THE UNIVERSITY *of* EDINBURGH

CONCURRENT DESIGN AND MOTION PLANNING IN ROBOTICS USING DIFFERENTIABLE OPTIMAL CONTROL

TRAIKO CVETANOV DINEV

Doctor of Philosophy
School of Informatics
University of Edinburgh

2023

Traiko Cvetanov Dinev:
Concurrent Design and Motion Planning in Robotics using Differentiable Optimal Control
Doctor of Philosophy, 2023

SUPERVISORS:
Prof. Sethu Vijayakumar, Ph.D., FRSE
Steve Tonneau, Ph.D.

ABSTRACT

Robot design optimization (what the robot is) and motion planning (how the robot moves) are two problems that are connected. Robots are limited by their design in terms of what motions they can execute – for instance a robot with a heavy base has less payload capacity compared to the same robot with a lighter base. On the other hand, the motions that the robot executes guide which design is best for the task. Concurrent design (co-design) is the process of performing robot design and motion planning together. Although traditionally co-design has been viewed as an offline process that can take hours or days, we view interactive co-design tools as the next step as they enable quick prototyping and evaluation of designs across different tasks and environments.

In this thesis we adopt a gradient-based approach to co-design. Our baseline approach embeds the motion planning into bi-level optimization and uses gradient information via finite differences from the lower motion planning level to optimize the design in the upper level. Our approach uses the full rigid-body dynamics of the robot and allows for arbitrary upper-level design constraints, which is key for finding physically realizable designs. Our approach is also between 1.8 and 8.4 times faster on a quadruped trotting and jumping co-design task as compared to the popular genetic algorithm covariance matrix adaptation evolutionary strategy (CMA-ES). We further demonstrate the speed of our approach by building an interactive co-design tool that allows for optimization over uneven terrain with varying height.

Furthermore, we propose an algorithm to analytically take the derivative of nonlinear optimal control problems via differential dynamic programming (DDP). Analytical derivatives are a step towards addressing the scalability and accuracy issues of finite differences. We further compared with a simultaneous approach for co-design that optimizes both motion and design in one nonlinear program. On a co-design task for the Kinova robotic arm we observed a 54-times improvement in computational speed.

We additionally carry out hardware validation experiments on the quadruped robot Solo. We designed longer lower legs for the robot, which minimize the peak torque used during trotting. Although we always observed an improvement in peak torque, it was less than in simulation (7.609% versus 28.271%). We discuss some of the sim-to-real issues including the structural stability of joints and slipping of feet that need to be considered and how they can be addressed using our framework.

In the second part of this thesis we propose solutions to some open problems in motion planning. Firstly, in our co-design approach we assumed fixed contact locations and timings. Ideally we would like the motion planner to choose the contacts instead. We solve a related, but simpler problem, which is the control of satellite thrusters, which are similar to robot feet but do not have the constraint of having to be in contact with the ground to exert force on the robot. We introduce a sparse, L_1 cost on control inputs (thrusters) and implement optimization via DDP-style solvers. We use full rigid-body dynamics and achieve bang-bang control via optimization, which is a difficult problem due to the discrete switching nature of the thrusters.

Lastly, we present a method for planning and control of a hybrid, wheel-legged robot. This is a difficult problem, as the robot needs to always actively balance on the wheel even when not driving or jumping forward. We propose the variable-length wheeled inverted pendulum (VL-WIP) template model that captures only the necessary dynamic interactions between wheels and base. We embedded this into a model-predictive controller (MPC) and demonstrated highly dynamic behaviors, including swinging-up and jumping over a gap.

Both of these motion planning problems expand the ability of our motion planning tools to new domains, which is an integral part also of the co-design algorithms, as co-design aims to optimize both design, and motion, together.

ACKNOWLEDGEMENTS

Firstly I would like to thank my supervisor Prof. Sethu Vijayakumar for his support for my research and for creating an environment in the Statistical Learning and Motor Control (SLMC) Group that facilitates the exploration and investigation of new ideas.

I would also like to thank my second supervisor, Dr. Steve Tonneau, for his invaluable feedback, especially when it comes to improving my scientific writing.

I would also like to thank my colleagues and mentors – Vladimir Ivan, Carlos Mastalli, Wolfgang Merkt, Songyan Xin – for their support and teaching me much of what I know about research and robotics.

Finally I would like to thank the rest of the SLMC group who helped me in various way throughout my journey – Andreas, Carlo, Chris, Christian, Daniel, Jaehyun, Jiayi, João, Lei, Marina, Matt, Ran, Ruaridh, Sanghyun, Serena, Theodoros.

PUBLICATIONS

The following is a list of conference and journal articles submitted during my studies. Some parts of this thesis include text and figures from these publications.

JOURNAL ARTICLES

- T. Dinev, C. Mastalli, V. Ivan, S. Tonneau, S. Vijayakumar. Differentiable Optimal Control via Differential Dynamic Programming'. Under review for *IEEE Transactions on Robotics (**TR-O**)*, 2023. (Chapter 3, Chapter 5)

CONFERENCE ARTICLES

- T. Dinev, S. Xin, W. Merkt, V. Ivan, S. Vijayakumar. 'Modeling and Control of a Hybrid Wheeled Jumping Robot'. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (**IROS**)*, Las Vegas, USA, 2020. (Chapter 3)
- T. Dinev*, W. Merkt*, V. Ivan, I. Havoutis, S. Vijayakumar. 'Sparsity-Inducing Optimal Control via Differential Dynamic Programming'. In *IEEE International Conference on Robotics and Automation (**ICRA**)*, Xi'an, China, 2021. (Chapter 6)
- T. Dinev, C. Mastalli, V. Ivan, S. Tonneau, S. Vijayakumar. 'A Versatile Co-Design Approach For Dynamic Legged Robots'. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (**IROS**)*, Kyoto, Japan, 2022. (Chapter 7)

DECLARATION

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Edinburgh, United Kingdom, 2023

Traiko Dinev
15th September 2023

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AD	automatic differentiation
ADMM	alternating direction method of multipliers
DDP	differential dynamic programming
DoF	degrees of freedom
QP	quadratic programming
LQR	linear-quadratic regulator
iLQR	iterative linear-quadratic regulator
MPC	model predictive control
NLP	nonlinear programming
OC	optimal control
w.r.t.	with respect to
CMA-ES	covariance matrix adaptation evolutionary strategy
DOC	differentiable optimal control
AD	automatic differentiation
SA	sensitivity analysis
URDF	unified robotics description format

1 INTRODUCTION

Modern robots are general-purpose devices that are capable of performing a variety of tasks, such as picking and placing objects, precise manipulation of their environment or walking and navigating challenging terrain. In Figure 1 we see examples of different robots across environments. Whether it is a humanoid or a robotic arm picking and placing objects or a quadruped walking, we are faced with the challenge of building and controlling efficient devices that often have to work in multiple domains or navigate multiple terrains. Our aim is to build accurate devices that can reliably execute their tasks repeatably.

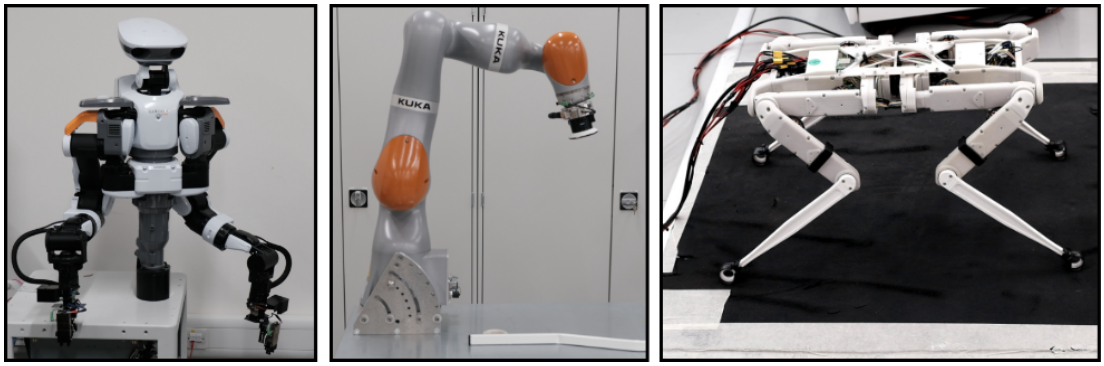


Figure 1: Examples of different modern robots. The NEXTAGE robot (left), the KUKA robotic arm (middle) and SOLO quadruped robot (right).

Indeed of the main challenges of robotics is representing real-world problems in a way that can be solved by computers. As roboticists, our work in this thesis will focus on ways to best model practical problems so that the tools at our disposal can solve them. Which tools we use and what requirements we have for those tools in terms of computational speed, accuracy and scalability will all be domain-specific.

One such core problem in robotics is designing the robot itself. We design robots for a specific set of tasks in mind – for instance a quadruped needs to traverse a variety of different terrains or a robotic arm in a factory needs to assemble hundreds of different components. Another core problem is computing commands to send to the robot so that it executes a task. Traditionally these have been two separate problems. Experts in mechanical and electrical engineering design robots with some objective in mind, for instance cost of manufacturing or making the robot light so that it can be more agile. These objectives are often heuristics that come from the experience of the design engineer. After we design and build the robot, we plan the motions and we build controllers so that the robot can execute those motion plans. These motion plans are based on the physical robot design and thus what the robot can execute is constrained by how it is designed. For instance, in Figure 2 we illustrate two robot designs. The one on the left has a larger and heavier base thus reducing its capacity for carrying a payload, as it has to carry both the weight of its base and the payload. The one on the right has a smaller base (both designs are otherwise the same), so it has a bigger

payload capacity. Although this example is simple, it illustrates that a small changes in design can affect the robot's capabilities.

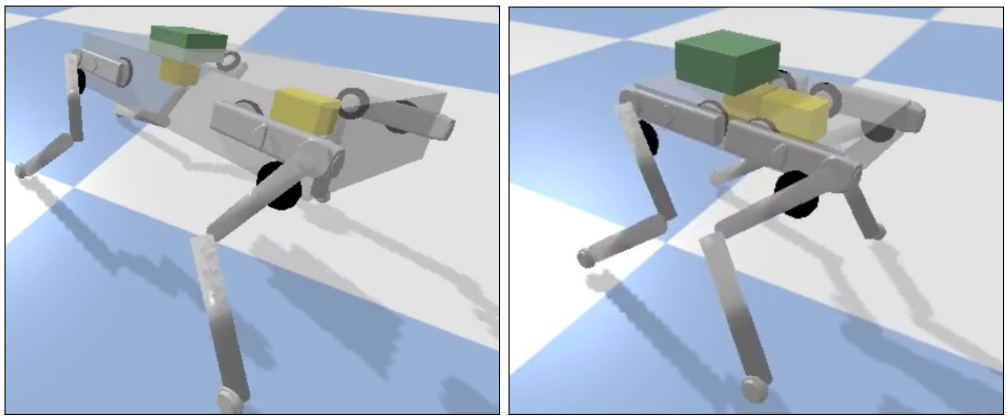


Figure 2: An illustration of two designs of a quadruped robot with a payload. The robot on the right has a smaller base and thus is lighter, which can enable it carry a heavier payload than the robot to the left, which stumbles due to the heavy weight.

While specialized robots that are made for a single task are indeed useful, the challenge is to create multi-purpose robots that can execute not just one, but dozens, if not hundreds of different tasks. If we can do that, we do not need to manufacture specialized hardware, which can be expensive. For instance a humanoid robot, just like a human, should be able to do things like carry different objects, use instruments with precision or open and close doors. This was tested during the DARPA Robotics Challenge [1], where robots had to solve various disaster response-related scenarios, such as operating vehicles and using drills to create escape routes. This motivates us to focus on legged robots (humanoids and quadrupeds). Quadrupeds specifically are made to navigate various terrains, jump, trot and walk on flat ground and over obstacles. Quadrupeds were also a key part of the MEMMO project [18] where they navigated industrial sites with various obstacles and inspection tasks. Considering all of these tasks together while building the robot is outside of the ability of any human. If an engineer could evaluate the robot's performance and use computational tools to quickly find better designs for different tasks, this can aid the design process. The designer can design with the motions the robot will execute in mind and can use these tools as a guide to what design is best for each task.

In the computational-based design paradigm we have to define what part of the process is automated and what part is left to the engineer or designer. In this thesis we let the designer specify the morphology of the robot – this involves for instance how many legs or hands (joints) the robot has, how they connect and how they move in relation to each other. We thus let the designer choose in essence the "type" of robot. We will address the problem of what the best design is in terms of the physical properties of the robot – its limb lengths, their attachment points and their masses and inertias – once the "type" of robot is specified. These properties will directly influence the ability of the robot to execute motions and also its efficiency while doing so. In other words we will be optimizing the continuous parameters that define the design and let the designer choose the discrete ones.

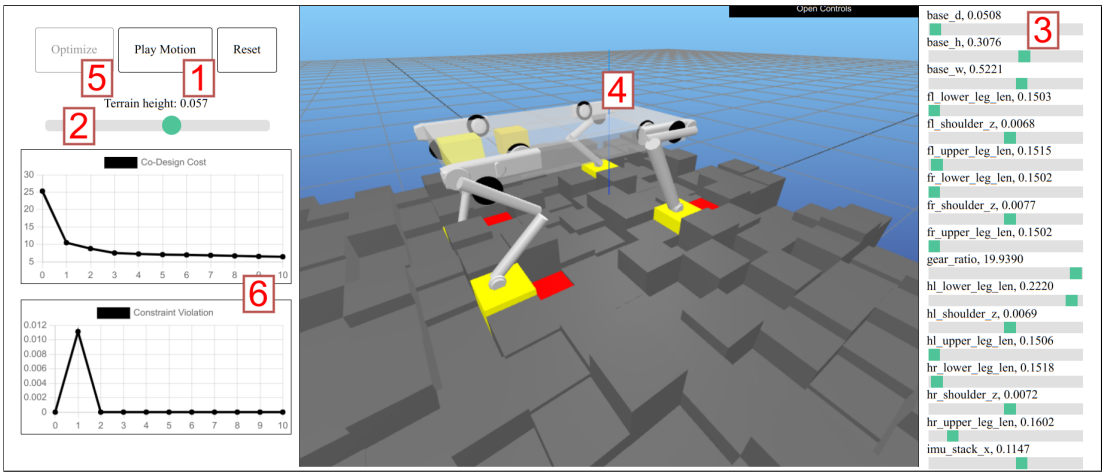


Figure 3: An interactive co-design application that enables the user to optimize designs over different terrains. A video demonstration is available at https://youtu.be/k6vDrjwmsHs.

If we solve this problem at *interactive speeds* (in the order of seconds to a minute at the slowest) , we will be able to provide engineers with tools that evaluate and optimize a given design type over many different scenarios. Robot design engineers will be able to quickly prototype different configurations and find better parameters for them. This process needs to be quick enough for prototyping and designers should be able to stop optimizing when they are satisfied without having to wait a long time for the algorithm to finish optimizing. So long as the algorithms give a large-enough improvement in key metrics and are fast, this can guide the design process.

To demonstrate our long-term vision, we developed an interactive application that showcases what a fast co-design algorithm enables for a single task and fixed gait parameters. This is described in detail in Chapter 3 and we also include a screenshot of it here in Figure 3 to illustrate our goal. A video demonstration is also available at https://youtu.be/k6vDrjwmsHs. This tool enables the user to change the task (terrain), initial robot design (on the right) and optimize the design and playback the motion for a given task and design. We envision such a tool to be integrated within computer-aided design software in the future, so that it can evaluate a design across tasks and environments and guide the design process.

In this thesis we will take steps towards this long-term goal. We will focus firstly on one task, as if we can not optimize designs quickly for one task, we will not be able to do it for more than one. We will also focus only on continuous design variables as well as predefined gait parameters. With this in mind, our goal is to develop fast and scalable (in terms of design parameters) algorithm for co-design and motion optimization.

Taking a step back, in order to develop co-design algorithms we need to set up the problem mathematically. We first need to define what a best design means. In optimization, we refer to this property as optimality. We can define optimality for both motions and designs. Optimal motions are defined via a cost function. This cost function is task-dependent and can include both terms on efficiency (e.g. energy) and task completion. In this thesis will will use the optimal control (OC) approach for generating motions. OC is a popular model-based approach for planning dynamic mo-

tions in robotics (e.g. [25, 27, 60]). It involves setting up a mathematical minimization program that computes optimal motions – this program minimizes the cost function for the task. A *Motion Planner* then is an algorithm that solves the OC problem for a given task and given robot design, as pictured here in Figure 4.

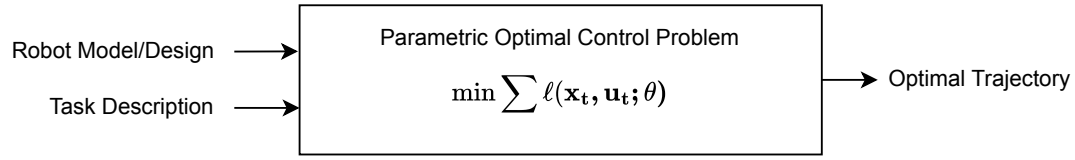


Figure 4: A parametric optimal control problem. We specify the robot model/robot design and the task. The optimal control problem solves a mathematical minimization problem. Here $\boldsymbol{x}, \boldsymbol{u}$ represent states (e.g. positions and velocities) and controls (e.g. forces and torques) of the system, $\ell(\boldsymbol{x}, \boldsymbol{u}; \boldsymbol{\theta})$ are the parametric cost functions, parameterized by a vector $\boldsymbol{\theta}$.

Similarly, when optimizing designs we can also think in the framework of mathematical optimization. We first need to define a cost function that defines optimality. For designs this cost function is made up of terms on the motion, such as how efficient it is – for instance the total energy of the motion. In addition, we might also have cost functions on the design of the robot as well – for instance some designs might be too expensive or difficult to manufacture. These criteria will need to be included in the cost as well. Finally, we necessarily will need design constraints. A robot is a physical structure and as such its parts should not intersect with each other – this will introduce collision constraints to our optimization. Additionally, we might not be able to manufacture for instance arbitrarily long or thin parts – this will introduce bounds on the lengths or masses of different parts.

The motion planning, design costs and design constraints and bounds need to all be considered together by co-design algorithms, which makes the problem challenging. In addition, motion planning will introduce a set of constraints and cost terms of its own. The motion planning constraints will involve the constraints that ensure the physical feasibility of the motion – those are known as dynamics constraints. Motion planning also has a set of cost terms that define the task and optimality as discussed. All of these need to be considered together in order to optimize for both at the same time. As we discuss next, the complexity of this problem can be addressed in different ways, each making a compromise in terms of either speed of computation or what parts of the full problem it solves.

1.1 EXISTING APPROACHES

There are multiple strategies for solving co-design problems. In the current literature we will distinguish between two broad classes of co-design algorithms – sampling-based and gradient-based.

Sampling-based co-design algorithms rely on sampling many candidate designs and iteratively finding better candidates. They involve calling an underlying *Motion Planner* to generate the motion for a given candidate design and then evaluate it based on the *optimality criterion* for the design (Figure 5, left). A popular algorithm for sampling-based co-design is covariance matrix adaptation evolutionary strategy (CMA-ES) [22, 37,

91]. . There is empirical evidence that given a large enough population size, standard deviation and time to converge, evolutionary approaches and CMA-ES can find global optima [39, 41]. They can also optimize discrete and integer variables. However, there are two major drawbacks to these approaches. Firstly, they have slow computational times and poor scalability due to the many samples needed to find and optimum. This is also known as the curse of dimensionality [39, 70], where the number of samples needed to explore the design space grows exponentially with the number of dimensions of the design. Secondly, they lack support for hard linear and nonlinear design constraints in the upper level and only support bounds on the design variables.

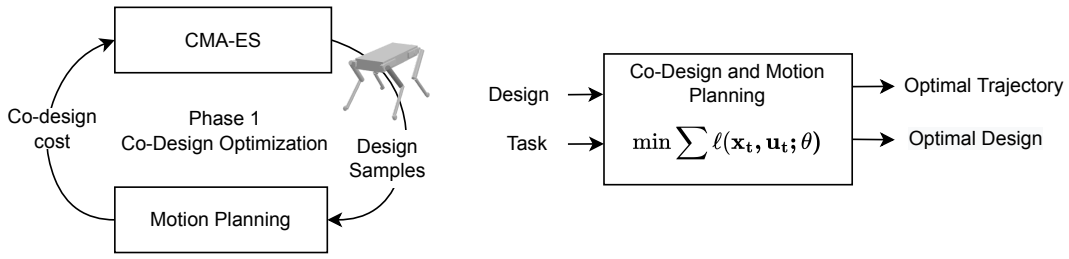


Figure 5: An illustration of sampling-based co-design algorithms via CMA-ES (left) and simultaneous gradient-based co-design (right). Sampling-based approaches sample designs and evaluate each by calling the motion planner. Simultaneous gradient-based approaches formulate a single nonlinear program that optimizes design and motion together.

Another broad category of co-design algorithms is what we call *gradient-based* co-design. These algorithms use in some way the derivative (or gradient) relationship between motion and design.

One approach is to formulate a single nonlinear program that optimizes both design and motions [64, 84]. We refer to this as *simultaneous gradient-based* co-design (Figure 5, right). The benefit of this approach is that if it converges, we can guarantee local optimality since gradient information gives us that information. The main drawbacks are the complexity of the resulting nonlinear program that results in slow computational speed and potentially the algorithm not even converging to a feasible solution [84].

1.2 APPROACH AND PROBLEM STATEMENT

To address the limitations of sampling-based and simultaneous gradient-based approaches while still using gradient information where available, we can directly differentiate the *Motion Planner*. Specifically, we will adopt a bi-level gradient-based optimization approach to co-design. We argue that there is a natural way to split the optimization in two levels – (i) an upper level that optimizes the design and (ii) a lower level that optimizes the motion. Our hypothesis is that we can include the full complexity of design constraints with such a bi-level approach and that this method can solve the problem successfully (converge to a feasible design with a lower cost). Furthermore, we aim to use the full constrained motion planning problem in the lower-level and compute its derivative. This constrained motion planning is important for the physics feasibility of the motion. We further set out to obtain a computational speed and scalability benefit with respect to (w.r.t.) existing approaches, which we

believe is achievable by leveraging the fast computation speed of specialized motion planners.

By using a motion planner in the lower level, we ensure all motion-related constraints are satisfied at every iteration. This is in contrast to simultaneous methods, which do not necessarily produce dynamically feasible motions at every iteration. As such we hypothesize that the co-design process is more stable[1] for bi-level methods in the sense that the motion is always optimal thus simplifying the entire optimization. If we can show these convergence properties, then the main challenge that remains is how to compute the gradients of constrained optimal control problems efficiently.

Finally, we have so-far approached motion planning as a solved problem and built co-design algorithms on top of it. In fact there are still relevant problems in the motion planning layer of co-design that need to be addressed. We discuss two such problems that arise in our co-design formulation. The first is the optimization of contact locations and timings, which in the first part of the thesis we treat as constants defined by the user (e.g. in Chapter 3 and Chapter 5). The second is that the derivatives of system dynamics with respect to (w.r.t.) design variables that are needed for our approach are not always available analytically via rigid body dynamics libraries (e.g. Chapter 4).

