjelonic, M., Sankar, P. K., Bellicoso, C. D., Vallery, H. & Hutter, M. Rolling in the deep–hybrid locomotion for wheeled-legged robots using online trajectory optimization. *IEEE Robotics and Automation Letters* **5**, 3626 (2020).

54. Bjelonic, M., Grandia, R., Harley, O., Galliard, C. M., Zimmermann, S. & Hutter, M. *Whole-Body MPC and Online Gait Sequence Generation for Wheeled-Legged Robots* in. 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2021); Conference Location: Prague, Czech Republic; Conference Date: September 27 - October 1, 2021 (2021-07).

55. Hutter, M., Gehring, C., Lauber, A., Gunther, F., Bellicoso, C. D., Tsounis, V., Fankhauser, P., Diethelm, R., Bachmann, S., Blösch, M., *et al.* Anymal-toward legged robots for harsh environments. *Advanced Robotics* **31**, 918 (2017).

56. Holl, P., Thuerey, N. & Koltun, V. *Learning to Control PDEs with Differentiable Physics* in *Int. Conf. on Learning Representations* (to appear) (2020).

57. Hu, Y., Liu, J., Spielberg, A., Tenenbaum, J. B., Freeman, W. T., Wu, J., Rus, D. & Matusik, W. ChainQueen: A Real-Time differentiable Physical Simulator for Soft Robotics. *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)* (2019).
58. Loubet, G., Holzschuch, N. & Jakob, W. Reparameterizing Discontinuous Integrands for Differentiable Rendering. *ACM Trans. Graph.* **38** (2019).
59. Brogliato, B. *Nonsmooth Mechanics: Models, Dynamics and Control* (Springer, 1999).
60. Baraff, D. *Fast Contact Force Computation for Nonpenetrating Rigid Bodies* in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques* (1994), 23.
61. Stewart, D. E. & Trinkle, J. C. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *International Journal for Numerical Methods in Engineering* **39**, 2673 (1996).
62. Anitescu, M. & Potra, F. A. Formulating Dynamic Multi-rigid-body Contact Problems with Friction as Solvable Linear Complementarity Problems. *NONLINEAR DYNAMICS* **14**, 231 (1997).
63. Duriez, C., Dubois, F., Kheddar, A. & Andriot, C. Realistic Haptic Rendering of Interacting Deformable Objects in Virtual Environments. *IEEE Transactions on Visualization and Computer Graphics* **12** (2006).
64. Erleben, K. Velocity-Based Shock Propagation for Multibody Dynamics Animation. *ACM Trans. Graph.* **26**, 12 (2007).
65. Pauly, M., Pai, D. K. & Guibas, L. J. *Quasi-Rigid Objects in Contact* in *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Eurographics Association, 2004), 109.
66. Otaduy, M. A., Tamstorf, R., Steinemann, D. & Gross, M. *Implicit Contact Handling for Deformable Objects* in. **28** (2009).
67. Kaufman, D. M., Edmunds, T. & Pai, D. K. Fast Frictional Dynamics for Rigid Bodies. *ACM Transactions on Graphics* **24**, 946 (2005).
68. Harmon, D., Vouga, E., Smith, B., Tamstorf, R. & Grinspun, E. *Asynchronous Contact Mechanics* in *ACM SIGGRAPH 2009 Papers* (2009).
69. Allard, J., Faure, F., Courtecuisse, H., Falipou, F., Duriez, C. & Kry, P. G. Volume Contact Constraints at Arbitrary Resolution. *ACM Trans. Graph.* **29** (2010).

70. Ding, O. & Schroeder, C. Penalty Force for Coupling Materials with Coulomb Friction. *IEEE Transactions on Visualization and Computer Graphics* **26**, 2443 (2020).

71. Li, J., Daviet, G., Narain, R., Bertails-Descoubes, F., Overby, M., Brown, G. E. & Boissieux, L. An Implicit Frictional Contact Solver for Adaptive Cloth Simulation. *ACM Trans. Graph.* **37** (2018).