In our co-design approach we assumed fixed, pre-specified contact locations and timings. Ideally we would like the motion planner to choose the contacts instead. We solve a related, but simpler problem, which is the control of satellite thrusters, which are similar to robot feet but do not have the constraint of having to be in contact with the ground to exert force on the robot (e.g. [87, Chapter 5]). We introduce a sparse, L_1 cost on control inputs and implement optimization via differential dynamic programming (DDP)-style solvers. We use full rigid-body dynamics and achieved bang-bang control via optimization, which is a difficult problem due to the discrete switching nature of inputs (choice of thrusters to fire). This approach in the context of co-design lets the user specify the task by specifying a more high-level goal (in the case of satellites a goal state and a reference center-of-mass trajectory) and not low-level details such as thruster (contact) timings and locations.

The second problem is the modeling and control of a highly dynamic system, for which analytic derivatives of dynamics w.r.t. design are not available or too expensive to compute. We solve the motion planning and control problem for a wheel-legged robot, which in addition to being able to move like a car with wheels can also jump with a prismatic joint. This is a complex system to control as it can not be statically balanced like a quadruped and thus needs to always be actively balanced. We show that we can solve the motion planning problem using a template model that captures the necessary dynamic interactions between wheels and base. This model can be used to analytically compute derivatives w.r.t. all parameters.

1 We define stable as fewer (or none) of the instances of the optimization diverging across initial parameters, hyper-parameters, robots, or environments.

We finish by summarizing our contributions by chapter, which also gives an outline of the rest of this thesis. Figure 6 shows a diagram of the thesis. The remainder of this thesis is organized in three parts. *Part 1* contains our work in co-design that combines motion planning and design optimization. *Part 2* contains additional work done in motion planning within the same optimal control (OC) framework. Finally, *Part 3* summarizes the thesis and discusses limitations and future work.

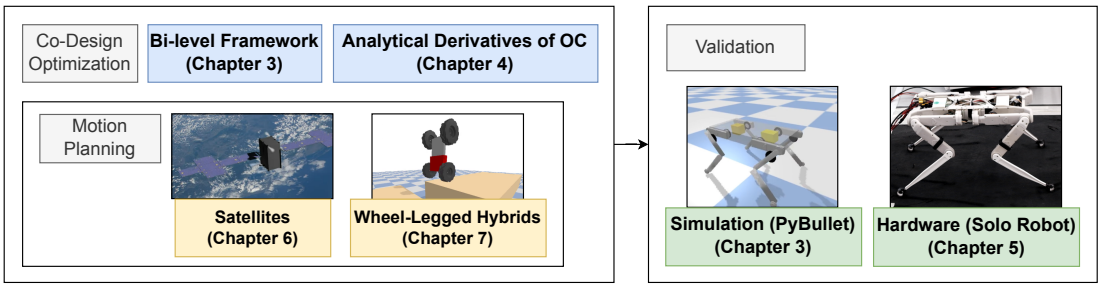


Figure 6: Thesis structure diagram.

Chapter 2 provides a focused review of the current literature on co-design. We discuss benefits and drawbacks of existing approaches and how they compare to our bi-level approach for co-design.

Part 1: Co-Design

In Chapter 3 we develop our bi-level optimization approach for robot co-design. This includes a modular co-design algorithm that differentiates a motion planner (using finite differences) and handles arbitrary co-design constraints and metrics in the upper level. We present a complete co-design framework and interactive application for quadruped robots and dynamic locomotion maneuvers using the motion planner CROCODDYL [58] and validate in simulation. This partially addresses our hypothesis – we demonstrate that the bi-level approach is faster than the covariance matrix adaptation evolutionary strategy (CMA-ES) algorithm (between 1.8 and 8.4 times faster) and can solve practical problems at interactive speeds, albeit for tasks with predefined contact locations and timings.

In Chapter 4 we develop an approach for analytically differentiating optimal control problems via differential dynamic programming (DDP), which is a popular second-order approach to solving OC problems [51, 67]. We verify the numerical accuracy of our approach on the pendulum and double pendulum systems and scale our approach to a co-design problem on the Kinova arm [14] using the solver CROCODDYL [58]. We further compare our approach to the simultaneous co-design approach and show significant improvement in computational time (54 times improvement). This further strengthens our hypothesis by demonstrating the computational benefit of bi-level optimization for co-design.

In Chapter 5 we experimentally validate our approach on real hardware on the Solo robot. We verify that we can execute motions for both the nominal Solo

design and the optimized design we manufacture. We further show an improvement in co-design cost (namely peak torque), though that improvement is less than in simulation due to model mismatch for both design and actuator capability. We analyze and discuss these sim-to-real issues and discuss how they can be addressed within our framework.

Part 2: Motion Planning

These final two chapters contribute to co-design as they enable motion planning in different environments and for different robots, which is part of the motion planning step in co-design.

In Chapter 6 we introduce sparsity-inducing regularization terms in differential dynamic programming (DDP)-type solvers and compare different strategies for sparsity-inducing regularization, namely SmoothL1, Huber, and Pseudo-Huber loss, in terms of their convergence and numerical stability. We present a heuristic for tuning the open parameters of these costs. Sparsity-inducing costs have applications in satellite thruster control and automatic joint selection for low-dimensional tasks, which we validate on real hardware on the Valkyrie robot. We use L_1 costs together with full rigid-body dynamics on the satellite system, which allows the planner to achieve bang-bang control inputs, which is a difficult problem due to the discrete switching nature of the thrusters. This problem relates to contact planning, as thrusters are similarly discrete as contacts for walking robots without the placement constraint of having to be in contact with the ground.

In Chapter 7 we propose the Variable-Length Wheeled Inverted Pendulum (VL-WIP) template model for planning and control of wheeled-legged systems. We implement a motion planner using the VL-WIP model and the direct transcription method which we integrate into a Model Predictive Controller (MPC). We show that using the VL-WIP model we are able to plan and execute dynamic motions such as jumping, which are only achievable by combining the action of wheels and base. We further validate our controller in simulation and evaluate its robustness on rough terrain locomotion and in the presence of sensor noise. This extends our motion-planning capabilities to the new and challenging system of wheel-legged hybrids.

Part 3: Conclusion and Future Work

Finally, Chapter 8 summarizes the thesis and provides a critical review of our approaches, including limitations and future work.

2 BACKGROUND: OPTIMAL CONTROL AND CO-DESIGN

In this chapter we review the relevant literature on co-design with a brief overview of motion planning and how it fits within co-design. Since co-design itself is at the crossroads of different fields, we give a very focused review of the literature that primarily compares algorithmic strategies for co-design and we only give details about motion planning where necessary for co-design. Specifically, we require constrained motion planning for full physical feasibility of the motions, an ability to define design constraints for feasibility of the design itself and a fully automatic approach for co-design that does not require user input while optimizing. With these criteria in mind, we begin by discussing the motion planning and build towards co-design methods and how they fit within these criteria.

Planning (dynamic) motions for robots is necessary for a wide range of applications, including co-design. *Dynamic* here refers to the inclusion of the robot's dynamics, which are expressed as its equations of motion governed by Newtonian mechanics. There are other forms of motion planning – purely *kinematic* motion planning only considers the geometry of the robot and plans for positions and velocities only. *Kino-dynamic* motion planning also plans for accelerations but does not consider the full nonlinear dynamics of the robot Dynamic motion planning is necessary for tasks where the dynamics can not be ignored. An example is jumping, where the ballistic trajectory of the robot when not in contact with the ground is dependent on the mass of the robot and can also be influenced by the actions of the legs while in the air. For these tasks we need to consider the dynamics of the robot in our planning algorithms. Dynamic motion planning as such is also the most *realistic* motion planning in terms of the simulation-to-reality gap. However, it is also the most complex, as we consider the full interaction with the environment and the full robot physics.

In this chapter we formulate the parametric optimal control (OC) problem mathematically. This is the problem that is solved by the motion planner. We then build on that formulation to formulate the co-design problem as a mathematical optimization and discuss different strategies for solving it, including the bi-level formulation that we will use for the rest of the thesis and its benefits and drawbacks.

2.1 MOTION PLANNING

We begin our discussion by formulating the optimal control (OC) problem. It is defined for a dynamic system, such as a robot, which is defined by a vector of states $\boldsymbol{x}$ and controls $\boldsymbol{u}$, where:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix} \tag{1}$$

is the state of the system, described by its generalized coordinates $\boldsymbol{q}$ and their velocities $\dot{\boldsymbol{q}}$. For a fixed-base system like the Kinova arm, $\boldsymbol{q}$ are the angles of each joint and $\dot{\boldsymbol{q}}$ are their angular velocities. For a floating-base system like the quadruped or wheel-legged

robot, we also need to include the position and orientation of the base and its linear and angular velocity. The controls $\boldsymbol{u}$ are the forces that are applied to the robot, for instance the motor torques on the arm or on each motor on the legs of a quadruped. The states and controls define how the system evolves in time:

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta}) \tag{2}$$

where the dynamics function $\boldsymbol{f}(\cdot)$ evolves the state from time t to time $t+1$. In this thesis we will consider discretized dynamics, which involve integrating the continuous physics of the system defined by the Newton-Euler differential equations by some time interval Δt. We also introduce the vector $\boldsymbol{\theta}$, which for the rest of the thesis will denote parameterized dynamics and costs. $\boldsymbol{\theta}$ may include robot design variables, such as link length and attachments, as well as cost function weights (described next). For instance, in Figure 7 we show an example parameterized design for a quadruped, where the parameters include the payload location as well as the link lengths and actuator masses.

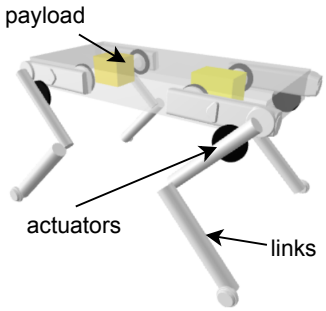


Figure 7: An example quadruped design parameterization.

In order to define the OC problem, we specify a set of cost functions and a set of constraints that define the task. The generic formulation is a mathematical minimization problem:

$$\begin{aligned}
\min_{\boldsymbol{X},\boldsymbol{U}} J_{\text{OC}}(\boldsymbol{X}, \boldsymbol{U}; \boldsymbol{\theta}) &= h(\boldsymbol{x}_T; \boldsymbol{\theta}) + \sum_{t=1}^{T-1} \ell(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta}) && (3)\\
\text{s.t.} \quad \boldsymbol{x}_{t+1} &= \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta}) && \text{(Dynamics Constraints)}\\
\boldsymbol{x}_1 &= \bar{\boldsymbol{x}}_1 && \text{(Starting State)}\\
\boldsymbol{g}_{\text{OC}}(\boldsymbol{X}, \boldsymbol{U}; \boldsymbol{\theta}) &= 0 && \text{(Additional equality constraints)}\\
\boldsymbol{h}_{\text{OC}}(\boldsymbol{X}, \boldsymbol{U}; \boldsymbol{\theta}) &\leq 0. && \text{(Additional inequality constraints)}
\end{aligned}$$

This OC problem solves for a trajectory $\boldsymbol{X}, \boldsymbol{U}$, which consists of two sets of state and control pairs in time: $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$ and $\boldsymbol{U} = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{T-1}\}$. This is a generic formulation that consists of several important ingredients:

- The OC cost, $J_{\text{OC}}(\boldsymbol{X}, \boldsymbol{U}; \boldsymbol{\theta})$. This cost is parameterized by $\boldsymbol{\theta}$ and is expressed as sum of costs on each state-control pair in time ($\ell(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta})$) plus a cost for the terminal state ($h(\boldsymbol{x}_T, \boldsymbol{\theta})$). The cost can for instance penalize distance from a goal state (in $h(\boldsymbol{x}_T, \boldsymbol{\theta})$) and can include costs that penalize large control inputs or velocities or undesired configurations. These costs can be parameterized by the vector $\boldsymbol{\theta}$.

- The system dynamics constraint $\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta})$. This ensures the resulting trajectory satisfies the laws of physics. It is also parameterized by $\boldsymbol{\theta}$.
- The starting state constraint $\boldsymbol{x}_1 = \bar{\boldsymbol{x}}_1$. We assume we know the starting position of our robot.
- Any additional equality ($\boldsymbol{g}_{\text{OC}}(\boldsymbol{X}, \boldsymbol{U}; \boldsymbol{\theta})$) and inequality ($\boldsymbol{h}_{\text{OC}}(\boldsymbol{X}, \boldsymbol{U}; \boldsymbol{\theta})$) constraints that help define the task. For instance, these can include contact or no-slip constraints.

There are different strategies for solving the same optimal control (OC) problem, depending on the formulation. We discuss one such approach, differential dynamic programming (DDP) ([62, 86]), in Chapter 4 and another approach, direct transcription, in Chapter 7. DDP is known as a shooting solver, that is it unrolls the dynamics constraint implicitly and uses cost functions for specifying a goal terminal state. As such, it does not support additional equality or inequality constraints. Direct transcription is an alternative (e.g. [27] as well as Chapter 7), where the nonlinear program is directly transcribed and solved by a nonlinear solver, such as IPOPT [90] or KNITRO [13]. This approach is preferred when there is a need to handle other hard constraints in addition to the dynamics. In the case of contact discontinuities where derivatives are not available, sampling-based approaches can be used. Sampled DDP [76] is one such approach, where the value function is approximated using covariance matrix adaptation evolutionary strategy (CMA-ES). In our formulation we assume we can compute the derivatives of dynamics needed by the solver including through contacts (e.g. Chapter 3).

Importantly here we want to distinguish between the motion planner/solver and the problem formulation. The problem formulation we have given is generic. Depending on the formulation certain motion planners will be more appropriate than others, as discussed above.

In the next section we discuss different co-design formulations, which build on the optimal control problem. Again, we make a distinction between formulations and algorithms that solve the co-design problem. Which algorithm is appropriate will depend now not only on the co-design formulation, but also on the motion planning formulation.

2.2 CO-DESIGN

Co-design is not a recent idea. More than 20 years ago, Li *et. al.* [54] optimized the design, control parameters and motions of pivot four-bar linkages. They used a simpler physics (or dynamics) model of the system to make the problem tractable and the design was optimized in multiple iterations, where in some the dynamics (physics) and motions were considered and in others, they were not. Still manual expert input was required for some of the stages and the robot's model was simplified.

We aim to optimize the robot design fully automatically, without expert input guidance during the optimization. The user specifies the task, the morphology of the robot (what the links are and how links are attached to each other), the optimality criterion (e.g. energy of the motion) and the design constraints. We are also interested

in using the full robot dynamics so that we can capture the full physics of its motion and how the design influences it.

For co-design specifically, we will distinguish between the general vector of parameters $\boldsymbol{\theta}$, which may include cost and constraint parameterization terms and the design vector $\boldsymbol{\rho}$ which strictly includes only parameters relating to the design of the robot, where $\boldsymbol{\rho}$ is in general part of $\boldsymbol{\theta}$. In the rest of this section, we will discuss approaches that optimize $\boldsymbol{\rho}$ only.

2.2.1 *Simultaneous Gradient-Based Co-Design*

The simultaneous formulation is the most straightforward, as we simply embed the motion planning and co-design problem in a single nonlinear program. We introduce a new cost term $J_{\text{CD}}(\boldsymbol{\rho}, \boldsymbol{X}, \boldsymbol{U})$, the co-design cost, which can be different than the optimal control cost and is defined on the motion and the design. We also introduce additional equality and inequality constraints on the design: $\boldsymbol{g}_{\text{CD}}(\boldsymbol{\rho}) = 0$ and $\boldsymbol{h}_{\text{CD}}(\boldsymbol{\rho}) \leq 0$. Finally, we need to have mixing weights w_1 and w_2 for the two costs. This leads to the following generic formulation:

$$
\begin{aligned}
\min_{X,U,\boldsymbol{\rho}} \; & w_1 J_{\text{CD}}(\boldsymbol{\rho}, \boldsymbol{X}, \boldsymbol{U}) + w_2 J_{\text{OC}}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{\rho}) && (4)\\
& = w_1 J_{\text{CD}}(\boldsymbol{\rho}, \boldsymbol{X}, \boldsymbol{U}) + w_2 h(\boldsymbol{x}_T, \boldsymbol{\rho}) + w_2 \sum_{t=1}^{T-1} \ell(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\rho}) \\
\text{s.t.} \quad & \boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\rho}) && \text{(Dynamics Constraints)} \\
& \boldsymbol{x}_1 = \bar{\boldsymbol{x}}_1 && \text{(Starting State)} \\
& \boldsymbol{g}_{\text{OC}}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{\rho}) = 0 && \text{(Additional equality constraints)} \\
& \boldsymbol{h}_{\text{OC}}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{\rho}) \leq 0. && \text{(Additional inequality constraints)} \\
& \boldsymbol{g}_{\text{CD}}(\boldsymbol{\rho}) = 0 && \text{(Design equality constraints)} \\
& \boldsymbol{h}_{\text{CD}}(\boldsymbol{\rho}) \leq 0. && \text{(Design inequality constraints)}
\end{aligned}
$$

This formulation has been used for co-design in the literature. For instance, Spielberg *et. al.* [84] used this formulation and solved it via the nonlinear programming solver SNOPT [32]. Mombaur *et. al.* [64] used this formulation to optimize the design of a biped for stable running, Buondonno *et. al.* [11] optimized actuator properties of compliant walkers and Bravo-Palacios *et. al.* [9] used a single optimization for design and motion, embedding it into a stochastic optimization over multiple tasks.

The benefits of this approach is that we optimize for both design and motion in a single-level optimization, thus allowing the solver to exploit the relationship between the two directly. However, this approach has the following drawbacks:

- When we optimize for robot motion, we often need to include cost functions that set-up the task. These may include costs on extreme velocity profiles or regularization terms that guarantee some desired behavior. The *optimality criterion* for co-design is also a cost term and by mixing the two we might not obtain either the desired motions or an optimal design. Thus we lack the flexibility of having different objectives – (i) to solve the motion task and (ii) to find the optimal design.

- The resulting nonlinear program might be simply too complex, as the robot design has an effect on the motion over the entire planning horizon, that is it influences all other decision variables in the optimization. This can lead to the program not converging or having long computation times. For instance, Spielberg *et. al.* [84] needed to solve the problem twice, once to find a feasible motion with no cost function and then to optimize with the cost function, starting from the feasible motion as the initial guess.
- From a practical point of view, we want to use *Motion Planners* that are as close to the ones we would deploy on the physical robot so that the design improvements are realistic and consistent. Including design variables in the nonlinear program by definition changes the motion planning too.

Finally, it is also in theory possible to solve this single-level problem using a variant of DDP called Parameterized DDP [72]. The authors in [72] derived an algorithm for including $\boldsymbol{\rho}$ in a standard DDP formulation and applied their approach to parameter estimation and switching-time optimization (but not co-design). Applied to co-design, this approach in theory can increase the speed of the solver as compared to a standard non-linear program due to it exploiting the structure of the problem (as standard DDP does). However, it most notably lacks linear and nonlinear design constraints (one key co-design requirement) and such speed improvements have not been empirically reported in [72].

2.2.2 *Bi-Level Co-Design*

Instead of a single optimization, we can formulate the co-design problem as a bi-level optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{\rho},\boldsymbol{X},\boldsymbol{U}} \; J_{\text{CD}}(\boldsymbol{\rho},\boldsymbol{X},\boldsymbol{U}) && (5)\\ &\text{s.t. } \boldsymbol{X},\boldsymbol{U} = \text{MP}(\boldsymbol{\rho},\textsc{Task}), && \text{(motion planning)}\\ &\quad \underline{\boldsymbol{\rho}} \leq \boldsymbol{\rho} \leq \bar{\boldsymbol{\rho}}, && \text{(design bounds)} \end{aligned}$$

where in the upper-level the design is optimized and in the lower level the motion is computed by a motion planner $\text{MP}(\boldsymbol{\rho},\textsc{Task})$ that is defined for a given robot design $\boldsymbol{\rho}$ and task. The motion planner solves an optimal control problem in order to find the optimal motion. This formulation only includes design bounds and not general equality and inequality constraints. We first discuss approaches that solve this particular bi-level formulation.

Bi-level optimization has been used in a variety of disciplines, such as economics, transportation or engineering [83]. One way of solving it is using evolutionary or genetic algorithms, where in the upper level we sample designs $\boldsymbol{\rho}$ and evaluate each by running the motion planner $\text{MP}(\boldsymbol{\rho},\textsc{Task})$. We refer to this as sampling-based co-design.

A popular tool for sampling-based co-design is the CMA-ES algorithm. CMA-ES is a genetic algorithm – it evolves a population of designs iteratively by assuming a Gaussian prior on candidate design parameters and estimating a covariance matrix needed for the following sampling steps. There are many examples of the use of

CMA-ES in co-design – Wampler *et. al.* [91] used a variant of CMA-ES to co-design creatures in simulation, Digumarti *et. al.* [22] co-designed the legs of the quadruped StarlETH to optimize its running speed, Ha *et. al.* [37] used CMA-ES to optimize design and swing trajectories of planar legged robots. Even recently, Chadwick *et. al.* [16] optimized the legs of quadrupeds and bipeds over uneven terrain for different user-defined co-design metrics and Fadini *et. al.* [24] optimized the actuator properties of a monoped using CMA-ES.

CMA-ES is popular because it can optimize in theory any function that has an optimum – whether it is smooth or not. It additionally has few parameters to tune (namely population size and initial standard deviation). There are a few major drawbacks to CMA-ES and sampling-based genetic algorithms:

- They do not support hard constraints on design parameters, such as nonlinear collision constraints and only support bounds. This can be worked around (as we show in Chapter 3) by using soft constraints, which are cost terms added to the objective function. These increase greatly the number of parameters that require cost tuning and we have no guarantee they will be satisfied. From a designer standpoint, we want to ability to specify arbitrary constraints on the design (such as no collision between design components or weight limits).
- While There is empirical evidence that CMA-ES has global convergence properties with increased population size and time to converge [39, 41]. We will show that in practice locally optimal gradient-based algorithms can converge to similar designs as CMA-ES. Moreover, CMA-ES scales exponentially with respect to (w.r.t.) the number of design parameters (*i.e.* decision variables), due to the curse of dimensionality [40, 70]. These properties limit its application to a reduced number of design parameters and constraints, which in turn limits its scalability.

To address the limitations of sampling-based and simultaneous gradient-based approaches while still using gradient information where available, we can directly differentiate the *Motion Planner*. This is the approach we take in this thesis.

In the bi-level formulation that CMA-ES solves (5), we omitted upper-level constraints but only included bound constraints (as only those are supported). The full bi-level co-design formulation we aim to solve is in fact:

$$
\begin{aligned}
&\min_{\boldsymbol{\rho},\boldsymbol{X},\boldsymbol{U}} \; J_{\text{CD}}(\boldsymbol{\rho},\boldsymbol{X},\boldsymbol{U}) && (6)\\
&\quad \text{s.t. } \boldsymbol{X},\boldsymbol{U} = \text{MP}(\boldsymbol{\rho},\textsc{Task}), && \text{(motion planning)}\\
&\qquad \underline{\boldsymbol{\rho}} \leq \boldsymbol{\rho} \leq \bar{\boldsymbol{\rho}}, && \text{(design bounds)}\\
&\qquad \boldsymbol{g}_{\text{CD}}(\boldsymbol{\rho}) = 0, && \text{(equality design constraints)}\\
&\qquad \boldsymbol{h}_{\text{CD}}(\boldsymbol{\rho}) \leq 0, && \text{(inequality design constraints)}
\end{aligned}
$$

where now we have design constraints in the upper-level. The approach we will adopt in this thesis is to embed this problem into a nonlinear program in the upper-level and use gradient information from the lower level (the motion planning).