72. Pabst, S., Thomaszewski, B. & Straundefineder, W. *Anisotropic Friction for Deformable Surfaces and Solids* in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2009), 149.

73. Erleben, K., Macklin, M., Andrews, S. & Kry, P. G. The Matchstick Model for Anisotropic Friction Cones. *Computer Graphics Forum* (2019).

74. Kaufman, D. M., Sueda, S., James, D. L. & Pai, D. K. Staggered Projections for Frictional Contact in Multibody Systems. *ACM Transactions on Graphics (SIGGRAPH Asia 2008)* **27**, 164:1 (2008).

75. Macklin, M., Erleben, K., Müller, M., Chentanez, N., Jeschke, S. & Makoviychuk, V. Non-smooth Newton Methods for Deformable Multi-body Dynamics. *ACM Transactions on Graphics* **38** (2019).

76. Bertails-Descoubes, F., Cadoux, F., Daviet, G. & Acary, V. A Nonsmooth Newton Solver for Capturing Exact Coulomb Friction in Fiber Assemblies. *ACM Trans. Graph.* **30** (2011).

77. Popović, J., Seitz, S. M., Erdmann, M., Popović, Z. & Witkin, A. *Interactive manipulation of rigid body simulations* in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00* (2000).

78. Twigg, C. D. & James, D. L. Many-Worlds Browsing for Control of Multibody Dynamics. *ACM Transactions on Graphics* **26** (2007).

79. Coros, S., Martin, S., Thomaszewski, B., Schumacher, C., Sumner, R. & Gross, M. Deformable Objects Alive! *ACM Trans. Graph.* **31** (2012).

80. Tan, J., Turk, G. & Liu, C. K. Soft Body Locomotion. *ACM Trans. Graph.* **31** (2012).

81. Chen, X., Andrews, S., Nowrouzezahrai, D. & Kry, P. Ballistic Shadow Art. *Proceedings of Graphics Interface 2017* **Edmonton**, 190 (2017).

82. Jain, S. & Liu, C. K. Controlling Physics-Based Characters Using Soft Contacts. *ACM Transactions on Graphics* **30** (2011).

83. Pai, D. K., van den Doel, K., James, D. L., Lang, J., Lloyd, J. E., Richmond, J. L. & Yau, S. H. *Scanning physical interaction behavior of 3D objects* in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01* (2001).
84. Monszpart, A., Thuerey, N. & Mitra, N. J. SMASH: Physics-Guided Reconstruction of Collisions from Videos. *ACM Trans. Graph.* **35** (2016).
85. Ly, M., Casati, R., Bertails-Descoubes, F., Skouras, M. & Boissieux, L. Inverse Elastic Shell Design with Contact and Friction. *ACM Trans. Graph.* **37** (2018).
86. Chen, D., Levin, D. I. W., Matusik, W. & Kaufman, D. M. Dynamics-aware numerical coarsening for fabrication design. *ACM Transactions on Graphics* **36**, 1 (2017).
87. Yunt, K. & Glocker, C. *Trajectory optimization of mechanical hybrid systems using SUMT* in *9th IEEE International Workshop on Advanced Motion Control* (2006).
88. Todorov, E. *A convex, smooth and invertible contact model for trajectory optimization* in *2011 IEEE International Conference on Robotics and Automation* (2011), 1071.
89. Erez, T. & Todorov, E. *Trajectory optimization for domains with contacts using inverse dynamics* in *2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems* (2012).
90. Todorov, E., Erez, T. & Tassa, Y. *MuJoCo: A physics engine for model-based control* in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2012).
91. De Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J. & Kolter, J. Z. in *Advances in Neural Information Processing Systems 31* 7178 (2018).
92. Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I. & Bachem, O. Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation. *CoRR* **abs/2106.13281** (2021).
93. Heiden, E., Millard, D., Coumans, E., Sheng, Y. & Sukhatme, G. S. *NeuralSim: Augmenting Differentiable Simulators with Neural Networks* in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (2021).
94. Schenck, C. & Fox, D. *SPNets: Differentiable Fluid Dynamics for Deep Neural Networks* in *Proceedings of The 2nd Conference on Robot Learning* **87** (PMLR, 2018), 317.