We will first rewrite the optimization moving the lower-level constraint into the cost function of the upper-level, making the constraint on the motion planning implicit:

$$
\begin{aligned}
&\min_{\boldsymbol{\rho}} J_{\text{CD}}(\boldsymbol{\rho}, \text{MP}(\boldsymbol{\rho}, \textsc{Task})) && \text{(co-design metric)} \\
&\text{s.t. } \underline{\boldsymbol{\rho}} \leq \boldsymbol{\rho} \leq \bar{\boldsymbol{\rho}}, && \text{(design bounds)} \\
&\quad \boldsymbol{g}_{\text{CD}}(\boldsymbol{\rho}) = 0, && \text{(equality design constraints)} \\
&\quad \boldsymbol{h}_{\text{CD}}(\boldsymbol{\rho}) \leq 0, && \text{(inequality design constraints)}
\end{aligned} \tag{7}
$$

To solve this problem, we only require the derivative of the cost function and the derivative of the constraints. The constraints are only dependent on $\boldsymbol{\rho}$, hence their derivatives are readily available. Given how we re-wrote this problem, we first propose to compute the cost derivative via finite differences:

$$
\nabla_{\boldsymbol{\rho}_i} J_{\text{CD}} \approx \frac{J_{\text{CD}}(\boldsymbol{\rho}_i^+, \text{MP}(\boldsymbol{\rho}_i^+; \textsc{Task})) - J_{\text{CD}}(\boldsymbol{\rho}, \text{MP}(\boldsymbol{\rho}; \textsc{Task}))}{\epsilon} \tag{8}
$$

where $\boldsymbol{\rho}_i^+$ is the design vector with ϵ added to its i^{th} element and ϵ is a small number. Since the dimensions of $\boldsymbol{\rho}$, $\dim(\boldsymbol{\rho})$ is small compared to that of the motion, we can compute this in parallel. This requires calling the motion planner $\dim(\boldsymbol{\rho})$ times, which at first seems impractical. However, as we will show in Chapter 3, we can obtain a computational advantage using this approach, as modern motion planners are optimized for efficiency in order to be used in online planning on the robots and the dimensions of $\boldsymbol{\rho}$ is not prohibitive for some real-world co-design problems.

Our full approach for co-design using this method is described in detail in Chapter 3. We apply it to the task of co-designing quadrupeds and compare it to the genetic algorithm CMA-ES.

2.3 Differentiating the Optimal Control Problem

Solving the bi-level problem by differentiating the lower-level (the motion planner) is an approach that belongs to the class of *descent approaches* [83] named because of the use of gradient descent. There exist works in the literature that differentiate optimal control problems and use the derivative to optimize the design (though none have the full complexity in terms of constraints in the upper-level).

2.3.1 *Unconstrained Optimization*

We first discuss methods that differentiate the unconstrained OC problem. To do this, we first need to reformulate our motion planning problem and convert all constraints to costs in order to arrive at the unconstrained OC problem. We consider the simplest motion planning problem without additional equality or inequality constraints, though this approach can be generalized to include them. The formulation is:

$$
\begin{aligned}
\boldsymbol{X},\boldsymbol{U} &= \operatorname{argmin}_{\boldsymbol{X},\boldsymbol{U}} J_{\text{TOT}}(\boldsymbol{X},\boldsymbol{U},\boldsymbol{\rho}) = \operatorname{argmin}_{\boldsymbol{X},\boldsymbol{U}} J_{\text{OC}}(\boldsymbol{X},\boldsymbol{U},\boldsymbol{\rho}) + J_{\text{CS}}(\boldsymbol{X},\boldsymbol{U},\boldsymbol{\rho}) \quad (9)\\
&= \operatorname{argmin}_{\boldsymbol{X},\boldsymbol{U}} \big(J_{\text{OC}}(\boldsymbol{X},\boldsymbol{U},\boldsymbol{\rho}) + w_0(\boldsymbol{x}_1 - \bar{\boldsymbol{x}}_1)^\top (\boldsymbol{x}_1 - \bar{\boldsymbol{x}}_1) \big)\\
&+ \sum_{t=1}^{T-1} w_t(\boldsymbol{x}_{t+1} - \boldsymbol{f}(\boldsymbol{x}_t,\boldsymbol{u}_t,\boldsymbol{\rho}))^\top (\boldsymbol{x}_{t+1} - \boldsymbol{f}(\boldsymbol{x}_t,\boldsymbol{u}_t,\boldsymbol{\rho}))
\end{aligned}
$$

where we introduced a new cost term $J_{\text{CS}}(\boldsymbol{X},\boldsymbol{U},\boldsymbol{\rho})$, the cost on constraints, which converts the equality dynamics constraints into quadratic penalties together with weights to mix them with the optimal control cost. The major drawback of using this formulation is that we can not guarantee the satisfaction of the dynamics constraints, which in turn means the resulting motion plan might not be physically feasible.

Differentiating this problem, however, is straightforward. To do so, we differentiate the optimality conditions of the problem w.r.t. $\boldsymbol{\rho}$. This is known as sensitivity analysis (SA) and is not unique to co-design – for example this technique has been used in machine learning to differentiate parameterized argmin and quadratic programming problems [3, 34]. We write the optimality conditions, expand via the chain rule and rearrange terms to obtain the derivative of the motion $\boldsymbol{m} = \{\boldsymbol{X},\boldsymbol{U}\}$ w.r.t. to $\boldsymbol{\rho}$:

$$
\begin{aligned}
\frac{\mathrm{d}J_{\text{TOT}}}{\mathrm{d}\boldsymbol{m}} &= 0 && \text{Optimality conditions}\\
\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\rho}}\left(\frac{\mathrm{d}J_{\text{TOT}}}{\mathrm{d}\boldsymbol{m}}\right) &= 0 && \text{Differentiate the optimality conditions}\\
\frac{\partial^2 J_{\text{TOT}}}{\partial \boldsymbol{m}^2}\frac{\mathrm{d}\boldsymbol{m}}{\mathrm{d}\boldsymbol{\rho}} + \frac{\partial^2 J_{\text{TOT}}}{\partial \boldsymbol{m}\partial\boldsymbol{\rho}} &= 0 && \text{Expand via the chain rule}\\
\frac{\mathrm{d}\boldsymbol{m}}{\mathrm{d}\boldsymbol{\rho}} &= -\left(\frac{\partial^2 J_{\text{TOT}}}{\partial \boldsymbol{m}^2}\right)^{-1}\frac{\partial^2 J_{\text{TOT}}}{\partial \boldsymbol{m}\partial\boldsymbol{\rho}} && \text{Rearrange terms}
\end{aligned}
$$

There are two broad classes of approaches to using this derivative relationship. In the first we consider this to be a mapping between motion and design. We then change the motion to stay on the manifold of optimal motions when the design is changed according to:

$$
\Delta\boldsymbol{m} = -\left(\frac{\partial^2 J_{\text{TOT}}}{\partial \boldsymbol{m}^2}\right)^{-1}\frac{\partial^2 J_{\text{TOT}}}{\partial \boldsymbol{m}\partial\boldsymbol{\rho}}\Delta\boldsymbol{\rho} \quad (10)
$$

This is the approach taken by Geilinger *et. al.* [29] in their *manual* design optimization mode. The main challenge with this method is that we need some external input that provides $\Delta\boldsymbol{\rho}$ so that we can update the motion to stay on the manifold of optimal motions.

More generally, if we want to automatically optimize the design, we again define the co-design cost function $J_{\text{CD}}(\boldsymbol{X},\boldsymbol{U},\boldsymbol{\rho})$. We can then obtain its derivative using the chain rule:

$$
\frac{\mathrm{d}J_{\text{CD}}}{\mathrm{d}\boldsymbol{\rho}} = \frac{\partial J_{\text{CD}}}{\partial \boldsymbol{m}}\frac{\mathrm{d}\boldsymbol{m}}{\mathrm{d}\boldsymbol{\rho}} + \frac{\partial J_{\text{CD}}}{\partial \boldsymbol{\rho}} \quad (11)
$$

We can then use this derivative in gradient optimization, via for instance gradient descent by taking small steps η in the gradient direction:

$$\boldsymbol{\rho}_{i+1} = \boldsymbol{\rho}_i - \eta \frac{\mathrm{d}J_{\mathrm{CD}}}{\mathrm{d}\boldsymbol{\rho}} \tag{12}$$

This approach is used in the *automatic* optimization of Skaterbots [29] as well as later in Geilinger *et. al.* [30] and Desai *et. al.* [21]. These approaches iterate between (i) optimizing design and (ii) optimizing motion by taking small steps in the gradient direction for both. Desai *et. al.* [21] instead first optimizes for motion until convergence and then optimizes for the design.

The main challenge with these approaches is the use of unconstrained optimal control as mentioned. Another challenge is in the computation of the derivative:

$$\frac{\mathrm{d}\boldsymbol{m}}{\mathrm{d}\boldsymbol{\rho}} = -\left(\frac{\partial^2 J_{\mathrm{TOT}}}{\partial \boldsymbol{m}^2}\right)^{-1} \frac{\partial^2 J_{\mathrm{TOT}}}{\partial \boldsymbol{m} \partial \boldsymbol{\rho}}. \tag{13}$$

This requires the inversion of a matrix that has the dimensions of the motion squared. This is a large and sparse matrix, as the constraints expressed in costs are defined only for sequential states. Desai *et. al.* [21] uses the adjoint method for computing it, however they still require its construction explicitly. We can indeed use this sparsity property to invert this matrix more efficiently. We discuss an approach that does so next for the constrained version of this problem.

2.3.2 *Differentiable Constrained Optimization*

There exist approaches that compute the derivative for the constrained optimization problem. Ha *et. al.* [38] used sensitivity analysis to derive $\frac{\mathrm{d}\boldsymbol{m}}{\mathrm{d}\boldsymbol{\rho}}$. However, they still require the use of finite differences and their approach does not compute explicitly the gradient of the co-design function, but rather requires user input and uses that in order to keep the motion and design on the optimal manifold.

Differentiating mathematical programs (which motion planners are) is a topic that is also relevant in the field of machine learning. For instance, Gould *et. al.* [34] discussed differentiating parameterized mathematical programs with an example in classification. The OptNet [3] neural network uses differentiable quadratic programs, which are constrained mathematical programs with a quadratic objective and linear constraints. The authors in [3] extended this idea to the linear-quadratic regulator (LQR), which is an OC algorithm for linear systems with quadratic costs and more generally to iterative linear-quadratic regulator (iLQR), which is an iterative algorithm that can solve nonlinear OC problems, such as pendulums and cartpoles, which the authors demonstrate their approach on [2]. This is a promising approach that we can apply to co-design and scale to more complex robotic systems too, which we discuss in Chapter 4. Here we describe this approach first and then discuss its limitations.

The authors in [3] differentiate iLQR, which solves the following optimal control problem:

$$\min_{X,U} J_{\text{OC}}(\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{\theta}) = h(\boldsymbol{x}_T, \boldsymbol{\theta}) + \sum_{t=1}^{T-1} \ell(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\theta}) \tag{14}$$
$$\text{s.t.} \quad \boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\theta}) \qquad \text{(Dynamics Constraints)}$$
$$\boldsymbol{x}_1 = \bar{\boldsymbol{x}}_1 \qquad \text{(Starting State)}$$

where the difference between the generic OC problem (3) and the OC problem that iLQR solves is that the iLQR problem only has dynamics constraints and a starting state constraint. Since the approach we discuss next differentiates w.r.t. cost functions parameters too, we now need to use the vector $\boldsymbol{\theta}$ instead of only $\boldsymbol{\rho}$, the design vector.

iLQR iterates between (i) formulating and solving a LQR approximation of the nonlinear problem and (ii) taking small steps in its optimal direction. A LQR problem is a specific type of OC problem that has a quadratic cost and linear dynamics. It can be solved in one iteration if it is convex. We provide more details of the algorithm in Chapter 4.

Here we continue our discussion on differentiating OC problems. Amos *et. al.* [2] differentiates the constrained nonlinear problem using sensitivity analysis on the final LQR approximation when the original problem is solved. To do so, they first write the LQR approximation of the original nonlinear problem:

$$\min_{\boldsymbol{x}_t, \boldsymbol{u}_t} \sum_t \frac{1}{2} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix}^\top \boldsymbol{C}_t \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix} + \boldsymbol{c}_t \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix} \tag{15}$$
$$\text{s.t. } \boldsymbol{x}_{t+1} = \boldsymbol{F}_t^\top \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix} + \bar{\boldsymbol{f}}_t.$$

The matrices $\boldsymbol{F}_t, \boldsymbol{C}_t, \boldsymbol{c}_t, \bar{\boldsymbol{f}}_t$ are obtained by a Taylor expansion of the original nonlinear problem:

$$\boldsymbol{F}_t = \begin{bmatrix} \boldsymbol{f}_{\boldsymbol{x}_t} \\ \boldsymbol{f}_{\boldsymbol{u}_t} \end{bmatrix} \quad \boldsymbol{F}_0 = \boldsymbol{0}, \quad \boldsymbol{C}_t = \begin{bmatrix} \ell_{\boldsymbol{x}\boldsymbol{x}_t} & \ell_{\boldsymbol{x}\boldsymbol{u}_t} \\ \ell_{\boldsymbol{x}\boldsymbol{x}_t} & \ell_{\boldsymbol{u}\boldsymbol{u}_t} \end{bmatrix}, \tag{16}$$
$$\boldsymbol{c}_t = \begin{bmatrix} \ell_{\boldsymbol{x}_t} - \ell_{\boldsymbol{x}\boldsymbol{x}_t}\boldsymbol{x}^r - \ell_{\boldsymbol{x}\boldsymbol{u}_t}\boldsymbol{u}^r \\ \ell_{\boldsymbol{u}_t} - \ell_{\boldsymbol{u}\boldsymbol{u}_t}\boldsymbol{u}^r - \ell_{\boldsymbol{u}\boldsymbol{x}_t}\boldsymbol{x}^r \end{bmatrix},$$
$$\bar{\boldsymbol{f}}_t = \boldsymbol{f}(\boldsymbol{x}_t^r, \boldsymbol{u}_t^r, \boldsymbol{\theta}) - \boldsymbol{f}_{\boldsymbol{x}_t}\boldsymbol{x}_t^r - \boldsymbol{f}_{\boldsymbol{u}_t}\boldsymbol{u}_t^r, \quad \bar{\boldsymbol{f}}_0 = \bar{\boldsymbol{x}}_1,$$

where the subscript notation represents partial derivatives w.r.t. the subscript evaluated at that point (for instance $\ell_{\boldsymbol{x}_t}$ is the derivative of ℓ w.r.t. $\boldsymbol{x}$ evaluated at $\boldsymbol{x}_t$). $\boldsymbol{x}^r$ and $\boldsymbol{u}^r$ are the reference trajectories, which are provided as an initial guess at the first iteration and for each later iteration taken from the best guess so far.

This optimization problem has a set of equations that must be satisfied if the problem is at the optimum. In the bi-level optimization scheme we solve the motion planning problem at every iteration so this is satisfied. The optimality conditions are also known as the Karush-Kuhn-Tucker conditions. We next describe the how to arrive at these KKT conditions.

Consider an equality constrained minimization problem of the form:

$$\begin{aligned} &\min_{\boldsymbol{z}} J(\boldsymbol{z}) \\ &\text{s.t. } \boldsymbol{c}(\boldsymbol{z}) = 0 \end{aligned} \tag{17}$$

where we used $\boldsymbol{z}$ as the optimization variable, $J(\boldsymbol{z})$ as the cost and $\boldsymbol{c}(\boldsymbol{z})$ as the constraint. Then the optimality (KKT) conditions are (e.g. [5, Chapter 1]):

$$\begin{aligned} &\nabla_{\boldsymbol{z}} \mathcal{L} = \mathbf{0} \\ &\nabla_{\boldsymbol{\lambda}} \mathcal{L} = \mathbf{0} \\ &\text{where } \mathcal{L} = J(\boldsymbol{z}) + \boldsymbol{\lambda}^\top \boldsymbol{c}(\boldsymbol{z}) \end{aligned} \tag{18}$$

where $\mathcal{L}$ is the Lagrangian and $\boldsymbol{\lambda}$ is known as the Lagrange multiplier and is associated with the each of the constraints. These KKT conditions for the LQR problem involve a Lagrange multiplier for each dynamics constraint (for each timestep) and can be written in matrix form as:

$$\underbrace{\begin{bmatrix} \ddots & & & & \\ & \ell_{\boldsymbol{x}\boldsymbol{x}_t} & \ell_{\boldsymbol{x}\boldsymbol{u}_t} & \boldsymbol{f}_{\boldsymbol{x}_t}^\top & \\ & \ell_{\boldsymbol{u}\boldsymbol{x}_t} & \ell_{\boldsymbol{u}\boldsymbol{u}_t} & \boldsymbol{f}_{\boldsymbol{u}_t}^\top & \\ & \boldsymbol{f}_{\boldsymbol{x}_t} & \boldsymbol{f}_{\boldsymbol{u}_t} & & -\mathbf{I} \\ & & & -\mathbf{I} & \ddots \end{bmatrix}}_{\mathbb{K}} \begin{bmatrix} \vdots \\ \boldsymbol{x}_t \\ \boldsymbol{u}_t \\ \boldsymbol{\lambda}_t \\ \boldsymbol{x}_{t+1} \\ \vdots \end{bmatrix} = - \begin{bmatrix} \vdots \\ \ell_{\boldsymbol{x}_t} \\ \ell_{\boldsymbol{u}_t} \\ \bar{\boldsymbol{f}}_t \\ \ell_{\boldsymbol{x}_{t+1}} \\ \vdots \end{bmatrix}, \tag{19}$$

where $\boldsymbol{\lambda}$ is the Lagrange multiplier associated with the dynamics and starting state constraint of the LQR. The next step is to differentiate the optimality conditions (KKT conditions) of the iLQR problem, which is similar to differentiating the conditions of the unconstrained problem. The authors in [2] then apply the approach of their previous work ([3]) to differentiate this problem. To do so, they replace the right side of the KKT conditions by a vector of $\mathbf{0}$ and the derivative of any upper-level cost J_{UL} (such as the co-design cost J_{CD}):

$$\mathbb{K} \begin{bmatrix} \vdots \\ d_{\boldsymbol{x}_t} \\ d_{\boldsymbol{u}_t} \\ d_{\boldsymbol{\lambda}_t} \\ d_{\boldsymbol{x}_{t+1}} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \nabla_{\boldsymbol{x}_t} J_{\text{UL}} \\ \nabla_{\boldsymbol{u}_t} J_{\text{UL}} \\ \mathbf{0} \\ \nabla_{\boldsymbol{x}_{t+1}} J_{\text{UL}} \\ \vdots \end{bmatrix} \tag{20}$$

This linear problem is also an LQR and gives the differential terms $d_{\boldsymbol{\tau}_t} = \{d_{\boldsymbol{x}_t} d_{\boldsymbol{u}_t}\}$ and $d_{\boldsymbol{\lambda}_t}$, which are then used to obtain the required derivatives:

$$\begin{aligned} &\nabla_{\boldsymbol{C}_t} J_{\text{UL}} = d_{\boldsymbol{\tau}_t} \otimes \boldsymbol{\tau}_t^*, \quad \nabla_{\boldsymbol{c}_t} J_{\text{UL}} = d_{\boldsymbol{\tau}_t}, \nabla_{\bar{\boldsymbol{x}}_1} J_{\text{UL}} = d_{\boldsymbol{\lambda}_0}, \\ &\nabla_{\boldsymbol{F}_t} J_{\text{UL}} = d_{\boldsymbol{\lambda}_{t+1}} \otimes \boldsymbol{\tau}_t^* + \boldsymbol{\lambda}_{t+1}^* \otimes d_{\boldsymbol{\tau}_t}, \nabla_{\bar{\boldsymbol{f}}_t} J_{\text{UL}} = d_{\boldsymbol{\lambda}_{t+1}}, \end{aligned} \tag{21}$$

where $\otimes$ denotes an outer product and $\boldsymbol{\tau}_t^* = \{\boldsymbol{x}_t^*, \boldsymbol{u}_t^*\}$ and $\boldsymbol{\lambda}_t^*$ are the optimal trajectory and Lagrange multipliers of the original OC problem. Finally, the Lagrange multipliers are obtained by the following recursive set of equations:

$$\begin{aligned} \boldsymbol{\lambda}_T^* &= \boldsymbol{h}_{\boldsymbol{x}\boldsymbol{x}_t} \boldsymbol{x}_T^* + \boldsymbol{h}_{\boldsymbol{x}_t} \\ \boldsymbol{\lambda}_t^* &= \boldsymbol{f}_{\boldsymbol{x}_t}^\top \boldsymbol{\lambda}_{t+1}^* + \boldsymbol{\ell}_{\boldsymbol{x}\boldsymbol{x}_t} \boldsymbol{x}_t^* + \boldsymbol{\ell}_{\boldsymbol{x}_t} + \boldsymbol{\ell}_{\boldsymbol{x}\boldsymbol{u}_t} \boldsymbol{u}_t^* \end{aligned} \tag{22}$$

This approach gives the correct derivatives of iLQR and is efficient, as it requires only an additional LQR to be solved to obtain the derivatives. However, since the authors differentiated the optimality conditions of the LQR approximation and not the original nonlinear problem, this approach encounters errors when scaled up to complex systems, where this approximation is inaccurate. The authors demonstrated their approach on pendulums and cartpoles, where this approximation gives the correct gradient direction (though not necessarily the correct magnitude). We demonstrate these errors and introduce the required corrections in order to differentiate the full nonlinear problem in Chapter 4.

2.4 MORPHOLOGY-OPTIMIZING APPROACHES

There exist approaches that solve the combinatorial optimization problem of finding the number of links and their placements. For instance, Zhao *et. al.* [99] and later Xu *et. al.* [97] used graph heuristic functions to construct the morphology of a robot based on a grammar, where links are added and removed from the robot. These methods use variants of Monte-Carlo searches that are *guided* by heuristic functions, which help select which links to add or remove. Ha *et. al.* [36] introduced another such heuristic based on virtual end-effectors that can solve for any motion and are iteratively replaced by more constrained real robot links. While these approaches for co-design allow for the construction of robots with varying morphologies (e.g. different number of end-effectors or legs), in this thesis we will consider the continuous problem of optimizing a fixed morphology. We will think of the morphology of the robot as a *design prior* specified by the designer and optimize link lengths, link attachment points and masses and inertias of the robot. In other words, our approach lets the designer find for instance the best quadruped for the task but will not determine whether a quadruped is the best robot (versus for instance a biped or an arm). The approaches discussed in this thesis can be included in the future within the broader morphology-finding algorithms to optimize the continuous parameters of non-fixed morphologies.

2.5 CO-DESIGN FOR MULTIPLE TASKS

There are recent approaches that consider co-design for multiple tasks and environments. Although this is out of the scope of this thesis, we believe it is an exciting direction for future work and the methods we develop in this thesis can be extended to address this problem. Bravo-Palacios *et. al.* [9] optimized designs and motions for multiple environments using *stochastic programming* where monolithic gradient-descent co-design is embedded within a stochastic optimization over environments. Xu *et. al.* [97] also optimize over multiple environments using a graph-based sampling heur-

istic for adding or removing robot links and a sampling-based planner for optimizing the motion. Kim *et. al.* [49] used Bayesian optimization to explore the space of designs for multiple tasks and find Pareto-optimal designs, which are designs that can not be improved on any of the tasks without decreasing their efficiency on other tasks.