95. Liang, J., Lin, M. & Koltun, V. *Differentiable Cloth Simulation for Inverse Problems* in *Advances in Neural Information Processing Systems 32* (2019), 772.
96. Toussaint, M., Allen, K., Smith, K. & Tenenbaum, J. *Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning* in *Robotics: Science and Systems XIV* (Robotics: Science and Systems Foundation, 2018).
97. Degrave, J., Hermans, M., Dambre, J. & Wyffels, F. A Differentiable Physics Engine for Deep Learning in Robotics. *Frontiers in Neurorobotics* **13**, 6 (2019).
98. Hu, Y., Anderson, L., Li, T.-M., Sun, Q., Carr, N., Ragan-Kelley, J. & Durand, F. DiffTaichi: Differentiable Programming for Physical Simulation. *International Conference on Learning Representations (ICLR)* (2019).
99. Hoshyari, S., Xu, H., Knoop, E., Coros, S. & Bächer, M. Vibration-Minimizing Motion Retargeting for Robotic Characters. *ACM Transactions on Graphics* **38** (2019).
100. Hahn, D., Banzet, P., Bern, J. M. & Coros, S. Real2Sim: visco-elastic parameter estimation from dynamic motion. *ACM Transactions on Graphics* **38**, 1 (2019).
101. Macklin, M., Erleben, K., Müller, M., Chentanez, N., Jeschke, S. & Kim, T.-Y. *Primal/dual descent methods for dynamics* in *Computer Graphics Forum* **39** (2020), 89.
102. Zimmermann, S., Poranne, R., Bern, J. M. & Coros, S. PuppetMaster: robotic animation of marionettes. *ACM Transactions on Graphics* **38** (2019).
103. Liu, C. K. & Jain, S. *A quick tutorial on multibody dynamics* 2012.
104. Shinar, T., Schroeder, C. A. & Fedkiw, R. *Two-way Coupling of Rigid and Deformable Bodies* in *Proceedings of the 2008 Eurographics/ACM SIGGRAPH Symposium on Computer Animation* (2008), 95.
105. Wang, Y., Weidner, N. J., Baxter, M. A., Hwang, Y., Kaufman, D. M. & Sueda, S. RedMax. *ACM Transactions on Graphics* **38** (2019).
106. Pan, Z. & Manocha, D. *Position-Based Time-Integrator for Frictional Articulated Body Dynamics* in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2018).

107. Zimmermann, S., Poranne, R. & Coros, S. Optimal control via second order sensitivity analysis. *CoRR* (2019).
108. Wriggers, P. *Computational Contact Mechanics* (Springer-Verlag GmbH, 2006).
109. Stewart, D. E. Rigid Body Dynamics with Friction and Impact. *SIAM Review*, 3 (2000).
110. Amos, B. & Kolter, J. Z. OptNet: Differentiable Optimization as a Layer in Neural Networks. *ICML'17* 136 (2017).
111. Brown, G. E., Overby, M., Forootaninia, Z. & Narain, R. Accurate dissipative forces in optimization integrators. *ACM Transactions on Graphics* **37** (2019).
112. Gallego, G. & Yezzi, A. A compact formula for the derivative of a 3-D rotation in exponential coordinates. *Journal of Mathematical Imaging and Vision* **51**, 378 (2015).
113. *Accompanying video of Chapter 4.* http://crl.ethz.ch/videos/ADD.mp4. 2020.
114. Quilez, I. *A distance functions for basic primitives* https://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm. Accessed: 2021-03-30.
115. Museth, K. VDB: High-Resolution Sparse Volumes with Dynamic Topology. *ACM Trans. Graph.* **32** (2013).
116. Park, J. J., Florence, P., Straub, J., Newcombe, R. & Lovegrove, S. *DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation* 2019.
117. Michalkiewicz, M., Pontes, J. K., Jack, D., Baktashmotlagh, M. & Eriksson, A. *Implicit Surface Representations As Layers in Neural Networks* in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), 4742.
118. Gropp, A., Yariv, L., Haim, N., Atzmon, M. & Lipman, Y. *Implicit Geometric Regularization for Learning Shapes* 2020.
119. Sitzmann, V., Martel, J. N., Bergman, A. W., Lindell, D. B. & Wetzstein, G. *Implicit Neural Representations with Periodic Activation Functions* in *Proc. NeurIPS* (2020).
120. Catto, E. *Reinventing the spring* in (2011).