2.6 SUMMARY OF LITERATURE

After having discussed the current state-of-the-art, we can summarize the benefits and drawbacks of each of the methods discussed:

- CMA-ES is the preferred approach when the number of decision variables is relatively low as complexity scales exponentially. There is no requirement for the OC problem to be differentiable but hard design constraints are not supported.
- A single *nonlinear program* that optimizes design and variables is preferred if the entire problem can be described with a single cost function and a set of constraints and the resulting problem is not intractable. The OC and co-design problem need to be differentiable and arbitrary design constraints are supported.
- Bi-level approaches that differentiate the OC problem are preferred when we have two objectives or can not express the problem as a single optimization due to intractability. It is also useful for machine learning, where we embed the OC problem within a larger learning framework. The OC and co-design/learning problem need to be differentiable.
- There exist methods from the machine learning literature ([2]) that can be directly applied to co-design and broadly used to differentiate the kinds of motion planners we use in robotics. However, we do not have experimental evidence of their scalability beyond toy systems like pendulums and cartpoles, for the purposes of co-design.

We also provide a table that compares some of the recent approaches for co-design (Table 1). This table is not meant to be exhaustive, as the same co-design approaches have been applied to various tasks, but rather it gives an example of each of the approaches discussed on the most relevant tasks as compared to our approach.

Note here that (i) for Skaterbots [29] and the follow-up work in [30] we are not given computational times and no source code is available, (ii) for Spielberg *et. al.* [84] experiments on the quadruped are available, but a high failure rate is reported (3/10 successful optimizations with different initial guesses) and no computational times are provided. For our results (as reported in detail in Chapter 3) we have two different co-design tasks (for trotting and for jumping) that took 37s. and 251s. to optimize respectively for the quadruped.

2.7 CONCLUSION

In this chapter we discussed optimal control (OC) formulations, how to extend them to formulate co-design problems and how to employ differentiable optimal control (DOC) and genetic algorithms to solve the co-design problems.

Work	Supports			Time/Robot
	Design Constraints	*Constrained Motion Planning*	*Fully Automatic*	
Desai *et. al.* [21]	✗	✗	✓	124.47s./ quadruped
Spielberg *et. al.* [84]	✓	✓	✓	685s./ biped
Vitruvio [16]	✗	✓	✓	15m. to 2.5h./ quadruped
Geilinger *et. al.* [30]	✗	✗	✓	?
Ha *et. al.* [38]	✓	✓	✗	2.23h. / quadruped
Ours (Chapter 3)	✓	✓	✓	37s. to 251s./ quadruped

Table 1: A comparison of different co-design approaches and their features and computational speed.

We further outlined the directions for the rest of this thesis. In the next chapter (Chapter 3) we introduce our co-design formulation based on bi-level optimization and how we scale and solve for co-design problems on quadrupeds. In Chapter 4 we discuss scaling up differentiable optimal control (DOC) formulations to the Kinova arm and numerical accuracy issues we encounter when using existing approaches and how to address them. Finally, we discuss hardware results and simulation-to-reality for co-design in Chapter 5.

Part 1
Co-Design

3 A VERSATILE CO-DESIGN APPROACH FOR DYNAMIC LEGGED ROBOTS

This chapter presents a versatile framework for the computational co-design of legged robots and dynamic maneuvers. Current state-of-the-art approaches are typically based on random sampling or concurrent optimization. We propose a novel bilevel optimization approach that exploits the derivatives of the motion planning sub-problem (i.e., the lower level). These motion-planning derivatives allow us to incorporate arbitrary design constraints and costs in an general-purpose nonlinear program (i.e., the upper level). Our approach allows for the use of any differentiable motion planner in the lower level and also allows for an upper level that captures arbitrary design constraints and costs. It efficiently optimizes the robot's morphology, payload distribution and actuator parameters while considering its full dynamics, joint limits and physical constraints such as friction cones. We demonstrate these capabilities by designing quadruped robots that jump and trot. We show that our method is able to design a more energy-efficient Solo robot for these tasks.

3.1 INTRODUCTION

We provide a brief overview of the key requirements and motivation for co-design, which we will refer to during the rest of this chapter. A more in-depth discussion is provided in Chapter 1 and a literature review – in Chapter 2.

To design a robot capable of executing dynamic motions, we need to consider the robot's mechanical design as well as the motion it will execute. A traditional approach is to iterate between mechanical design and motion planning (e.g., [80]). However, it is a challenging process, especially for complex and dynamic robots, as it requires experts both in motion planning and mechanical design. Instead, *concurrent design* (co-design [54]) aims to automate this process by numerically optimizing both the motion and design parameters. As the designer, we first specify a set of design parameters (e.g., morphologies or motor characteristics), constraints (e.g., collision avoidance between robot components), high-level tasks (e.g., a jump) and evaluation metrics (e.g., energy). The algorithm then finds optimal design parameters and motions to more efficiently execute the task.

For the algorithm to find realistic design improvements, it needs to be able to plan feasible motions by considering the robot's full-body dynamics and actuation limits. We can do it efficiently through motion planning frameworks such as Crocoddyl [58], which can run fast enough for predictive control applications [59]. On the other hand, from a designer standpoint, we need to be able to specify arbitrary design constraints and cost functions in order to give the designer tools to fully specify all the parameters of the design.

Re-implementing motion planning in order to add additional design parameters requires considerable technical work, which is why we seek a modular framework that exploits state-of-the-art motion planners while considering design constraints. With

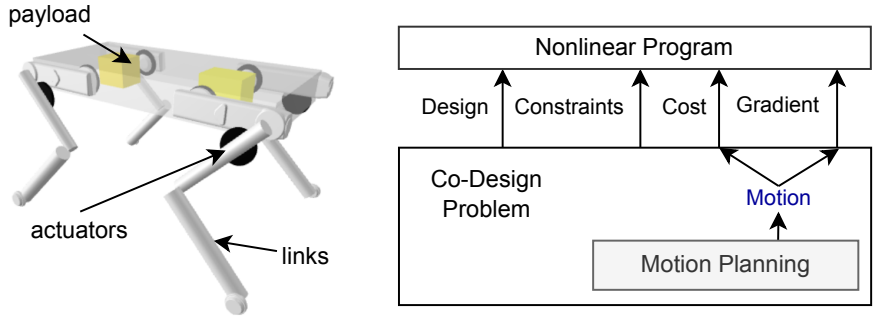


Figure 8: Illustration of our bilevel optimization approach for robot co-design. In the upper level we use gradient information from the motion planning (lower level) to optimize the design of the robot. Please find the accompanying video at https://youtu.be/Yxn7K1HXt_I

this motivation in mind, we developed a co-design algorithm with the following scope: 1) ability to define arbitrary costs and constraints on continuous design variables, 2) treat the motion planning as a module, and 3) exploit state of the art motion planners that can compute dynamic motion for legged robots which include constraints on the motion parameters.

In this chapter, we propose an approach where we directly take the derivative of the motion planner and embed it into a nonlinear program. Our approach contains an upper and a lower level optimization for robot design and motion planning, respectively. In the lower level, we use an efficient state-of-the-art constrained motion planner, which is continuously differentiable. In the upper level, we formulate the design constraints and metrics as a nonlinear program, which we solve with a general-purpose nonlinear optimization software that handles arbitrary constraints.

Our approach is modular for differentiable motion planners, similar to genetic algorithms, while also supporting hard constraints on design parameters, which genetic algorithms do not. Since it uses derivative information, it inherently has faster local convergence. Finally, it does not require unconstrained motion planning (as is the case in [29]).

3.2 Co-design formulation – robot model, cost function and constraints

We begin by describing the co-design problem for the quadruped robot Solo. We then discuss how to solve it via bi-level optimization.

Our design vector ρ consists of the lengths of the lower- and upper-leg limbs, the x-, and z-attachment points of the legs, the trunk shape: width, height and depth. We also model the x-, and z-positions of the two electronics boxes in the base of the robot. We thus implicitly constrain a symmetrical design along the direction of motion (the x-direction). A visualization of the robot design parameterization is in Figure 8.

Next, we use an actuator model and optimize both the gear ratio and motor mass, which are the same for all motors, for simplicity. All these properties are included in the robot model to compute masses and inertias of the relevant links. For the limbs,

we scale the volume linearly with the length of the leg as a simple proxy measure for structural integrity.

3.2.1 *Actuator Model and Cost Function*

Following [24] and [98] we model the mass of the motor m_m and parameterize the control limits $\underline{\boldsymbol{u}}(\boldsymbol{\rho})$ and $\overline{\boldsymbol{u}}(\boldsymbol{\rho})$ using an exponential regression based on m_m. We used the regression values from [24], which were fitted on datasheets from *Antigravity*, *Turnigy*, *MultiStar* and *PropDrive* motors. This maps the mass of the motor to its control limits and is used to set the limits of the motion planning optimization too. The resulting function is:

$$\overline{\boldsymbol{u}}(\boldsymbol{\rho}) = -\underline{\boldsymbol{u}}(\boldsymbol{\rho}) = 5.48\, m_m^{0.97}. \tag{23}$$

Following [24], the dynamics of the system in the motion planning phase are frictionless and the actuator model is present in the co-design cost function. Given applied controls $\boldsymbol{u}$ at the robot's joints, the total torque at the motor (τ_t) is:

$$\tau_t = \frac{\boldsymbol{u}}{n} + \tau_f, \tag{24}$$

where n is the gear ratio and τ_f is the friction torque. The friction torque itself models the combined Coulomb and viscous friction at the transmission, which the motor needs to overcome. Thus:

$$\tau_f = \tau_\mu \operatorname{sign}(\omega_m) + b\,\omega_m, \tag{25}$$

where τ_μ is the Coulomb friction parameter, b is the viscous friction parameter and ω_m is the motor angular speed, which is n times the joint angular speed.

We then consider three power losses – mechanical power, Joule effect from the motor winding resistance, and friction losses from the transmission:

$$P_{\text{mech}} = \tau_f \omega_m, \quad P_{\text{joule}} = \frac{1}{K_m}\tau_f^2, \quad P_{\text{fric}} = \tau_f \omega_m, \tag{26}$$

where $K_m = 0.15 m_m^{1.39}$ is the speed-torque gradient of the motor, again computed using an exponential regression on the motor mass.

Unlike in [24], we cannot ignore the mechanical power, as the foot start and end positions are dependent on the robot body structure and the total energy is not conserved between designs (and thus not constant). We thus follow [98] and compute the integral of the above terms ignoring power regenerative effects, summed over each of the motors:

$$J_{\text{CD}} = \int_{t_0}^{t_N} \sum_{\text{motor}} P_{\text{elec}} + \max(P_{\text{fric}}, 0)\ \mathrm{dt}, \tag{27}$$

where $P_{\text{elec}} = \max(P_{\text{mech}} + P_{\text{joule}}, 0)$ is the positive electrical power (as defined in [98]). The friction power is separate, as it is due to the transmission. We integrate over the planning horizon and sum the non-negative power of each of the 12 Solo motors. Thus $J_{\text{CD}}(\cdot)$ is the integral of these power terms, corresponding to the energy used during the motion (the total work). Finally, we note that the Solo robot's actuators use a custom gearbox, thus making the gear ratio independent from the motor [35]. This allows us to treat them as separate optimization targets.

3.2.2 *Constraints*

We then specify constraints on the design vector $\boldsymbol{\rho}$. Firstly, we add a volumetric collision constraint on the electronics boxes, the Inertial Measurement Unit (IMU) box and the motherboard (MB) box:

$$(x_{\text{mb}} - z_{\text{imu}})^2 + (x_{\text{mb}} - z_{\text{imu}})^2 \leq (r_{\text{mb}} + r_{\text{imu}})^2, \tag{28}$$

where $x_{\text{mb}}, z_{\text{mb}}, x_{\text{imu}}, z_{\text{imu}}$ are the coordinates of the two boxes and $r_{\text{mb}} = 0.0361\text{m}, r_{\text{imu}} = 0.0282\text{m}$ are the radii of the smallest circumscribed sphere around them.

Finally, we specify linear constraints on the positions of the two electronics boxes and the positions of the legs so that they are within the base of the robot:

$$\begin{aligned} -\frac{w_b}{2} \leq & x_{\text{imu}} \leq \frac{w_b}{2}, -\frac{w_b}{2} \leq x_{\text{mb}} \leq \frac{w_b}{2}, -\frac{d_b}{2} \leq z_{\text{imu}} \leq \frac{d_b}{2}, \\ -\frac{d_b}{2} \leq & z_{\text{imu}} \leq \frac{d_b}{2}, -\frac{w_b}{2} \leq x_{\text{fr}} \leq \frac{w_b}{2}, \ -\frac{w_b}{2} \leq x_{\text{hr}} \leq \frac{w_b}{2}, \\ & -\frac{d_b}{2} \leq z_{\text{fr}} \leq \frac{d_b}{2}, \ -\frac{d_b}{2} \leq z_{\text{hr}} \leq \frac{d_b}{2} \end{aligned} \tag{29}$$

where w_b and d_b are the width and depth of the base and $x_{\text{fr}}, z_{\text{fr}}$ and $x_{\text{hr}}, z_{\text{hr}}$ are the x- and z-coordinates of the front and hind shoulders. Note these inequalities constraints are defined in the upper level optimization.

3.2.3 *Task Description*

We are interested in optimizing the Solo robot design for specific tasks. As such, we fix the task description and optimize for the most efficient robot in terms of energy.

For trotting, the high-level motion task is to take two steps forward, each of 0.05m, with a fixed step height of 0.05m. The step height is fixed, as the optimal step height is always 0m. We allocated 22 and 37 knots[1] for the swing and double support phases of the motion, respectively, and used a symplectic Euler integrator with time-step of 10ms.

For jumping, the task is to jump forward 0.1m with a step height of 0.15m. We used the same integrator and time-step as in the trotting case. We defined 20 knots for the flight phase and 40 knots for the take-off and landing phases.

Finally, for both tasks, the initial design parameters $\boldsymbol{\rho}_0$ were matched to the Solo robot design. We next describe how we solve the tasks for a given fixed design $\boldsymbol{\rho}$, i.e. our motion planning.

3.3 MOTION PLANNING

The motion planning level of our co-design optimization algorithm computes the motion trajectory $\boldsymbol{X} = \{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_N\}$ and controls $\boldsymbol{U} = \{\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N-1}\}$ given a task and design $\boldsymbol{\rho}$. Here N is the number of knots, also known as the planning horizon. We formulate this lower level optimization as a hybrid nonlinear optimal control problem with fixed contact sequence and timings (Equation (30)). In the following,

1 Knots are points in time for the discretization of the optimal control problem.

we additionally highlight in blue where the design vector $\boldsymbol{\rho}$ parameterizes the motion planning problem:

$$\underset{\boldsymbol{X},\boldsymbol{U}}{\operatorname{argmin}} \sum_{k=0}^{N-1} ||\boldsymbol{q}_k \ominus \boldsymbol{q}_{\text{ref}}||^2_{\boldsymbol{Q}} + ||\boldsymbol{v}_k||^2_{\boldsymbol{N}} + ||\boldsymbol{u}_k||^2_{\boldsymbol{R}} + ||\boldsymbol{\gamma}_k||_{\boldsymbol{K}} \tag{30}$$

s.t.

for each contact phase: $p \in \mathcal{P} = \{1, 2, \cdots, N_p\}$

if $\underline{\Delta} t_p \leq k \leq \bar{\Delta} t_p$:

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k \oplus \int_{t_k}^{t_k+\Delta t_k} \boldsymbol{v}_k \, dt, \qquad \text{(integrator)}$$

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \int_{t_k}^{t_k+\Delta t_k} \dot{\boldsymbol{v}}_k \, dt,$$

$$(\dot{\boldsymbol{v}}_k, \boldsymbol{\gamma}_k) = \boldsymbol{f}_p(\boldsymbol{q}_k, \boldsymbol{v}_k, \boldsymbol{u}_k, \boldsymbol{\rho}), \qquad \text{(contact dyn.)}$$

else:

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k,$$

$$(\boldsymbol{v}_{k+1}, \boldsymbol{\gamma}_k) = \boldsymbol{\Delta}_p(\boldsymbol{q}_k, \boldsymbol{v}_k, \boldsymbol{\rho}), \qquad \text{(impulse dyn.)}$$

$$\boldsymbol{g}(\boldsymbol{q}_k, \boldsymbol{v}_k, \boldsymbol{u}_k, \boldsymbol{\rho}) = \boldsymbol{0}, \qquad \text{(equality)}$$

$$\boldsymbol{h}(\boldsymbol{q}_k, \boldsymbol{v}_k, \boldsymbol{u}_k, \boldsymbol{\rho}) \leq \boldsymbol{0}, \qquad \text{(inequality)}$$

$$\underline{\boldsymbol{x}} \leq \boldsymbol{x}_k \leq \bar{\boldsymbol{x}}, \qquad \text{(state bounds)}$$

$$\underline{\boldsymbol{u}}(\boldsymbol{\rho}) \leq \boldsymbol{u}_k \leq \bar{\boldsymbol{u}}(\boldsymbol{\rho}). \qquad \text{(control bounds)}$$

The state $(\boldsymbol{q}, \boldsymbol{v}) \in X$ lies in a differential manifold formed by the configuration $\boldsymbol{q} \in \mathbb{SE}(3) \times \mathbb{R}^{n_j}$ and its tangent vector $\boldsymbol{v} \in \mathbb{R}^{n_x}$ (with n_x and n_j as the dimension of the state manifold and number of joints, respectively). The control $\boldsymbol{u} \in \mathbb{R}^{n_j}$ is the vector of input torques, $\boldsymbol{\gamma}_k$ is the vector of contact forces, $\ominus$ and $\oplus$ are the *difference* and *integration* operators of the state manifold, respectively. Then $\boldsymbol{q}_{\text{ref}}$ is the reference standing upright robot posture, and $\boldsymbol{f}_p(\cdot)$ represents the contact dynamics under the phase p. To account for effects of discrete contact changes, $\boldsymbol{\Delta}_p(\cdot)$ is used to define an autonomous system that describes the contact-gain transition ([26]). $\boldsymbol{Q}$, $\boldsymbol{N}$, $\boldsymbol{R}$ and $\boldsymbol{K}$ are positive-define weighting matrices, $(\underline{\boldsymbol{x}}, \bar{\boldsymbol{x}})$ and $(\underline{\boldsymbol{u}}, \bar{\boldsymbol{u}})$ are the lower and upper bounds of the system state and control. $\underline{\Delta} t_p$ and $\bar{\Delta} t_p$ defines the timings of the contact phase p. We compute the hybrid dynamics and its derivatives as described in [58].

Below we provide more details about the hybrid dynamics, constraints and problem resolution. The hybrid dynamics implicitly compute the contact forces $\boldsymbol{\gamma}_k$ by assuming a rigid interaction (e.g., as in [10]).

The hybrid system is composed of contact and impulse dynamics, i.e.,

$$\begin{bmatrix} \dot{\boldsymbol{v}} \\ -\boldsymbol{\gamma} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M} & \boldsymbol{J}_c^\top \\ \boldsymbol{J}_c & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\tau}_b \\ -\boldsymbol{a}_0 \end{bmatrix} \text{ or}$$

$$\begin{bmatrix} \boldsymbol{v}^+ \\ -\boldsymbol{\gamma} \end{bmatrix} = \begin{bmatrix} \boldsymbol{M} & \boldsymbol{J}_c^\top \\ \boldsymbol{J}_c & \boldsymbol{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{M}\boldsymbol{v}^- \\ -e\boldsymbol{J}_c\dot{\boldsymbol{v}}^- \end{bmatrix},$$

respectively. Both dynamics describe a rigid contact where $\boldsymbol{M}$ is the joint-space inertia matrix and $\boldsymbol{J}_c$ is the contact Jacobian expressed in the local frame. $\boldsymbol{\tau}_b = \boldsymbol{S}\boldsymbol{u} - \boldsymbol{b}$

is the force-bias vector that accounts for the selected torque commands, Coriolis and gravitational effects. $\boldsymbol{a}_0$ is the desired acceleration in the constraint space (with Baumgarte stabilization as described in [58]). $e \in [0, 1]$ is the restitution coefficient, $\boldsymbol{v}^-$ and $\boldsymbol{v}^+$ are generalized velocities before and after the impulse, respectively. We compute the hybrid dynamics and its derivatives as described in [58].

During contact phases, we use a linearized friction-cone constraint via a $(\boldsymbol{A}\boldsymbol{\gamma}_{\mathcal{C}(k)} \leq \boldsymbol{r})$, where $(\boldsymbol{A}, \boldsymbol{r})$ are computed from a predefined number of edges, and minimum and maximum normal contact forces, respectively. $\mathcal{C}(k)$ describes the set of active contacts. During the swing phases, we also include contact-placement constraints $(\log(\boldsymbol{p}_{\mathcal{G}(k)}^{-1} \circ \boldsymbol{M}_{\boldsymbol{p}_{\mathcal{G}(k)}}) = \boldsymbol{0})$, where $\log(\cdot)$ describes the log operator used in Lie algebra, $\boldsymbol{p}_{\mathcal{G}(k)}$ and $\boldsymbol{M}_{\boldsymbol{p}_{\mathcal{G}(k)}}$ are the reference and current placements of the set of swing contacts $\mathcal{G}(k)$.

3.3.1 *Motion Planning Function*

In this chapter we consider the motion planner as a general function that maps the design and task to a trajectory: $\boldsymbol{X}, \boldsymbol{U} = \text{MP}(\boldsymbol{\rho}, \text{TASK})$. However, this motion planning function is not simply the optimization problem we described. To compute the motion, the motion planner takes in the following parameters:

- The task, which is the desired gait, consisting of the contact sequence and timings
- The initial joint configuration $\boldsymbol{q}_0$
- The robot's joint limits

Each of these are computed in the upper level from the design vector $\boldsymbol{\rho}$ and updated each time the optimizer calls the motion planner to compute the optimal trajectory. We compute the initial state of the robot $\boldsymbol{q}_0$ using inverse kinematics so that the angle at the knee joint of the shortest leg is 45°. We then run forward kinematics to set the foot positions, gait sequence and timings based on the task. We used the library Pinocchio [15] for computing the robot's kinematics and dynamics. We also set the lower and upper control bounds $(\underline{\boldsymbol{u}}, \bar{\boldsymbol{u}})$, and finally compute the optimal motion. In Algorithm 1, we show an overview of the procedure $\text{MP}(\boldsymbol{\rho}, \text{TASK})$.

After this setup of the initial conditions. we solve the motion planning problem (Eq. (30)) with the Feasibility-Driven Control-limited DDP (Box-FDDP) algorithm [60], a variant of the Differential Dynamic Programming (DDP) algorithm. Box-FDDP uses direct-indirect hybridization and enforces hard-constraints for the control limits. We employ a soft quadratic barrier to enforce inequality, equality and state constraints defined in Eq. (30). We implemented the algorithm using the open-source library Crocoddyl [58].

Crocoddyl uses differential dynamic programming (DDP) as the underlying solver. We chose DDP as the solver as it guarantees dynamics feasibility due to the nonlinear rollout performed in its forward pass (we discuss this in detail in Chapter 4). Furthermore, Crocoddyl uses the full rigid-body-dynamics of the robot and constraints on the control inputs, which in our case are a function of the mass of the actuator. Finally, Crocoddyl and DDP in general can be used to differentiate optimal control problems, which we discuss more in the next section as well as in Chapter 4.

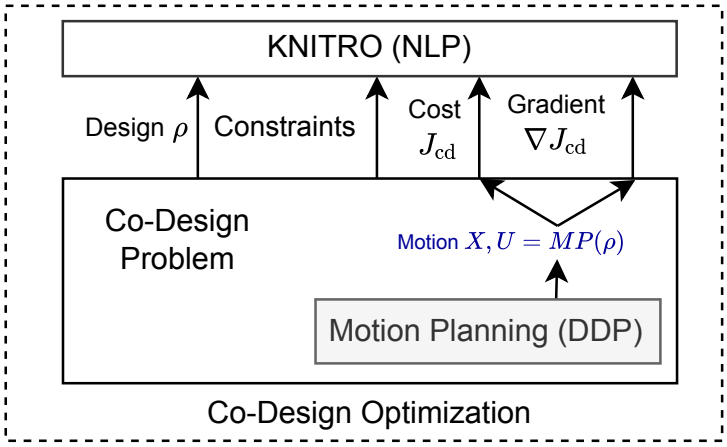


Figure 9: A schematic of our co-design framework. We optimize the robot's design by differentiating through a motion planner, which is the lower-level problem of our bi-level formulation.

3.4 CO-DESIGN THROUGH BILEVEL OPTIMIZATION

Our co-design framework is illustrated in Figure 9. We have already described the motion planning lower-level. We formulate the co-design problem over the design vector $\boldsymbol{\rho}$ as a bilevel optimization problem:

$$
\begin{aligned}
&\min_{\boldsymbol{\rho},\boldsymbol{X},\boldsymbol{U}} \; J_{\text{CD}}(\boldsymbol{\rho},\boldsymbol{X},\boldsymbol{U}) && \text{(co-design metric)}\\
&\text{s.t. } \boldsymbol{X},\boldsymbol{U} = \text{MP}(\boldsymbol{\rho};\text{TASK}), && \text{(motion planning)}\\
&\quad \underline{\boldsymbol{\rho}} \leq \boldsymbol{\rho} \leq \bar{\boldsymbol{\rho}}, && \text{(design bounds)}\\
&\quad \boldsymbol{g}_{\text{CD}}(\boldsymbol{\rho}) = 0, && \text{(equality design constraints)}\\
&\quad \boldsymbol{h}_{\text{CD}}(\boldsymbol{\rho}) \leq 0, && \text{(inequality design constraints)}
\end{aligned}
\tag{31}
$$

where $J_{\text{CD}}(\cdot)$ is a user-specified co-design metric that evaluates the performance of the design through the efficiency of the motion (in this chapter it is the total energy used), $\text{MP}(\cdot)$ is the motion planning function, which we described in Section 3.3, $\underline{\boldsymbol{\rho}}$ and $\bar{\boldsymbol{\rho}}$ are the lower and upper bounds of the design parameters. $\boldsymbol{g}(\boldsymbol{\rho})$ and $\boldsymbol{h}(\boldsymbol{\rho})$ are general equality and inequality constraints on the design vector (e.g., no collision constraints). For the specific problem of designing quadrupeds, we can instantiate this formulation with the cost and constraints described in Section 3.2.

We next rewrite the optimization moving the lower-level constraint into the cost function of the upper-level, making the constraint on the motion planning implicit:

$$
\begin{aligned}
&\min_{\boldsymbol{\rho}} \; J_{\text{CD}}(\boldsymbol{\rho},\text{MP}(\boldsymbol{\rho},\text{TASK})) && \text{(co-design metric)}\\
&\text{s.t. } \underline{\boldsymbol{\rho}} \leq \boldsymbol{\rho} \leq \bar{\boldsymbol{\rho}}, && \text{(design bounds)}\\
&\quad \boldsymbol{g}_{\text{CD}}(\boldsymbol{\rho}) = 0, && \text{(equality design constraints)}
\end{aligned}
$$

$$h_{CD}(\rho) \leq 0, \qquad \text{(inequality design constraints)} \tag{32}$$

To solve this problem, we only require the derivative of the cost function and the derivative of the constraints. The constraints are only dependent on ρ, hence their derivatives are readily available. Given how we re-wrote this problem, we propose to compute the cost derivative via finite differences. For each component of ρ, we have:

$$\nabla_{\rho_i} J_{CD} \approx \frac{J_{CD}(\rho_i^+, \text{MP}(\rho_i^+; \text{Task})) - J_{CD}(\rho, \text{MP}(\rho; \text{Task}))}{\epsilon} \tag{33}$$

where ρ_i^+ is the design vector with ϵ added to its i^{th} element, ϵ is a small number ($\epsilon = 1e-4$ in our experiments) and ρ_i is the i^{th} element of the design vector. Since the dimensions of ρ, $\dim(\rho)$ is small compared to that of the motion, we can compute this in parallel. For instance, in our trotting experiment, $\dim(\rho) = 17$, while $\dim(m) = 9163$. This parallel scheme requires calling the motion planner $\dim(\rho)$ times, which at first seems impractical. However, we can obtain a computational advantage using this approach, as modern motion planners are optimized for efficiency in order to be used in online planning on the robots and the dimensions of ρ is not prohibitive for some real-world co-design problems.

This approach directly considers the motion as a function of the motion planner and does not assume a particular form of motion planning. Thus it allows us to use the full-body dynamics, friction cone constraints, control and state bounds in a nonlinear optimal control formulation (motion planner). This is in contrast to previous fixed-point approaches ([38], [21], [29], [30]) in which (i) the update rule needs to be derived manually for the used motion planner and (ii) arbitrary design constraints (on the vector ρ) are not supported.

We present an overview of our algorithm in Algorithm 1. In the upper level, we use the interior-point/direct algorithm provided in KNITRO [12], which requires the derivatives of the motion planner using the parallel scheme described.

3.5 VERIFICATION IN SIMULATION

We also validated our design improvements in the PYBULLET physics simulator. ([19]). To do so, we execute the motion plan for both the nominal and the optimized designs, and record the percentage improvement in costs ΔJ_{CD} (similar to [74]). We use a proportional-derivative (PD) controller with feed-forward torque to track the planned motion:

$$u = u^* + K_p(q_j^* - q_j) + K_d(v_j^* - v_j), \tag{34}$$

where u^*, q_j^* and v^* are the reference feed-forward command, joint positions and velocities computed in Eq. (30), respectively. K_p and K_d are the PD gains. We tune these gains through a grid search procedure. We run the simulator on a 20×20 grid for $K_p \in [1, 20]$ and $K_d \in [0.1, K_d/2]$. Then, we pick the gains that lead to the smallest tracking error for both designs. This procedure allows us to fairly compare and account for different robot dimensions and weights, as larger robots require higher gains and vice-versa.

Algorithm 1 Co-design optimization

procedure MP($\boldsymbol{\rho}$; TASK)
  Compute initial state $\boldsymbol{q}_0$ using inverse kinematics
  Set control bounds $\underline{\boldsymbol{u}}, \bar{\boldsymbol{u}}$ based on actuator parameters
  Run forward kinematics on $\boldsymbol{q}_0$ and set foot positions
  Set gait sequence and timings based on TASK
  Compute $\boldsymbol{m}$, the optimal motion
  return $\boldsymbol{m}$
end procedure
procedure CoDesign
  Start at a design $\boldsymbol{\rho} = \boldsymbol{\rho}_0$
  while $J_{\text{CD}}(\boldsymbol{\rho}, \text{MP}(\boldsymbol{\rho}; \text{TASK}))$ decreasing **do**
    Compute $\nabla_{\boldsymbol{\rho}} J_{\text{CD}}$ via finite differences in parallel
    Update $\boldsymbol{\rho}$ using one step of the NLP solver
    Save the resulting motion to $\boldsymbol{m}$ and cost J_{CD}
  end while
  return $(\boldsymbol{\rho}, J_{\text{CD}})$ – optimal design and its cost value
end procedure

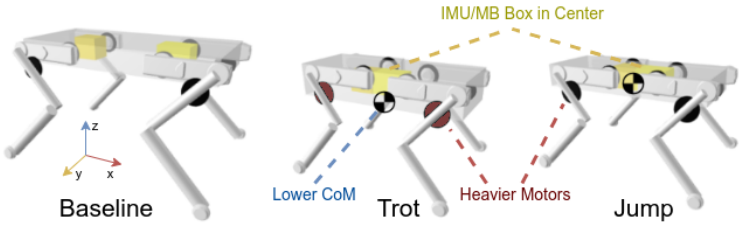
A designer can use this second stage to validate the correctness of the dynamics model used in motion planning and the improvements in co-design cost.

3.6 RESULTS

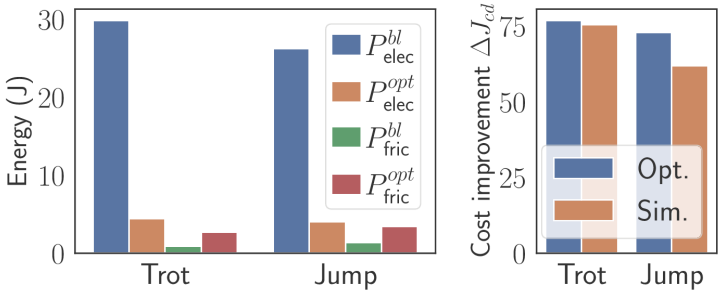
The resulting robot designs and cost improvements are in Figures 10a and 10b. For both trotting and jumping, we plotted the energy contributions from the positive electrical power at the motor as P_{elec} versus the friction contribution from the transmission as P_{fric}. The algorithm chooses to minimize the electro-mechanical losses while increasing the friction losses. This is similar to [24], as small motors are much more energy inefficient since the reciprocal of the speed-torque gradient exponentially decreases ($K_m = 0.15 m_m^{1.39}$), increasing the Joule losses.

For trotting specifically, the friction losses are smaller, as trotting is a more static motion with smaller motor velocities, and friction is velocity-dependent. Thus the dominating cost is the electro-mechanical energy. This allows for a heavier robot with bigger motors than the optimal design for a jumping task – the optimal motor mass is $m_m = 0.179$kg and gear ratio is $N = 16.062$ with a total robot weight of 3.805kg. The initial motor mass and gear ratio for the Solo robot are $m_m = 0.053$kg and $N = 9$ and the robot weighs 2.421kg. With a higher gear ratio the optimizer reduced the electro-mechanical energy further. Furthermore, we see a increase in base depth, which allows for the upper legs to be attached higher to the base of the robot. This allows for a lower center of mass, which can increase stability.

For jumping, however, a heavy robot is not optimal, as the entire mass of the robot needs to be moved. Thus the optimizer found $m_m = 0.168$ and $N = 17.325$ with a total mass of 3.592kg. The robot is heavier than the baseline, however the legs and the base are smaller. Compared to the optimal trotting design, the motors are lighter, but the



(a) Resulting robot designs for trotting and jumping.



(b) Cost improvements.

Figure 10: Robot designs and cost improvements on the trotting and jumping tasks. The costs are broken down for electric and friction contributions. We show the optimization and simulation percentage improvement on the bottom right.

gear ratio for both designs is similar. For both optimal designs, notably the boxes are optimally in the middle of the robot.

Finally, for both optimal designs, we also observed that the cost improvements remain in simulation within 10% of the ones found during optimization.

3.6.1 *Optimality and Scalability*

We compared our gradient-based co-design approach to the CMA-ES genetic algorithm on the trotting task in order to check convergence properties and optimality. We used the open-source CMA-ES library PyCMA [42]. In order to evaluate scalability, we varied the dimensions of the co-design vector by including subsets of the decision variables, namely:

1. $\dim(\boldsymbol{\rho}) = 4$ – leg lengths (front and back)
2. $\dim(\boldsymbol{\rho}) = 6$ – same as 4, and motor mass and gear ratio
3. $\dim(\boldsymbol{\rho}) = 9$ – same as 6, and base shape
4. $\dim(\boldsymbol{\rho}) = 13$ – same as 9, and electronics boxes
5. $\dim(\boldsymbol{\rho}) = 17$ – full model

For CMA-ES we specified a quadratic soft penalty for all constraints. We ran CMA-ES with population sizes $N = [10, 20, 50]$ and selected $N = 50$, which achieved the same or lower costs than our approach on all problems.

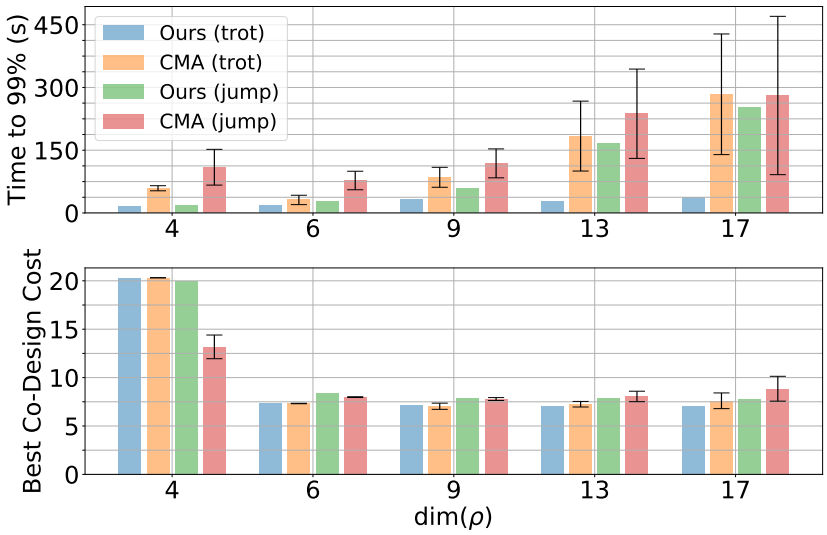


Figure 11: Scalability results for different problem dimensions.

Importantly, this corresponds to 50 calls to the motion planner by CMA at each iteration versus $\dim(\boldsymbol{\rho}) + 1$ calls for our approach. Thus we measure time to convergence, as a per-iteration measure would favor our approach heavily. Both approaches used multi-threading with 8 threads and were given the same computational budget.

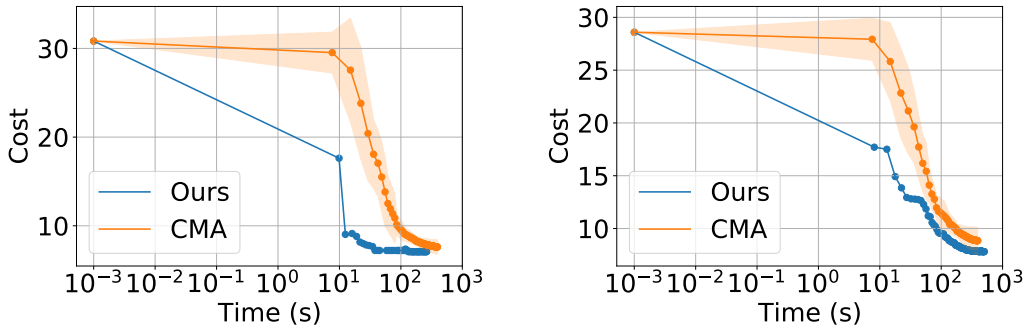
We then plot the mean and standard deviation for costs and time to 99% convergence over 20 runs at $N = 50$ in Fig. 11. On the trotting task our approach has better scalability than CMA-ES, which is expected given the convergence properties of CMA-ES. On the jumping task convergence is slower for both with CMA-ES having a large deviation in convergence time for larger problems. Importantly, CMA-ES is not deterministic and although the average time for the complex jumping task is comparable to our approach, the worst-case time we observed is 600 seconds for CMA versus 252 seconds for our approach (both for the 17-DOF jumping co-design task).

Finally, of interest is that we are able to achieve similar best co-design costs as CMA-ES across problem dimensions for the given co-design problems. Although we have no guarantee this is the true global optimum, there is empirical evidence that covariance matrix adaptation evolutionary strategy (CMA-ES) can find global optima with enough time to converge and sufficiently large population size [39, 41]. We chose the motion planning problem to be one where multi-modal solutions are unlikely. This means we used regularization terms on controls and state as well as predefined contacts to "force" the solution to be a trot or a jump for a design within our design bounds. This means that likely there are not multiple modes and a gradient-based optimizer is likely to be successful in finding the optimal solution.

3.6.2 *Convergence and Constraint Satisfaction*

One drawback of our approach is the need for differentiability and convexity of the motion planner with respect to (w.r.t.) design vector. In theory this can not always be guaranteed for nonlinear planners. However, due to the speed of the method, in practice we can simply run several iterations of the method and inspect the cost and constraint violation plots. Examples for the full 17-dof task are in Figure 12 and

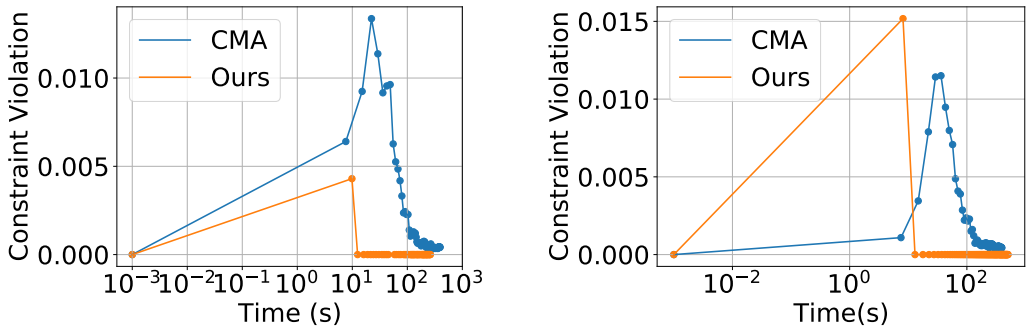
Figure 13, where we plotted the co-design costs (energy) and constraint violation for unsatisfied constraints, where 0 means full satisfaction.



(a) Trotting

(b) Jumping

Figure 12: Time versus co-design cost for our method versus CMA.



(a) Trotting

(b) Jumping

Figure 13: Time versus co-design constraint violation for our method versus CMA. Since we plot on a log-axis, the $t = 0$ iteration is hard-coded to be at 10^{-3} seconds.

These figures tell us that within approximately 10 seconds for the trotting task and within less than 50 seconds for the jumping task we already observe a significant improvement in cost. This in practice can guide whether we want to continue the optimization. For CMA this is not clear as quickly, as it takes more iterations for a significant cost improvements due to each iteration having small incremental improvements. The constraint violation plot (Figure 13) for our approach has at most one iteration where constraints are not satisfied with all other subsequent iterations satisfying all constraints. KNITRO does allow for constraints to not be satisfied *sometimes* during the optimization process to aid exploration and only enforcing it at convergence. CMA, on the other hand, does not enforce constraints at all (as we have soft constraints) and therefore it takes a significant amount of time for the soft constraints to be satisfied. Because of this we can not always interrupt the optimization prematurely for CMA but we can do so with our approach for all except the first iteration in both cases.

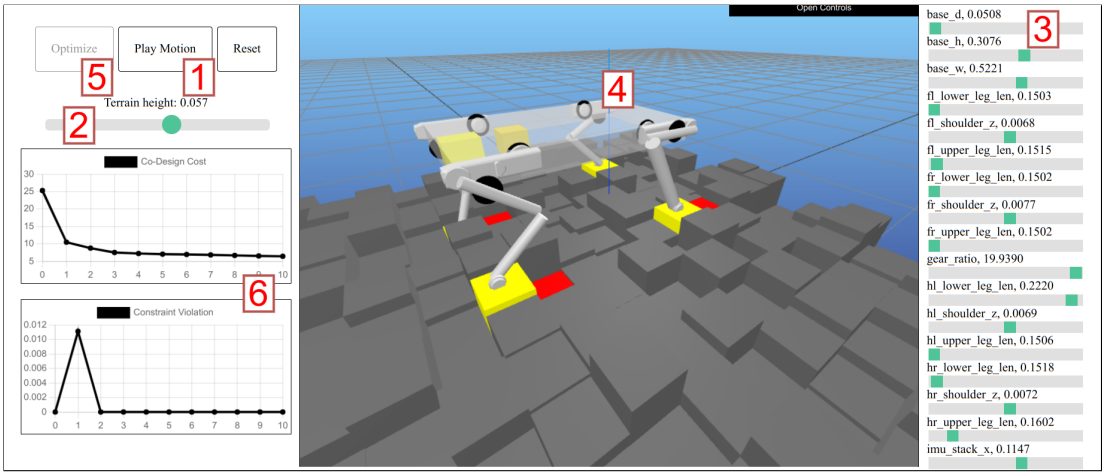


Figure 14: An interactive co-Design application that enables the user to optimize designs over different terrains. A video demonstration is available at https://youtu.be/k6vDrjwmsHs.

In this section, we present an interactive demonstration of our co-design framework that leverages the speed of computation we observed in the previous section. A screenshot of the application we built is in Figure 14 and a video demonstration is available at https://youtu.be/k6vDrjwmsHs. We extended our approach to allow the user to select a terrain height. The feet position and target position are computed automatically via inverse kinematics.[2] The application has the following features which correspond to the annotations in red in Figure 14:

1. The user can generate and playback the robot's motion based on the current terrain (1). The robot always moves from the red to the yellow tiles by trotting.

2. The user can generate different terrain heights (2) by moving the slider. This will perform inverse kinematics and place the robot's feet in the correct position in the middle of the red tiles.

3. The user can change the robot design (3). This will be visualized in the robot display (4) and also serves as the initial design that the optimization will start from.

4. The user can see both the robot's motion as well as the design while it is being optimized in this window (3 and 4).

5. Finally, the user can optimize the design for a given terrain and also start from a chosen design if they wish (3).

6. These windows (6) show the current co-design cost (as defined in this section) as well as the constraint violation of each iteration.

2 The gait and contact timings are still predefined, only the z-position (elevation) is based on the terrain.

Since optimization only takes a few seconds per iteration even on a personal computer (for the 21 parameters optimized on the right), the user can do this interactively over multiple terrains and for multiple starting designs. This kind of interactivity is what will enable rapid prototyping and evaluation of different robot design morphologies, even across a range of tasks and environments. We show here that we can achieve interactive speeds for a single task and up to 21 parameters in the application. We next discuss some of the drawbacks of our approach and how we propose to address them.

3.8 DISCUSSION AND CONCLUSION

In this chapter we proposed a modular co-design framework for dynamic quadruped locomotion. Our approach is based on bilevel optimization and exploits the derivatives of a hybrid nonlinear optimal control problem (lower level problem that describes the motion planner). We embed this problem in a nonlinear optimization in the upper level that optimizes for design variables. Our algorithm allows for the use of constrained state-of-the-art motion planners in the co-design loop together with linear and nonlinear design constraints in the upper level. One advantage of using DDP-style motion planning in our work is the guaranteed physical feasibility of the motion. When using other motion planners, this consistency might not be guaranteed and the resulting gradients might be noisy if the motion constraints are not satisfied.

Using finite differences to take the derivative of the motion planner serves as a baseline as it works in theory on any differentiable motion planning algorithm. Since DDP is such an algorithm, we can easily implement a bi-level optimization scheme using parallel finite differences. Even though this involves calling the motion planner $\dim(\boldsymbol{\rho})$ times per computation of the gradient, we still obtained a computational advantage, outperforming CMA-ES, thus enabling the creation of an interactive application.

Still, finite differences is one of the drawbacks of this approach. It introduces numerical errors when computing the derivative of the motion planner and require many calls until convergence of the motion planning as discussed. This approach still can be improved by taking derivatives analytically in a way that does not involve calling the motion planner once per dimension of the optimization vector. In the next chapter we show how to analytically take the derivatives of DDP-based motion planners and discuss considerations when implementing this in practice.

4 DIFFERENTIABLE OPTIMAL CONTROL VIA DIFFERENTIAL DYNAMIC PROGRAMMING

So far we have discussed robot design optimization (Chapter 3). Robot design optimization shares a common problem with imitation learning and system identification. These require optimization over robot or task parameters at the same time as optimizing the robot motion. To solve these problems, we can use *differentiable optimal control* for which the gradients of the robot's motion with respect to the parameters are required. We have shown how to use this gradient in practice in Chapter 3 using finite differences.