121. Bern, J., Banzet, P., Poranne, R. & Coros, S. Trajectory optimization for cable-driven soft robot locomotion. *Robotics: Science and Systems* (2019).
122. Sola, J., Deray, J. & Atchuthan, D. A micro Lie theory for state estimation in robotics. *arXiv preprint arXiv:1812.01537* (2018).
123. Kingma, D. P. & Ba, J. ADAM: A Method for Stochastic Optimization (2014).
124. O'Hara, K. *OptimLib* https://www.kthohr.com/optimlib.html. 2019-11-28. 2019.
125. Nocedal, J. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation* **35**, 773 (1980).
126. Qiu, Y. *LBFGS++* http://yixuan.cos.name/LBFGSpp. 2019-03-12. 2019.
127. Bradley, A. *PDE-constrained optimization and the adjoint method* tech. rep. (Stanford University, 2013).
128. Macklin, M., Erleben, K., Müller, M., Chentanez, N., Jeschke, S. & Corse, Z. Local optimization for robust signed distance field collision. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* **3**, 1 (2020).
129. Mora, M. A. Z., Peychev, M. P., Ha, S., Vechev, M. & Coros, S. *PODS: Policy Optimization via Differentiable Simulation* in *International Conference on Machine Learning* (2021), 7805.

CURRICULUM VITAE

PERSONAL DATA

Name	Moritz Geilinger
Date of Birth	January 29, 1988
Place of Birth	Zürich, Switzerland
Citizen of	Switzerland

EDUCATION

2017 – 2021	ETH Zurich, Zürich, Switzerland Phd Student, Computational Robotics Lab
2012 – 2015	ETH Zurich, Zürich, Switzerland *Final degree:* MSc in Mechanical Engineering
2007 – 2012	ETH Zurich Zürich, Switzerland *Final degree:* Bsc in Mechanical Engineering
– 2006	Literargymnasium Rämibühl Zürich, Switzerland *Final degree:* Matura

EMPLOYMENT

2015 – 2017	Research Engineer *Disney Research,* Zürich, Switzerland

PUBLICATIONS

Articles in peer-reviewed journals:

1. Geilinger, M., Poranne, R., Desai, R., Thomaszewski, B. & Coros, S. Skaterbots: Optimization-Based Design and Motion Synthesis for Robotic Creatures with Legs and Wheels. *ACM Transactions on Graphics (TOG)* **37**, 1 (2018).
2. Geilinger, M., Hahn, D., Zehnder, J., Bächer, M., Thomaszewski, B. & Coros, S. ADD: analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)* **39**, 1 (2020).
3. Geilinger, M., Winberg, S. & Coros, S. A computational framework for designing skilled legged-wheeled robots. *IEEE Robotics and Automation Letters* **5**, 3674 (2020).

Conference contributions:

4. Geilinger, M., Winberg, S. & Coros, S. *Motion synthesis for legged-wheeled robotic creatures* in *AMAM 2019 Conference Proceedings* (2019).