4.1 INTRODUCTION

Differentiable optimal control can be implemented via numerical differentiation (Chapter 3), automatic differentiation [2], or sensitivity analysis (SA) [2, 34]. Numerical differentiation has scalability issues but can be used when the dimensionality is low. Automatic differentiation (AD) can be applied to the entire optimization, however, this involves the unrolling of the optimization loop. It is memory intensive, suffers from slow computational times and requires the code to be compatible with AD tools, which is not feasible for complex robot problems [2][1]. Sensitivity analysis (SA) offers better computational speed, often requiring a single iteration through time. Using SA we can build on top of existing solvers without needing AD support. The downside of SA is the need to compute derivatives of dynamics w.r.t. the optimization variables which are otherwise often omitted to reduce algorithm complexity.

In this chapter we present a sensitivity analysis approach for differentiating optimal control problems via DDP, which is a second-order approach to solving OC problems [51, 67]. In the literature a related 1.5-order approach known as the iterative linear quadratic regulator [55] (iLQR) is often preferred, as it does not use second-order derivative tensors.

We know from theory that the second-order dynamics derivative terms are necessary for the computation of the solver derivatives, however in practice the work in [2] shows that these do not need to be considered on all systems (pendulum and cartpole in the case of [2]). We show a full derivation of the derivative of DDP and how to include these second-order terms to compute the exact derivative. We further simplify the resulting expression in the backward pass and give an analytical formula for its computation, whereas [2] relied on automatic differentiation for this computation.

We moreover carry out experiments to study whether these second-order dynamics terms are necessary. We measure this by whether the sign of the resulting gradient is correct, which is necessary for optimization. Our experiments confirm that on the pendulum the signs are correct (as in [2]), however we observe errors on the double pendulum system as well as on the Kinova arm.

1 Automatic differentiation is still useful to compute some of the components, e.g. dynamics derivatives, when analytical expressions are not available. For small systems, it can be as fast as analytical derivatives.

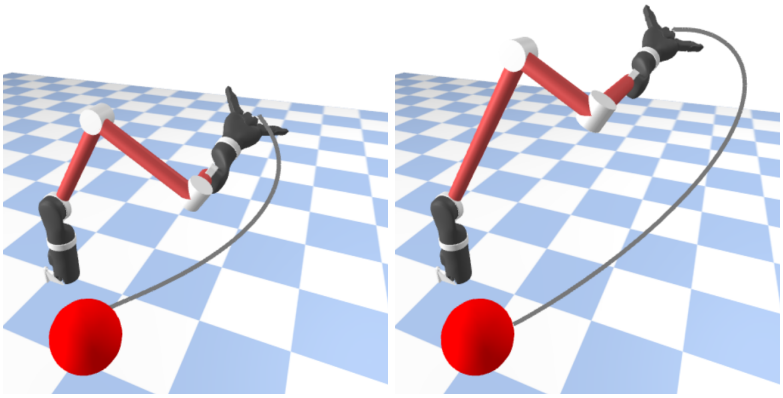


Figure 15: Optimized robot design for minimum joint velocity (left) and initial design (right) for the Kinova arm. The optimized robot has the end-effector at the same z-plane as the target (in red), thus needing only the base to move, which minimizes joint velocity. Please find the accompanying video at https://youtu.be/riXP9k2PUVs

Finally, we compare our approach that differentiates DDP to a simultaneous approach for co-design on the Kinova arm that shows a 54-times improvement in computational speed and also shows similar results in terms of cost improvement and design.

4.2 BACKGROUND

In this section we describe in details DDP and how it solves optimal control problems in detail. Consider the nonlinear optimal control (OC) problem parameterized by the vector $\boldsymbol{\theta}$ [2]:

$$\begin{aligned} \min_{X,U} J_{\text{OC}}(X, U; \boldsymbol{\theta}) &= h(\boldsymbol{x}_T; \boldsymbol{\theta}) + \sum_{t=1}^{T-1} \ell(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta}) \\ \text{s.t. } \boldsymbol{x}_{t+1} &= \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta}) \\ \boldsymbol{x}_1 &= \bar{\boldsymbol{x}}_1, \end{aligned} \tag{35}$$

where $\boldsymbol{x} = [\boldsymbol{q}, \dot{\boldsymbol{q}}]$ is the state of the system, composed of its generalized coordinates and velocities $(\boldsymbol{q}, \dot{\boldsymbol{q}})$. The nonlinear system evolves in time according to its dynamics $\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta})$, $X = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T\}$ is the set of states where T is number of states discretized in time, $U = \{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_{T-1}\}$ is the set of controls, $\bar{\boldsymbol{x}}_1$ defines the starting state, and $\boldsymbol{x}_T$ is the terminal state. $\{X, U\}$ is known as the system trajectory.

Subject to the dynamics constraints, the OC problem minimizes a set of nonlinear running costs $\ell(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta})$, and a terminal cost $h(\boldsymbol{x}_T; \boldsymbol{\theta})$. These costs allow us to define the goal of our OC problem. For instance, we could set-up $h(\boldsymbol{x}_T; \boldsymbol{\theta}) = (\boldsymbol{x}_T - \boldsymbol{x}^*)^\top \boldsymbol{Q}_f (\boldsymbol{x}_T - \boldsymbol{x}^*)$ and $\ell(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta}) = \boldsymbol{u}_t^\top \mathbf{R} \boldsymbol{u}_t$, where $\boldsymbol{x}^*$ is the goal state and the cost weights $\boldsymbol{Q}_f$, $\boldsymbol{R}$ penalize distance from the goal state and large control inputs, respectively. Then the OC problem finds a trajectory that reaches the goal state with

2 As we did in Chapter 2, we use θ to distinguish between design parameterization ($\boldsymbol{\rho}$) and more general parameterization, which includes parametric cost terms.

minimum control input. In this chapter we optimize some of the parameters of the OC problem, which can be the weights of the cost functions (e.g. $\boldsymbol{\theta} = \{\boldsymbol{Q}_f, \boldsymbol{R}\}$) when performing imitation learning or dynamic properties of the robot when optimizing its design.

4.2.1 *Solving the Optimal Control Problem*

We can solve the OC problem for a fixed $\boldsymbol{\theta}$ using DDP [61] or iLQR [55]. In the following, we omit $\boldsymbol{\theta}$ for clarity. We begin by defining the Value function as the optimal cost-to-go starting at time t:

$$\mathcal{V}(\boldsymbol{x}, t) \triangleq \min_{\boldsymbol{u}_t, \ldots, \boldsymbol{u}_{T-1}} h(\boldsymbol{x}_T) + \sum_{i=t}^{T-1} \ell(\boldsymbol{u}_i, \boldsymbol{x}_i). \tag{36}$$

We then define the action-value function (Q-function) as the optimal cost-to-go plus the current state-action cost:

$$Q(\boldsymbol{x}, \boldsymbol{u}, t) = \ell(\boldsymbol{x}, \boldsymbol{u}) + \mathcal{V}(\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}), t+1). \tag{37}$$

DDP and iLQR then iterate between (i) minimizing the quadratic approximation of the Q-function in a backward pass and (ii) integrating the system dynamics in a forward pass. They differ in the approximation used as we describe next.

In the backward pass we approximate the Q-function by a second order Taylor series around a reference trajectory $U^r = \{\boldsymbol{u}_1^r, \ldots, \boldsymbol{u}_{T-1}^r\}$ and $X^r = \{\boldsymbol{x}_1^r, \ldots, \boldsymbol{x}_T^r\}$:

$$\begin{aligned} Q(\boldsymbol{x}, \boldsymbol{u}) = Q(\boldsymbol{x}^r, \boldsymbol{u}^r) + \boldsymbol{Q}_{\boldsymbol{x}} \delta \boldsymbol{x} + \mathbf{Q}_{\mathbf{u}} \delta \boldsymbol{u} + \\ \frac{1}{2} \delta \boldsymbol{x}^\top \boldsymbol{Q}_{\boldsymbol{x}\boldsymbol{x}} \delta \boldsymbol{x} + \frac{1}{2} \delta \boldsymbol{u}^\top \boldsymbol{Q}_{\boldsymbol{u}\boldsymbol{u}} \delta \boldsymbol{u} + \delta \boldsymbol{u}^\top \boldsymbol{Q}_{\boldsymbol{u}\boldsymbol{x}} \delta \boldsymbol{x}. \end{aligned} \tag{38}$$

This approximation is done in the tangential space (i.e., δ-space), such that $\delta \boldsymbol{x} = \boldsymbol{x} - \boldsymbol{x}^r$ and $\delta \boldsymbol{u} = \boldsymbol{u} - \boldsymbol{u}^r$. We then give the derivatives of the Q-function, where $\mathcal{V}'$ is the value function at the next state:

$$\begin{aligned} \boldsymbol{Q}_{\boldsymbol{x}} &= \boldsymbol{\ell}_{\boldsymbol{x}} + \boldsymbol{f}_{\boldsymbol{x}}^\top \mathcal{V}'_{\boldsymbol{x}}, \\ \boldsymbol{Q}_{\boldsymbol{u}} &= \boldsymbol{\ell}_{\boldsymbol{u}} + \boldsymbol{f}_{\boldsymbol{u}}^\top \mathcal{V}'_{\boldsymbol{x}}, \\ \boldsymbol{Q}_{\boldsymbol{x}\boldsymbol{x}} &= \boldsymbol{\ell}_{\boldsymbol{x}\boldsymbol{x}} + \boldsymbol{f}_{\boldsymbol{x}}^\top \mathcal{V}'_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{f}_{\boldsymbol{x}} + \underbrace{\mathcal{V}'_{\boldsymbol{x}} \cdot \boldsymbol{f}_{\boldsymbol{x}\boldsymbol{x}}}, \\ \boldsymbol{Q}_{\boldsymbol{u}\boldsymbol{u}} &= \boldsymbol{\ell}_{\boldsymbol{u}\boldsymbol{u}} + \boldsymbol{f}_{\boldsymbol{u}}^\top \mathcal{V}'_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{f}_{\boldsymbol{u}} + \underbrace{\mathcal{V}'_{\boldsymbol{x}} \cdot \boldsymbol{f}_{\boldsymbol{u}\boldsymbol{u}}}, \\ \boldsymbol{Q}_{\boldsymbol{u}\boldsymbol{x}} &= \boldsymbol{\ell}_{\boldsymbol{u}\boldsymbol{x}} + \boldsymbol{f}_{\boldsymbol{u}}^\top \mathcal{V}'_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{f}_{\boldsymbol{x}} + \underbrace{\mathcal{V}'_{\boldsymbol{x}} \cdot \boldsymbol{f}_{\boldsymbol{u}\boldsymbol{x}}}_{\text{DDP terms}}. \end{aligned} \tag{39}$$

The last set of terms in Eq. (39) is included in DDP, which is a second-order approach, but *excluded* in iLQR, which we described as 1.5-order, as it includes second-order cost derivatives, but not second-order dynamics derivatives.

We then minimize this quadratic approximation w.r.t. the control $\delta \boldsymbol{u}$:

$$\delta \boldsymbol{u}^* = \underset{\delta \boldsymbol{u}}{\operatorname{argmin}}\, Q(\cdot) = -\boldsymbol{Q}_{\boldsymbol{u}\boldsymbol{u}}^{-1} \boldsymbol{Q}_{\boldsymbol{u}} - \boldsymbol{Q}_{\boldsymbol{u}\boldsymbol{u}}^{-1} \boldsymbol{Q}_{\boldsymbol{u}\boldsymbol{x}} \delta \boldsymbol{x}. \tag{40}$$

This defines feed-forward terms $\boldsymbol{k} = \boldsymbol{Q}_{\boldsymbol{uu}}^{-1}\boldsymbol{Q}_{\boldsymbol{u}}$ and state feedback term $\boldsymbol{K} = \boldsymbol{Q}_{\boldsymbol{uu}}^{-1}\boldsymbol{Q}_{\boldsymbol{ux}}$ at time t. Using this result in the expansion of Q, we can obtain a set of equations for the optimal cost to go $\mathcal{V}$:

$$\begin{aligned}\mathcal{V} = Q(\boldsymbol{x}_t, \boldsymbol{u}_t^*) &= Q(\boldsymbol{x}_t^r, \boldsymbol{u}_t^r) - \boldsymbol{Q}_{\boldsymbol{u}}\boldsymbol{Q}_{\boldsymbol{uu}}^{-1}\boldsymbol{Q}_{\boldsymbol{u}},\\ \mathcal{V}_{\boldsymbol{x}} &= \boldsymbol{Q}_{\boldsymbol{x}} - \boldsymbol{Q}_{\boldsymbol{xu}}\boldsymbol{Q}_{\boldsymbol{uu}}^{-1}\boldsymbol{Q}_{\boldsymbol{u}},\\ \mathcal{V}_{\boldsymbol{xx}} &= \boldsymbol{Q}_{\boldsymbol{xx}} - \boldsymbol{Q}_{\boldsymbol{xu}}\boldsymbol{Q}_{\boldsymbol{uu}}^{-1}\boldsymbol{Q}_{\boldsymbol{ux}}.\end{aligned} \tag{41}$$

This process is repeated for time T to 1 recursively. Finally, a forward pass from 1 to $T-1$ computes the new reference trajectory using the optimal gains. We also use the full, non-linear system dynamics to ensure full feasibility:

$$\begin{aligned}\hat{\boldsymbol{x}}_1 &= \bar{\boldsymbol{x}}_1,\\ \delta\hat{\boldsymbol{u}}_t &= \hat{\boldsymbol{u}}_t - \boldsymbol{u}_t^r = -\boldsymbol{k}_t - \boldsymbol{K}_t(\hat{\mathbf{x}}_t - \mathbf{x}_t^r),\\ \hat{\boldsymbol{x}}_{t+1} &= \boldsymbol{f}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t),\end{aligned} \tag{42}$$

Note that this reduces to LQR when the dynamics are linear, which is solved in a single iteration. In the next section, we describe how to optimize the parameters $\boldsymbol{\theta}$ of the optimal control problem using DDP.

4.2.2 *Optimizing the Parameters of Optimal Control Problems*

To optimize the parameters $\boldsymbol{\theta}$ of the OC problem, we begin by formulating a bi-level optimization problem with an inner loop containing the parameterized OC problem, i.e.:

$$\begin{aligned}&\min_{\boldsymbol{\theta}, X, U} J_{\text{UL}}(\boldsymbol{\theta}, X, U)\\ &\text{s.t. } X, U = \text{OC}(\boldsymbol{\theta}),\end{aligned} \tag{43}$$

where $\text{OC}(\boldsymbol{\theta})$ is the optimal control minimization problem as defined in Eq. (35). This problem computes the state and control trajectories X, U and can be solved by both DDP and iLQR.

The bi-level optimization minimizes an upper-level cost J_{UL} by computing a set of optimal parameters $\boldsymbol{\theta}$ along the optimal trajectories X, U. This upper-level cost can be, for instance, the total energy when optimizing the design or an imitation learning loss function or a system identification cost as in [2]. We can then optimize $\boldsymbol{\theta}$ using gradient descent:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\nabla_{\boldsymbol{\theta}} J_{\text{UL}} \tag{44}$$

where η is the learning rate, k is the iteration, and $\nabla_{\boldsymbol{\theta}} J_{\text{UL}}$ is the gradient of the upper level cost-function we compute via DDP [3].

4.3 Derivatives via Differential Dynamic Programming (DDP)

To obtain $\nabla_{\boldsymbol{\theta}} J_{\text{UL}}$ via DDP, we first define the optimality conditions of the nonlinear problem. Then we show the equivalent LQR problem that has the same optimality conditions and use the method of [2] to differentiate it. Crucially, we derive updates to show where DDP terms are imperative to be included to ensure correctness.

3 We use $\mathrm{d}J_{\text{UL}}/\mathrm{d}\boldsymbol{\theta}$ and $\nabla_{\boldsymbol{\theta}} J_{\text{UL}}$ interchangeably for the total derivative.

4.3.1 *Optimality Conditions and Newton Update*

We begin by writing the Lagrangian of the nonlinear problem problem we defined in Equation 35:

$$\mathcal{L} = \sum_{t=1}^{T-1} \Big(\ell(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{\lambda}_t^\top \left(\boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t) - \boldsymbol{x}_{t+1} \right) \Big) + h(\boldsymbol{x}_T) + \boldsymbol{\lambda}_0^\top (\boldsymbol{x}_1 - \bar{\boldsymbol{x}}_1), \tag{45}$$

where $\boldsymbol{\lambda}_t$ are Lagrange multiplier associated with the initial conditions and the dynamics. We then write the optimality and primal feasibility conditions (KKT conditions) as:

$$\frac{\partial \mathcal{L}}{\partial \{\boldsymbol{x}_t, \boldsymbol{u}_t\}} = \mathbf{0}, \quad \frac{\partial \mathcal{L}}{\partial \boldsymbol{\lambda}_t} = \mathbf{0}, \tag{46}$$

which form a system of equations. We then apply the Newton method to the KKT conditions, obtaining the following update rule:

$$\underbrace{\begin{bmatrix} \ddots & & & & \\ & \mathcal{L}_{\boldsymbol{x}\boldsymbol{x}_t} & \mathcal{L}_{\boldsymbol{x}\boldsymbol{u}_t} & \boldsymbol{f}_{\boldsymbol{x}_t}^\top & \\ & \mathcal{L}_{\boldsymbol{u}\boldsymbol{x}_t} & \mathcal{L}_{\boldsymbol{u}\boldsymbol{u}_t} & \boldsymbol{f}_{\boldsymbol{u}_t}^\top & \\ & \boldsymbol{f}_{\boldsymbol{x}_t} & \boldsymbol{f}_{\boldsymbol{u}_t} & & -\mathbf{I} \\ & & & -\mathbf{I} & \ddots \end{bmatrix}}_{\mathbb{K}} \begin{bmatrix} \vdots \\ \delta \boldsymbol{x}_t \\ \delta \boldsymbol{u}_t \\ \boldsymbol{\lambda}_t^+ \\ \delta \boldsymbol{x}_{t+1} \\ \vdots \end{bmatrix} = - \begin{bmatrix} \vdots \\ \ell_{\boldsymbol{x}_t} \\ \ell_{\boldsymbol{u}_t} \\ \bar{\boldsymbol{f}}_t \\ \ell_{\boldsymbol{x}_{t+1}} \\ \vdots \end{bmatrix}, \tag{47}$$

where $\mathbb{K}$ is the KKT matrix, $\boldsymbol{\lambda}_t^+ = \boldsymbol{\lambda}_t + \delta \boldsymbol{\lambda}_t$ is the updated Lagrange multiplier and $\bar{\boldsymbol{f}}_t = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t) - \boldsymbol{x}_{t+1}$ defines the gaps in the dynamics.[4] The Hessian of the Lagrangian is:

$$\begin{aligned} \mathcal{L}_{\boldsymbol{x}\boldsymbol{x}_t} &= \ell_{\boldsymbol{x}\boldsymbol{x}_t} + \boldsymbol{\lambda}_t \cdot \boldsymbol{f}_{\boldsymbol{x}\boldsymbol{x}_t}, \\ \mathcal{L}_{\boldsymbol{x}\boldsymbol{u}_t} &= \ell_{\boldsymbol{x}\boldsymbol{u}_t} + \boldsymbol{\lambda}_t \cdot \boldsymbol{f}_{\boldsymbol{x}\boldsymbol{u}_t} = \mathcal{L}_{\boldsymbol{u}\boldsymbol{x}_t}^\top, \\ \mathcal{L}_{\boldsymbol{u}\boldsymbol{u}_t} &= \ell_{\boldsymbol{u}\boldsymbol{u}_t} + \boldsymbol{\lambda}_t \cdot \boldsymbol{f}_{\boldsymbol{u}\boldsymbol{u}_t}, \end{aligned} \tag{48}$$

The expressions in Equation 48 resemble the DDP update equation for Q. This is not a coincidence, as in fact $\boldsymbol{\lambda} := \mathcal{V}_{\boldsymbol{x}}$ and DDP is an efficient way of inverting the sparse KKT matrix, as described in [60].

4.3.2 *Derivatives of the Optimal Control Problem*

To obtain the analytical derivatives of the upper-level cost, we use the chain rule to split it into (i) the derivatives of the cost w.r.t. the DDP parameters $\boldsymbol{\xi}$ and (ii) the derivative of those parameters w.r.t. $\boldsymbol{\theta}_i$, the i^{th} component of $\boldsymbol{\theta}$:

$$\frac{\mathrm{d} J_{\text{UL}}}{\mathrm{d} \boldsymbol{\theta}_i} = \frac{\partial J_{\text{UL}}}{\partial \boldsymbol{\theta}_i} + \sum_t \frac{\partial J_{\text{UL}}}{\partial \boldsymbol{\xi}_t} \frac{\mathrm{d} \boldsymbol{\xi}_t}{\mathrm{d} \boldsymbol{\theta}_i}, \tag{49}$$

4 When using DDP or iLQR this gap is $\mathbf{0}$ due to the forward pass.

where ∂J_{UL} is the partial differential, $\mathrm{d}J_{\text{UL}}$ is the total differential, $\boldsymbol{\xi}_t = \{\boldsymbol{F}_t, \boldsymbol{C}_t, \boldsymbol{c}_t, \bar{\boldsymbol{f}}_t\}$ and these parameters are defined as:

$$\begin{aligned}
\boldsymbol{C}_t &= \begin{bmatrix} \mathcal{L}_{\boldsymbol{x}\boldsymbol{x}_t} & \mathcal{L}_{\boldsymbol{x}\boldsymbol{u}_t} \\ \mathcal{L}_{\boldsymbol{u}\boldsymbol{x}_t} & \mathcal{L}_{\boldsymbol{u}\boldsymbol{u}_t} \end{bmatrix} \quad \boldsymbol{c}_t = \begin{bmatrix} \ell_{\boldsymbol{x}_t} - \mathcal{L}_{\boldsymbol{x}\boldsymbol{x}_t}\boldsymbol{x}^r - \mathcal{L}_{\boldsymbol{x}\boldsymbol{u}_t}\boldsymbol{u}^r \\ \ell_{\boldsymbol{u}_t} - \mathcal{L}_{\boldsymbol{u}\boldsymbol{x}_t}\boldsymbol{x}^r - \mathcal{L}_{\boldsymbol{u}\boldsymbol{u}_t}\boldsymbol{u}^r \end{bmatrix} \\
\boldsymbol{F}_t &= \begin{bmatrix} \boldsymbol{f}_{\boldsymbol{x}_t} \\ \boldsymbol{f}_{\boldsymbol{u}_t} \end{bmatrix}, \quad \bar{\boldsymbol{f}}_t = \boldsymbol{f}(\boldsymbol{x}_t^r, \boldsymbol{u}_t^r; \boldsymbol{\theta}) - \boldsymbol{f}_{\boldsymbol{x}_t}\boldsymbol{x}_t^r - \boldsymbol{f}_{\boldsymbol{u}_t}\boldsymbol{u}_t^r, \\
\boldsymbol{F}_0 &= \boldsymbol{0}, \quad \bar{\boldsymbol{f}}_0 = \bar{\boldsymbol{x}}_1,
\end{aligned} \tag{50}$$

where $\boldsymbol{F}_0$ and $\bar{\boldsymbol{f}}_0$ define the initial conditions of the problem. We now show how these parameters define an LQR problem, which has the same optimality conditions as the original problem. We can then use the approach of [2] to differentiate this LQR at the optimum. The LQR problem is obtained via a Taylor expansion:[5]

$$\begin{aligned}
&\min_{\delta\boldsymbol{x}_t, \delta\boldsymbol{u}_t} \sum_{t=1}^{T-1} \frac{1}{2} \begin{bmatrix} \delta\boldsymbol{x}_t \\ \delta\boldsymbol{u}_t \end{bmatrix}^\top \begin{bmatrix} \mathcal{L}_{\boldsymbol{x}\boldsymbol{x}_t} & \mathcal{L}_{\boldsymbol{x}\boldsymbol{u}_t} \\ \mathcal{L}_{\boldsymbol{x}\boldsymbol{u}_t} & \mathcal{L}_{\boldsymbol{u}\boldsymbol{u}_t} \end{bmatrix} \begin{bmatrix} \delta\boldsymbol{x}_t \\ \delta\boldsymbol{u}_t \end{bmatrix} + \begin{bmatrix} \ell_{\boldsymbol{x}_t} \\ \ell_{\boldsymbol{u}_t} \end{bmatrix}^\top \begin{bmatrix} \delta\boldsymbol{x}_t \\ \delta\boldsymbol{u}_t \end{bmatrix} \\
&\text{s.t. } \delta\boldsymbol{x}_{t+1} = \begin{bmatrix} \boldsymbol{f}_{\boldsymbol{x}_t} \\ \boldsymbol{f}_{\boldsymbol{u}_t} \end{bmatrix}^\top \begin{bmatrix} \delta\boldsymbol{x}_t \\ \delta\boldsymbol{u}_t \end{bmatrix},
\end{aligned} \tag{51}$$

where, importantly, the second-order DDP terms from Equation 47 are included (in $\mathcal{L}_{\boldsymbol{x}\boldsymbol{x}}, \mathcal{L}_{\boldsymbol{x}\boldsymbol{u}}$ and $\mathcal{L}_{\boldsymbol{u}\boldsymbol{u}}$). This problem has the same KKT matrix and gives the same update step for $\delta\boldsymbol{x}, \delta\boldsymbol{u}$ as applying the Newton method in Equation 47. We expand and collect terms to obtain the equivalent LQR problem in $\boldsymbol{x}, \boldsymbol{u}$-space:

$$\begin{aligned}
&\min_{\boldsymbol{x}_t, \boldsymbol{u}_t} \sum_{t=1}^{T-1} \frac{1}{2} \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix}^\top \boldsymbol{C}_t \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix} + \boldsymbol{c}_t \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix} \\
&\text{s.t. } \boldsymbol{x}_{t+1} = \boldsymbol{F}_t^\top \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix} + \bar{\boldsymbol{f}}_t.
\end{aligned} \tag{52}$$

where the matrices $\boldsymbol{F}_t, \boldsymbol{C}_t, \boldsymbol{c}_t, \bar{\boldsymbol{f}}_t$ obtained by expanding $\delta\boldsymbol{x}, \delta\boldsymbol{u}$ and collecting terms from Equation 51 as defined above. With this, we can compute the derivative of any upper-level cost function J_{UL} w.r.t. to these parameters by differentiating through the KKT conditions of the LQR problem. This approach was proposed by [2], and we summarize it below.

To compute the derivatives of the KKT conditions, we replace the linear terms on the right side of Equation 47 with the derivatives of the upper-level function (and $\boldsymbol{0}$):

$$\mathbb{K} \begin{bmatrix} \vdots \\ d_{\boldsymbol{x}_t} \\ d_{\boldsymbol{u}_t} \\ d_{\boldsymbol{\lambda}_t} \\ d_{\boldsymbol{x}_{t+1}} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \nabla_{\boldsymbol{x}_t} J_{\text{UL}} \\ \nabla_{\boldsymbol{u}_t} J_{\text{UL}} \\ \boldsymbol{0} \\ \nabla_{\boldsymbol{x}_{t+1}} J_{\text{UL}} \\ \vdots \end{bmatrix}. \tag{53}$$

5 Here we do not include the terminal cost explicitly. Introduce $\delta\boldsymbol{u}_T = \boldsymbol{0}$ and set $\ell_T(\boldsymbol{x}, \boldsymbol{u}; \boldsymbol{\theta}) := h(\boldsymbol{x}; \boldsymbol{\theta})$ to obtain the equivalent problem. This avoids the need for an extra term for the terminal cost and makes the rest of the section easier to follow.

This forms "another" LQR problem as it shares the same KKT matrix $\mathbb{K}$ as the optimal control problem. In fact this allows us to reuse the computation of the final backward pass of DDP, as we do not need to invert the KKT matrix twice. Solving the linear problem in Equation 53 provides the differential terms $d_{\boldsymbol{\tau}_t} = \{d_{\boldsymbol{x}_t} d_{\boldsymbol{u}_t}\}$ and $d_{\boldsymbol{\lambda}_t}$, which are then used to obtain the required derivatives:

$$\begin{aligned}
\nabla_{\boldsymbol{C}_t} J_{\text{UL}} &= d_{\boldsymbol{\tau}_t} \otimes \boldsymbol{\tau}_t^* \\
\nabla_{\boldsymbol{c}_t} J_{\text{UL}} &= d_{\boldsymbol{\tau}_t}, \\
\nabla_{\bar{\boldsymbol{x}}_1} J_{\text{UL}} &= d_{\boldsymbol{\lambda}_0}, \\
\nabla_{\boldsymbol{F}_t} J_{\text{UL}} &= d_{\boldsymbol{\lambda}_{t+1}} \otimes \boldsymbol{\tau}_t^* + \boldsymbol{\lambda}_{t+1}^* \otimes d_{\boldsymbol{\tau}_t} \\
\nabla_{\boldsymbol{f}_t} J_{\text{UL}} &= d_{\boldsymbol{\lambda}_{t+1}},
\end{aligned} \tag{54}$$

where $\otimes$ denotes an outer product[6] and $\boldsymbol{\tau}_t^* = \{\boldsymbol{x}_t^*, \boldsymbol{u}_t^*\}$ and $\boldsymbol{\lambda}_t^*$ are the optimal trajectory and Lagrange multipliers of the original OC problem. Finally, the Lagrange multipliers are obtained by the following recursive set of equations:

$$\begin{aligned}
\boldsymbol{\lambda}_T^* &= \boldsymbol{h}_{\boldsymbol{x}\boldsymbol{x}_t} \boldsymbol{x}_T^* + \boldsymbol{h}_{\boldsymbol{x}_t}, \\
\boldsymbol{\lambda}_t^* &= \boldsymbol{f}_{\boldsymbol{x}_t}^\top \boldsymbol{\lambda}_{t+1}^* + \boldsymbol{\ell}_{\boldsymbol{x}\boldsymbol{x}_t} \boldsymbol{x}_t^* + \boldsymbol{\ell}_{\boldsymbol{x}_t} + \boldsymbol{\ell}_{\boldsymbol{x}\boldsymbol{u}_t} \boldsymbol{u}_t^*.
\end{aligned} \tag{55}$$

For details on the derivation of Equation 53, we refer the reader to [2]. We next describe how to include the DDP second-order terms, which is our key contribution.

4.3.3 *Computing Derivatives via DDP*

This last step involves inverting the KKT matrix in Equation 47. To do so, we can use Riccati recursion too, as it is equivalent to solving a single LQR problem, as we discussed. However, when we do so, we need to include the DDP terms from Equation 48. We need to solve the following sub-problems at each timestep:

$$d_{\boldsymbol{u}_t} = \underset{d_{\boldsymbol{u}_t}}{\operatorname{argmin}} \frac{1}{2} d_{\boldsymbol{\tau}_t}^\top \begin{bmatrix} \boldsymbol{Q}_{\boldsymbol{x}\boldsymbol{x}_t} & \boldsymbol{Q}_{\boldsymbol{x}\boldsymbol{u}_t} \\ \boldsymbol{Q}_{\boldsymbol{u}\boldsymbol{x}_t} & \boldsymbol{Q}_{\boldsymbol{u}\boldsymbol{u}_t} \end{bmatrix} d_{\boldsymbol{\tau}_t} + d_{\boldsymbol{\tau}_t}^\top \begin{bmatrix} \boldsymbol{Q}'_{\boldsymbol{x}_t} \\ \boldsymbol{Q}'_{\boldsymbol{u}_t} \end{bmatrix}. \tag{56}$$

We can re-use the derivatives of the value function from the final DDP iteration:

$$\begin{aligned}
\boldsymbol{Q}_{\boldsymbol{x}\boldsymbol{x}} &= \boldsymbol{\ell}_{\boldsymbol{x}\boldsymbol{x}} + \boldsymbol{f}_{\boldsymbol{x}}^\top \mathrm{V}'_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{f}_{\boldsymbol{x}} + \underbrace{\mathcal{V}'_{\boldsymbol{x}} \cdot \boldsymbol{f}_{\boldsymbol{x}\boldsymbol{x}}}, \\
\boldsymbol{Q}_{\boldsymbol{x}\boldsymbol{u}} &= \boldsymbol{\ell}_{\boldsymbol{x}\boldsymbol{u}} + \boldsymbol{f}_{\boldsymbol{x}}^\top \mathrm{V}'_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{f}_{\boldsymbol{u}} + \underbrace{\mathcal{V}'_{\boldsymbol{x}} \cdot \boldsymbol{f}_{\boldsymbol{x}\boldsymbol{u}}}, \\
\boldsymbol{Q}_{\boldsymbol{u}\boldsymbol{u}} &= \boldsymbol{\ell}_{\boldsymbol{u}\boldsymbol{u}} + \boldsymbol{f}_{\boldsymbol{u}}^\top \mathrm{V}'_{\boldsymbol{x}\boldsymbol{x}} \boldsymbol{f}_{\boldsymbol{u}} + \underbrace{\mathcal{V}'_{\boldsymbol{x}} \cdot \boldsymbol{f}_{\boldsymbol{u}\boldsymbol{u}}}_{\text{DDP terms}},
\end{aligned} \tag{57}$$

where V' is the value function for the LQR in Equation 53 and $\mathcal{V}'$ is the value function from the last iteration of the original optimal control solver (as defined in Equation 41). If using iLQR to solve the OC problem, these DDP terms need to be included to obtain the correct derivatives. Instead when we use DDP to solve the OC problem, the second-order derivatives of the $\boldsymbol{Q}$-function can be directly re-used. To see this, note that they only depend on the second-order derivatives of the value function ($\mathcal{V}'_{\boldsymbol{x}\boldsymbol{x}}$ and

6 The outer product is defined as $a \otimes b = ab^\top$.

$\mathtt{V}'_{\boldsymbol{xx}}$). However, those themselves are updated using only second-order derivatives of the $\boldsymbol{Q}$-function (Equation 41) which are shared through the KKT matrix $\mathbb{K}$ between the LQR (Equation 53) and DDP (Equation 47).

Finally, the first-order derivatives of $\boldsymbol{Q}'$ are different for the LQR and are given by:

$$\begin{aligned}\boldsymbol{Q}'_{\boldsymbol{x}} &= \nabla_{\boldsymbol{x}} J_{\text{UL}} + \boldsymbol{f}_{\boldsymbol{x}}^{\top} \mathtt{V}'_{\boldsymbol{x}} \\ \boldsymbol{Q}'_{\boldsymbol{u}} &= \nabla_{\boldsymbol{u}} J_{\text{UL}} + \boldsymbol{f}_{\boldsymbol{u}}^{\top} \mathtt{V}'_{\boldsymbol{x}}.\end{aligned} \tag{58}$$

Using an LQR to take the derivative of the OC problem without the DDP terms ($\mathcal{V}'_{\boldsymbol{x}}$) is equivalent to differentiating iLQR as done in [2]. This can give the incorrect gradient, which as we show experimentally later, may or may not impact the resulting optimization.

So far we have described how to take the derivative of the upper-level cost w.r.t. $\boldsymbol{\xi}$, the parameters of DDP itself. The key was the inclusion of the second-order DDP terms in the Lagrangian of the problem, which has not been observed/reported in the literature. Next we show how to compute the derivative w.r.t. optimization parameters $\boldsymbol{\theta}$. We additionally show how to simplify this expression in the case of co-design.

4.3.4 *Chain Rule*

Next, once we have the derivatives of the upper-level cost w.r.t. $\boldsymbol{\xi}$, we use the chain rule to compute the required derivatives w.r.t. $\boldsymbol{\theta}_i$ (giving derivatives element-wise to avoid complex tensor notation):

$$\nabla_{\boldsymbol{\theta}_i} J_{\text{UL}} = \frac{\mathrm{d} J_{\text{UL}}}{\mathrm{d}\boldsymbol{\theta}_i} = \frac{\partial J_{\text{UL}}}{\partial \boldsymbol{\theta}_i} + \sum_t \frac{\partial J_{\text{UL}}}{\partial \boldsymbol{\xi}_t} \frac{\mathrm{d}\boldsymbol{\xi}_t}{\mathrm{d}\boldsymbol{\theta}_i}. \tag{59}$$

The derivative term $\frac{d\tilde{\zeta}}{d\boldsymbol{\theta}_i}$ also requires second-order derivative terms w.r.t dynamics. The authors in [2] use automatic differentiation and PYTORCH([73]) for this derivative. For robotics problems, however, automatic differentiation is not always available, especially when using rigid body dynamics libraries (e.g., CROCODDYL or PINOCCHIO as presented in [58] and [15], respectively) and computing this last step needs to be done analytically (up to the dynamics themselves). Indeed one must be careful when deriving them, as a recursive update is required. To make our discussion complete, we give these derivatives analytically. To obtain the derivative we firstly expand the chain rule:

$$\begin{aligned}\frac{\mathrm{d} J_{\text{UL}}}{\mathrm{d}\boldsymbol{\theta}_i} = \frac{\partial J_{\text{UL}}}{\partial \boldsymbol{\theta}_i} + \sum_t \Bigg[&\left(\frac{\partial J_{\text{UL}}}{\partial \boldsymbol{F}_t}\right)^T \frac{\mathrm{d}\boldsymbol{F}_t}{\mathrm{d}\boldsymbol{\theta}_i} + \left(\frac{\partial J_{\text{UL}}}{\partial \bar{\boldsymbol{f}}_t}\right)^T \frac{\mathrm{d}\bar{\boldsymbol{f}}_t}{\mathrm{d}\boldsymbol{\theta}_i} \\ &+ \left(\frac{\partial J_{\text{UL}}}{\partial \boldsymbol{C}_t}\right)^T \frac{\mathrm{d}\boldsymbol{C}_t}{\mathrm{d}\boldsymbol{\theta}_i} + \left(\frac{\partial J_{\text{UL}}}{\partial \boldsymbol{c}_t}\right)^T \frac{\mathrm{d}\boldsymbol{c}_t}{\mathrm{d}\boldsymbol{\theta}_i} \Bigg].\end{aligned} \tag{60}$$

We then simply plug in the expressions from Equation 54 into the expanded chain rule, obtaining:

$$\frac{\mathrm{d} J_{\text{UL}}}{\mathrm{d}\boldsymbol{\theta}_i} = \frac{\partial J_{\text{UL}}}{\partial \boldsymbol{\theta}_i} + \sum_t \left(\left\langle \boldsymbol{\lambda}_t^* \otimes d_{\boldsymbol{\tau}_t}, \begin{bmatrix} \nabla_{\boldsymbol{\theta}_i} \boldsymbol{f}_{\boldsymbol{x}_t} \\ \nabla_{\boldsymbol{\theta}_i} \boldsymbol{f}_{\boldsymbol{u}_t} \end{bmatrix} \right\rangle_{\mathrm{F}} + \langle d_{\boldsymbol{\lambda}_t}, \boldsymbol{f}_{\boldsymbol{\theta}_i} \rangle_{\mathrm{F}} + \left\langle d_{\boldsymbol{\tau}_t}, \begin{bmatrix} \ell_{\boldsymbol{x}_t \boldsymbol{\theta}_i} \\ \ell_{\boldsymbol{u}_t \boldsymbol{\theta}_i} \end{bmatrix} \right\rangle_{\mathrm{F}} \right), \tag{61}$$

where $\otimes$ is the outer product, $\cdot$ denotes a tensor dot product (contraction), and $\langle , \rangle_F$ denotes the Frobenius product, which is an element-wise product followed by a summation.[7] To obtain the derivatives of $\boldsymbol{f}_{\boldsymbol{x}}$ and $\boldsymbol{f}_{\boldsymbol{u}}$ w.r.t. $\boldsymbol{\theta}_i$ we use the following recursive expression:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}_i} \boldsymbol{f}_{\boldsymbol{x}_t} &= \boldsymbol{f}_{\boldsymbol{x}\boldsymbol{\theta}_i} + \boldsymbol{f}_{\boldsymbol{x}\boldsymbol{x}_t} \cdot \frac{\mathrm{d}\boldsymbol{x}_t}{\mathrm{d}\boldsymbol{\theta}_i}, \\
\nabla_{\boldsymbol{\theta}_i} \boldsymbol{f}_{\boldsymbol{u}_t} &= \boldsymbol{f}_{\boldsymbol{u}\boldsymbol{\theta}_i} + \boldsymbol{f}_{\boldsymbol{u}\boldsymbol{x}_t} \cdot \frac{\mathrm{d}\boldsymbol{x}_t}{\mathrm{d}\boldsymbol{\theta}_i}, \\
\frac{\mathrm{d}\boldsymbol{x}_t}{\mathrm{d}\boldsymbol{\theta}_i} &= \boldsymbol{f}_{\boldsymbol{\theta}_i} + \boldsymbol{f}_{\boldsymbol{x}_t} \frac{\mathrm{d}\boldsymbol{x}_{t-1}}{\mathrm{d}\boldsymbol{\theta}_i},
\end{aligned} \tag{62}$$

where the last line follows directly from the nonlinear rollout performed in the forward pass (described in Equation 42). Note that we do not need third-order derivatives in this expression, which at first seems necessary as, for instance, we are differentiating $\mathcal{L}_{\boldsymbol{x}\boldsymbol{x}}$ w.r.t. $\boldsymbol{\theta}_i$. However, these terms in fact cancel out when we differentiate the KKT conditions.

Algorithm 2 Analytical Derivatives of Optimal Control via DDP

1: **procedure** DIFF-DDP($J_{\mathrm{UL}}(\cdot)$, $\boldsymbol{\theta}$)
2: $\boldsymbol{X}, \boldsymbol{U} = \mathrm{OC}(\boldsymbol{\theta})$ ▷ Compute the optimal trajectory using the solver.
3: Run an LQR to compute $\partial J_{\mathrm{UL}} / \partial \{\boldsymbol{F}_t, \boldsymbol{C}_t, \boldsymbol{c}_t, \bar{\boldsymbol{f}}_t\}$, adding second order DDP terms to the KKT matrix $\mathbb{K}$ (Equation 57).
4: Use the chain rule to obtain $\mathrm{d}J_{\mathrm{UL}} / \mathrm{d}\boldsymbol{\theta}_i$ (Equation 61) for every element $\boldsymbol{\theta}_i$.
5: **end procedure**

4.3.5 *Summary of Algorithm*

Our algorithm is summarized in Algorithm 2. For solving the optimal control problem it does not matter whether first or second-order terms are used, so long as the optimal control solver converges. We can thus include the second order terms only when computing derivatives (Equation 57) making this process flexible.

4.4 EXPERIMENTS ON PENDULUMS AND THE KINOVA ARM

To experimentally show the need for DDP derivatives (versus iLQR derivatives), we studied three systems: a pendulum, a double pendulum and the Kinova robotic arm. We started by validating the accuracy of our derivatives on the pendulum and double pendulum systems.

4.4.1 *Validation of Gradients*

For each experiment, we define an upper-level optimization vector $\boldsymbol{\theta} = \{\boldsymbol{\rho}, q_f\}$, where $\boldsymbol{\rho}$ defines the parametric dynamics used in each of the systems (described below in

7 This notation is unusual, but required, since we differentiate w.r.t. matrices, which means w.r.t. each matrix element. This is why we need element-wise operations for each matrix element.

each subsection) and q_f defines the diagonal entries of the weighting term at the terminal state cost:

$$h(\boldsymbol{x}_T; \boldsymbol{\theta}) = (\boldsymbol{x}_T - \boldsymbol{x}^*)^T \underbrace{\text{DIAG}(q_f, \ldots, q_f)}_{\boldsymbol{Q}_f} (\boldsymbol{x}_T - \boldsymbol{x}^*) \tag{63}$$

where DIAG creates a diagonal matrix and $\boldsymbol{x}^*$ is the goal state. For the pendulum and double pendulum, we first sampled 100 points for the vector $\boldsymbol{\theta}$ with problem-specific bounds defined below. Then we compared the average gradient error versus *Automatic Differentiation* (AD), which unrolls the entire optimization loop. We used the ForwardDiff [78] package in Julia [6] for automatic differentiation. We computed the error as the absolute difference:

$$\mathcal{G}_{\text{ERR}} = \sum_{\boldsymbol{\theta}_i} |\nabla^{\text{AD}}_{\boldsymbol{\theta}_i} J_{\text{UL}} - \nabla^{\text{SA}}_{\boldsymbol{\theta}_i} J_{\text{UL}}|, \tag{64}$$

where, again, SA stands for sensitivity analysis derivatives. We computed this for both DDP derivatives (our approach) as well as iLQR derivatives [2].

Additionally, we report results for both 64-bit and 128-bit float precision. The optimality conditions for any sensitivity-based approach hold only if the solver has converged. Since we are looking at accumulating gradients over a relatively large number of timesteps ($T = 50$ timesteps) convergence with better precision allowed us to evaluate whether any errors we observe are due to missing terms or simply due to poor convergence.

We used the common gradient convergence metric $\sum_t \boldsymbol{Q}_{\boldsymbol{u}} \boldsymbol{Q}_{\boldsymbol{uu}}^{-1} \boldsymbol{Q}_{\boldsymbol{u}}$ and specify a threshold of convergence of 10^{-15} when using 64-bit precision and 10^{-30} when using 128-bit precision. We re-sampled if the motion planning did not converge for 64-bit precision. For 128-bit precision we used the same samples as for 64-bit precision. Of those 0 and 4 samples did not converge when using 128-bit precision for the pendulum and double pendulum, respectively. Since few samples were excluded (less than 5 percent), we did not resample when using 128-bit precision. Next we describe our results on the pendulum and double pendulum systems.

4.4.2 *Pendulum (SysID/Imitation Learning)*

The first system we studied is a simple pendulum (e.g. [87, Chapter 2]). The pendulum is attached at the origin and has a parametrized link length $\boldsymbol{\rho}$. The state space is comprised of the angle and angular velocity of the joint and the controls are the torques applied at the joint. Thus the dimensions of the state space are $N_x = 2$ and $N_u = 1$.

We generated a swing-up trajectory for the pendulum with a known link length of $\rho = 0.5\,\text{m}$, $q_f = 10^3$. The trajectory has $T = 50$ knots with a timestep of $\Delta t = 10^{-2}$ seconds for a trajectory that lasts half a second. We defined an imitation learning/system identification upper level cost in the same way as the authors in [2] did:

$$J_{\text{UL}} = \sum_{t=1}^{T=49} ||\boldsymbol{u}_t - \boldsymbol{u}_t^i||_2, \tag{65}$$

where $\boldsymbol{u}_t^i$ is the imitation learning trajectory generated above. The goal of optimizing this function is to find the pendulum link length (system identification) together with

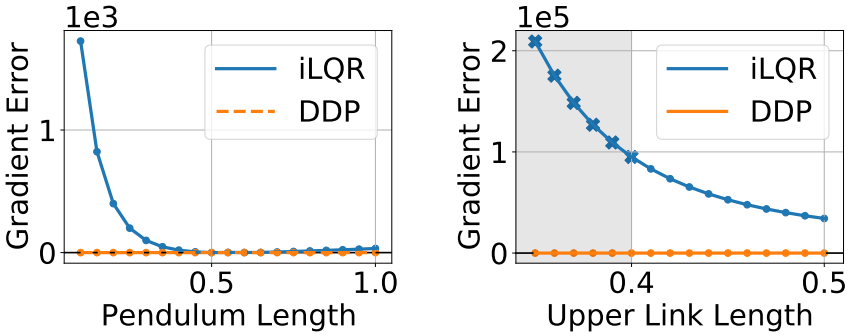


Figure 16: Pendulum (left) and double pendulum (right) gradient errors. An 'x' marker indicates the gradient has the wrong sign and a '.' – the correct sign. For the pendulum both methods give the same optimum and sign, however, for the double pendulum iLQR derivatives have the wrong magnitude and sign. The shaded area represents the region where iLQR has a wrong gradient sign. DDP always gives the correct sign.

the q_f (imitation learning). Note that this is one important benefit of differentiable optimal control, as we can solve both problems jointly. We then computed the gradient of the upper-level function w.r.t. the link length $\boldsymbol{\theta} = \{\rho, q_f\}$ for link lengths in the range $\rho \in [0.1, 1.0]$m and $q_f \in [1, 10^4]$. We additionally included the known optimum ($\rho = 0.5\,\text{m}$, $q_f = 10^3$) when computing the errors.

The numerical results for all variables are shown in Table 2 (Pendulum). We can see that using DDP terms leads to least error and the error reduces significantly when using 128-bit precision. However, for this problem we also see that the minimum error for all methods is small for 64-bit precision and goes to 0 for 128-bit precision. This is in fact at the known optimum for which both methods correctly give a gradient of 0. Furthermore, the sign of the gradient is correct for both methods everywhere. This suggests that the errors when using iLQR gradients are only in the gradient magnitudes. This would suggest that for optimization purposes it does not matter which derivative computation method we choose.

To illustrate this point further, we fixed q_f to the optimum and plot the gradient error w.r.t. the link length of the pendulum in Figure 16 (left). Although there are errors for the gradients away from the optimum, gradients for both methods intersect with the origin at the same point (the optimal $\rho = 0.5\,\text{m}$) and always have the correct sign.

The fact that both methods give the same optimum is an important observation that comes from the upper-level cost gradients. We remind the reader that to compute them we ran another LQR pass with the following substitutions:

$$\boldsymbol{c}_t = \left[\frac{\partial J_{\text{UL}}}{\partial \boldsymbol{x}_t}, \frac{\partial J_{\text{UL}}}{\partial \boldsymbol{u}_t}\right]^T, \quad \bar{\boldsymbol{f}}_t = \boldsymbol{0}. \tag{66}$$

If the partials above are $\boldsymbol{0}$ when the derivative of interest $\nabla_{\boldsymbol{\theta}} J_{\text{UL}}$ is also $\boldsymbol{0}$, then all methods will give a gradient of $\boldsymbol{0}$ at the same value of $\boldsymbol{\theta}$.[8] Note in this case the

8 Note that this error is non-zero for 64-bit precision in Table 2. This is due to that algorithmic differentiation gives differently wrong gradients for 64-bit precision, again due to convergence errors. This is another reason why we need 128-bit precision for comparison.

		Pendulum			
		Min err.	Max err.	Mean err.	Sign errors
64 bit	iLQR derivative	1.65e-05	1.63e+03	2.08e+02	0/100
	DDP derivative (ours)	**4.59e-07**	**9.61e-03**	**4.11e-04**	0/100
128 bit	iLQR derivative	0	1.63e+03	2.08e+02	0/100
	DDP derivative (ours)	0	**5.76e-14**	**4.77e-15**	0/100
		Double Pendulum			
		Min err.	Max err.	Mean err.	Sign errors
64 bit	iLQR derivative	1.42e+04	5.45e+05	1.04e+05	18/100
	DDP derivative (ours)	**2.41e-04**	**1.60e-01**	**2.13e-02**	**0/100**
128 bit	iLQR derivative	1.42e+04	4.44e+05	8.78e+04	18/100
	DDP derivative (ours)	**1.42e-15**	**4.05e-10**	**4.53e-12**	**0/100**

Table 2: Gradient errors as compared to Automatic Differentiation for iLQR and DDP gradients. Using iLQR derivatives gives large errors and on the double pendulum, can give wrong signs. With 128-bit precision the solver can achieve better numerical convergence – DDP gradients achieve significantly lower error (maximum below 1e−9) on both problems.

upper-level function only has gradients w.r.t. $\boldsymbol{u}_t$ and is $\mathbf{0}$ when the controls are the same as the imitation learning baseline.

In conclusion, for this task the gradient signs are correct for both iLQR and DDP and both are usable in optimization.

4.4.3 *Double Pendulum (SysID/Imitation Learning)*

In this section we will show an example of an upper level cost function and system for which the gradient sign and magnitude are wrong when using iLQR.

To begin, we define the double pendulum system (e.g. [87, Appendix B]). It consists of two links with a point mass at the end of each link. Each of the two link lengths is parameterized as $\boldsymbol{\rho} = \{l_1, l_2\}$. The state space consists of the angle and angular velocity of each link and the controls are the torques applied at the joints. Thus $N_x = 4$ and $N_u = 2$. The goal is to swing the pendulum from the bottom to the top position in $T = 50$ knots of $\Delta t = 10^{-2}$ seconds for a total trajectory length of half a second.

We generated a trajectory with $\boldsymbol{\rho} = \{0.5\,\mathrm{m}, 0.5\,\mathrm{m}\}$, $q_f = 10^3$. The imitation learning trajectory is different, however:

$$J_{\mathrm{UL}} = \dot{\boldsymbol{q}}_{50}^T \dot{\boldsymbol{q}}_{50} + \sum_{t=1}^{49} ||\boldsymbol{u}_t - \boldsymbol{u}_t^i||_2 + \dot{\boldsymbol{q}}_t^T \dot{\boldsymbol{q}}_t, \tag{67}$$

where we added a velocity cost on $\dot{\boldsymbol{q}}$, the angular velocity of the double pendulum. This cost was used for imitation learning/SysID of the target trajectory with the added objective of minimizing the joint velocity of the robot as well.

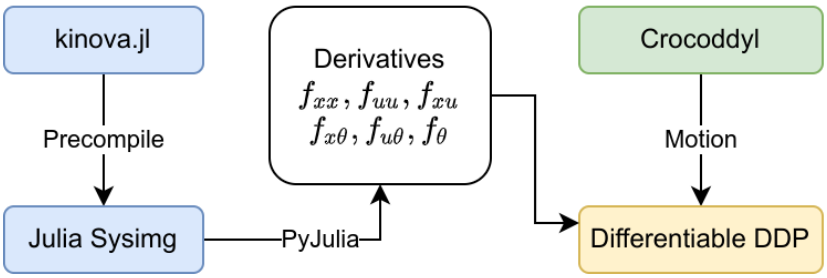


Figure 17: Pipeline for computing second-order derivatives of the rigid-body forward dynamics of the Kinova arm. These terms are needed for obtaining the analytical derivatives of the optimal control problem via DDP using Crocoddyl [58]

The resulting gradient errors for this cost and samples in the range $l_1, l_2 \in [0.25, 0.5]$m and $q_f \in [10^2, 10^4]$ are in Table 2 (Double Pendulum). Once again, using DDP terms leads to the least amount of error.

More importantly, the iLQR derivatives have errors in sign as well (18 out of the 100 samples had the wrong sign). To illustrate this point further, we plot the gradient for the second link length while holding all other variables constant in Figure 16 (right). Indeed we see that the iLQR gradients have the wrong sign for $\rho \leq 0.35$m. In fact iLQR gradients are always positive, where the true gradient is negative for $\rho \leq 0.35$m. Autodiff (AD) and the DDP gradients always have the same sign. We thus conclude that on this system it is important to use DDP terms when computing gradients.

4.4.4 *Co-Design with a Kinova Manipulator*

In this experiment we scaled up our approach to a robotic manipulator that demonstrates a real-world use-case for differentiable OC. We used Crocoddyl [58] as our motion planning library and show how to make it differentiable for the Kinova manipulator introduced in [14].

We parameterized three of the links of the Kinova 7-DOF manipulator as follows: $\boldsymbol{\theta} = \{l_1, l_2, l_3\}$. These three links are represented as cylinders (as shown in Figure 15) and the link inertias are computed based on the link lengths.

We formulated a motion planning problem that aims to reach the goal end-effector position of $[0.5, -0.5, 0.5]$m. The initial robot configuration, the end effector trajectory, and the goal are also shown in Figure 15. The motion planning horizon is specified by $T = 50$ knots with $\Delta t = 0.02$ seconds for an optimization horizon of 1 second. We then defined a co-design problem with an upper-level cost on the velocity of the joints together with a reachability cost that ensures the target is always reachable in the lower level:

$$J_{\text{UL}}(\boldsymbol{q}, \boldsymbol{\theta}) = \sum_{t=1}^{T} \dot{\boldsymbol{q}}_t^T \dot{\boldsymbol{q}}_t + \underbrace{10^3 \ \max\left(0, d - \sqrt{l_1^2 + l_2^2 + l_3^2}\right)^2}_{\text{Reachability}} \tag{68}$$

where d is the distance to the target, adjusted for the fixed-length robot links.

To compute the analytical derivative of the upper-level cost function, we need the second-order dynamics derivatives, as well as the derivatives of the dynamics w.r.t. the link lengths of the robot, which are currently unavailable analytically in rigid body

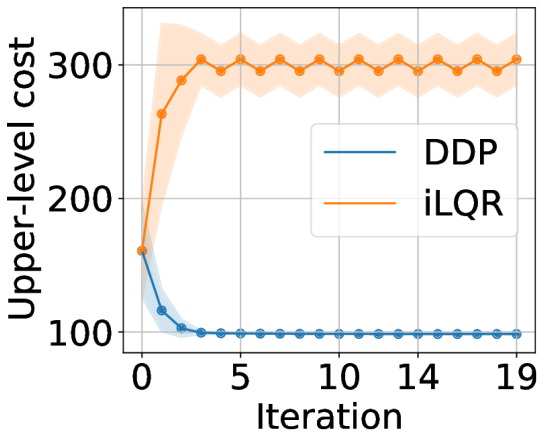


Figure 18: Kinova co-design optimization. Mean cost (joint velocity) and one standard deviation is shown. Using iLQR derivatives leads to wrong gradients that lead to divergence.

dynamics libraries such as Pinocchio [15] used by Crocoddyl, or RBD.jl [52], which we used for previous experiments in this chapter.

RBD.jl, however, supports algorithmic differentiation. Therefore, we created a robot model programmatically and inserted dual variables for the link lengths (to compute the required gradients). We only need to do this once after the optimization completes (as shown in Algorithm 2). We then re-used the factorization of the KKT matrix computed by Crocoddyl in order to compute the upper level cost. An illustration of our pipeline is in Figure 17.

We optimized the robot designs using gradient descent (Eq. (44) with $\eta = 5 \times 10^{-4}$). This co-design problem optimized the robot design that minimizes the joint velocity for the task of reaching a given end-effector position. To evaluate both iLQR and DDP derivatives more robustly, we sampled and optimized 50 initial robot designs (link-lengths) uniformly between 0.1 m and 0.5 m (for all three links), and also set the limits of the link lengths to $[0.1, 0.5]$m.

Figure 18 shows the mean costs and standard deviation, and Figure 15 shows an example design and the mean optimal designs (as found by our method). The mean optimal design found was $\boldsymbol{\theta} = \{0.330 \pm 0.003\,\text{m}, 0.289 \pm 0.003\,\text{m}, 0.1 \pm 0.0\,\text{m}\}$ with a maximum standard deviation of 3 mm. This design aligns the target with the end-effector on the z-plane, leading to minimum velocity. Using DDP derivatives is crucial for this problem, as the gradient descent optimization quickly diverges if we use the iLQR derivatives. Additionally, we observe little variation between samples when using the correct DDP derivatives and the optimization quickly converges (in fewer than 5 iterations) to the optimal design.

4.5 Comparison with a simultaneous approach

In this section we compare our approach to a simultaneous approach for co-design on the Kinova arm based on the nonlinear programming solver Ipopt described in [90]. We used the same problem goal and design bounds as already described. For

a simultaneous formulation we need to transcribe the co-design problem as a single nonlinear program (NLP). We used the formulation proposed in [84], i.e.:

$$\min_{\boldsymbol{X},\boldsymbol{U},\boldsymbol{\rho}} \sum_{t=1}^{T-1} \dot{\boldsymbol{q}}_t^T \dot{\boldsymbol{q}}_t + ||\boldsymbol{x}_t - \bar{\boldsymbol{x}}_1||_{\boldsymbol{R}_x} + ||\boldsymbol{u}_t||_{\boldsymbol{R}_u} \tag{69}$$

$$\text{s.t.} \quad \boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\rho}), \qquad \text{(Dynamics)}$$

$$\boldsymbol{x}_1 = \bar{\boldsymbol{x}}_1, \qquad \text{(Starting State)}$$

$$\boldsymbol{p}_{\text{EFF}} = [0.5, -0.5, 0.5], \qquad \text{(Goal)}$$

$$\dot{\boldsymbol{q}}_T = \boldsymbol{0}, \qquad \text{(Final Velocity)}$$

$$\{0.1, 0.1, 0.1\} \leq \boldsymbol{\theta} \leq \{0.5, 0.5, 0.5\}, \qquad \text{(Bounds)}$$

where in the cost we include the same regularization terms as DDP plus the co-design cost. The terms $\boldsymbol{R}_x = 0.5\boldsymbol{I}$ and $\boldsymbol{R}_u = 0.001\boldsymbol{I}$ are the regularization weights. [60] noted that using a cost term for the goal state significantly negatively affects convergence. [9] In our experiments the performance of the solver was even worse with a cost formulation with IPOPT not converging after 5000 iteration when only solving for the optimal control problem (versus 12 iterations when using the constraint formulation). For this reason we included the goal as a constraint instead with $\boldsymbol{p}_{\text{EFF}}$ being the position of the end effector at the final timestep. We additionally included bounds for the design variables, namely the link lengths of the Kinova arm and initialized both approaches with a control input that holds the arm in-place counteracting gravity.

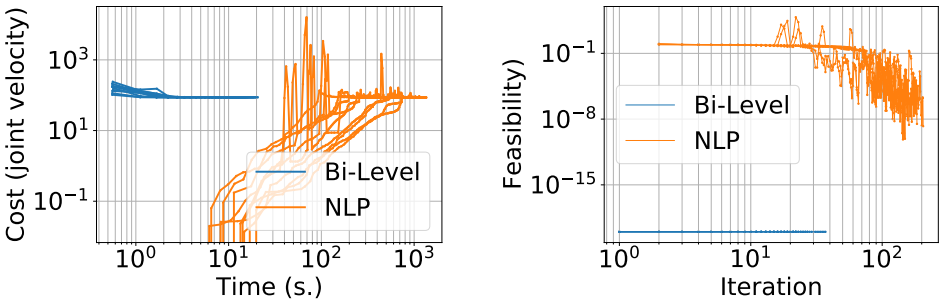
Method	Opt. cost	Opt. feas.	Time (s.)
Single NLP	**86.112** ± 0.0045	1.5e-08 ± 1.97e-08	814.702 ± 280.107
BiLevel	87.346 ± 0.0013	**0** ± **0**	**15.540** ± **5.870**
	Optimal design		Goal distance
Single NLP	{ 0.336 ± 0.0001, 0.375 ± 0.0002, 0.100 ± 0 }		**1.149e-19** ± **2.414e-19**
BiLevel	{ 0.308 ± 0.005, 0.418 ± 0.005, 0.100 ± 0 }		2.048e-06 ± 1.007e-07

Table 3: Comparison of a simultaneous formulation for co-design to our bilevel formulation using differentiable optimal control.

We noticed that the NLP formulation indeed reached the target with better precision that our DDP formulation. To make the comparison fair, we therefore increased the cost placed on the end-effector for DDP from $5e3$ to $1e6$. In the case of DDP, this cost is mixed with the rest of the cost regularization terms. Finally, we solved the NLP problem with IPOPT. Since IPOPT uses a BFGS approximation for the Hessian of the constraints, we also used finite differences for these terms, namely the derivatives w.r.t. design variables and their second-order tensors. Although this makes our approach not fully analytical, it is needed for a fair comparison due to the excessive overhead in calling JULIA from PYTHON for the computation of dynamics (Figure 17).

9 The solver used by the authors was KNITRO ([12]). In our experiments KNITRO was an order of magnitude slower than IPOPT (119 seconds versus 5.6 seconds) for solving the motion planning problem, so we chose the faster IPOPT for co-design.

We report the final optimization cost, feasibility (in terms of dynamics constraints and in the case of NLP, end-effector position constraint), optimal design, distance to the goal state and time of computation in Table 3. We ran the optimization for 25 different random initial designs and computed a mean and standard deviation over those trials. We further plot the (log) cost and the (log) feasibility versus time in Figure 19a and Figure 19b, respectively.



(a) Cost (joint velocity) over time

(b) Feasibility per iteration

Figure 19: Cost and feasibility over time and per iteration for simultaneous versus bi-level co-design formulations (10 out of the 25 trials plotted for readability).

These results show that formulating via a single nonlinear program leads to a more optimal solution in terms of cost and the end-effector reaching the goal state with less error (though for both methods in practice the error is orders of magnitude lower than the hardware resolution). DDP is always feasible, which is due to the forward dynamics integration, whereas the NLP formulation has some feasibility error even at convergence, though it is in the order of $1e-08$, which is acceptable. Most importantly, we observe a vast difference in computational time for both methods. Our approach using bilevel optimization and DDP is on average 54 times faster than the simultaneous approach for co-design. This result agrees with numbers reported in the literature. [84] reported that with 792 decision variables on a biped, their algorithm took between 460 and 685 seconds. In our experiments, the simultaneous approach took between 376 and 1350 seconds (on average 814 seconds) for 1497 decision variables (50 timesteps for a robot with 6 DoF together with 3 design variables and 50 constraints). The bi-level approach has 1491 variables (no final state constraint) and is also 54-times faster on average. These results demonstrate the potential in terms of computational speed of the bilevel approach. This, together with the fact that we always obtain a feasible solution for the motion (Figure 19b) shows the promise of bilevel approaches in interactive co-design tools. They can be interrupted, still returning a feasible solution and have a vast computational advantage compared to simultaneous (NLP) approaches.

We believe that the main reason for the slow computational speed of the single nonlinear program is that the design variable $\boldsymbol{\theta}$ influences all (nonlinear) constraints, both dynamics and the final goal state constraint. This nonlinear dependence on the design means that a step in the gradient direction of the design variable violates *all other* constraints. To illustrate this point, we plotted the sparsity structure of the Jacobian of constraints for the nonlinear formulation in Figure 20. We used $T = 10$ instead of the $T = 50$ used in the experiments for readability of the figure. The last three rows in the figure represent the design variables (link lengths), which influence

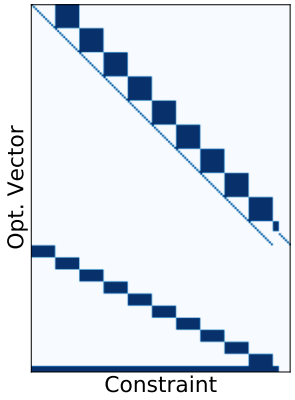


Figure 20: An illustration (for $T = 10$ for readability) of the sparsity structure of the Jacobian of constraints for the nonlinear programming formulation. The last row represents the design variables, which are present in all constraints except the final velocity constraint.

all other constraints, except for the constraint that the final velocity of the robot is zero.

Still, one of the remaining challenges is how to efficiently compute the required first- and second-order dynamics derivatives w.r.t. design variables in practice. We discuss this next.

4.6 DISCUSSION

We presented two ways of computing the analytical derivatives of the solver – (i) using algorithmic differentiation and a URDF model created via code (Section 4.4.4) and (ii) using finite differences (Section 4.5). Both of these are used to compute the terms $\boldsymbol{f_{xx}}$, $\boldsymbol{f_{xu}}$, $\boldsymbol{f_{uu}}$ as well as $\boldsymbol{f_{x\theta}}$, $\boldsymbol{f_{u\theta}}$ and $\boldsymbol{f_{\theta}}$. As we show in Figure 17, we used RigidBodyDynamics.jl ([52]) and ForwardDiff.jl to compute these via automatic differentiation or simply used a two-sided finite differences approximation in the second case. This was needed as these terms are not readily available analytically in rigid body dynamics libraries. When using finite differences, our approach takes on average 10.9 ± 4.144 seconds to converge, whereas when using derivatives via automatic differentiation, it takes 187.403 ± 81.453s.[10] The majority of this time is due to the computation of the dynamics derivatives and in fact for iLQR derivatives, which exclude the second-order tensors the time is 179.758 ± 73.462s, that is on average only 7.7 seconds quicker. As such, if we could compute these derivatives quickly via rigid body dynamics libraries, this would greatly benefit the computation time of our algorithm and as such is an exciting direction for future research.

One approach is to code generate these derivatives, as done by Spielberg *et. al.* [84]. However, this process is computationally expensive and if multiple morphologies are to be co-designed, needs to be repeated for each one. Moreover, not all tools can handle the complexity of using symbolic derivatives as the resulting expression leads to text files that are megabytes large. In practice, RigidBodyDynamics.jl [52] in our

10 This is still in fact on average 3.6 times faster than the simultaneous formulation.

experience could not compute the derivatives of the Kinova arm in this way and hangs the Julia interpreter.

In fact recently in the robotics community there have been results showing the benefits of using second-order dynamics in DDP to solve OC problems. This involves the terms $\boldsymbol{f_{xx}}$, $\boldsymbol{f_{xu}}$, $\boldsymbol{f_{uu}}$, and an efficient way to compute the tensor product in Eq. (39) [81, 82]. A direction for future work is the use of the efficient algorithms developed in [82]. Moreover, we still lack efficient algorithms for computing derivatives w.r.t. design variables including second-order derivatives and their tensor products in Eq. (61) – $\boldsymbol{f_{x\theta}}$, $\boldsymbol{f_{u\theta}}$ and $\boldsymbol{f_{\theta}}$. Future work lies in developing such algorithms in order to enable online deployment of differentiable solvers for learning and system identification on real robots.

Finally, in the next chapter we present hardware experiments on the Solo robot using our co-design framework. We compare the nominal Solo robot design to an optimized design for trotting that minimizes the peak torque.

5 HARDWARE VALIDATION

In this chapter we describe the hardware experiments we performed on the Solo robot [35], pictured here in Figure 21. The goal of this chapter is to experimentally test the improvements resulting from co-design on real hardware as well as to validate our framework including the motion planning. We also need to evaluate the motion planning because if the robot can not execute the motions, then improvements in co-design metrics are meaningless. We thus also evaluate how well the robot tracks the motions that were planned by the motion planning part of the co-design algorithm. The tracking and the improvements in co-design cost are connected as the co-design cost is defined on the motions of the robot. We thus have three concrete research questions – (i) is the robot, qualitatively, able to execute the motion, (ii) how well does the robot track the motion plan (quantitatively, in terms of tracking error) and (iii) what are the improvements in co-design cost and how do they compare to the results obtained in simulation?

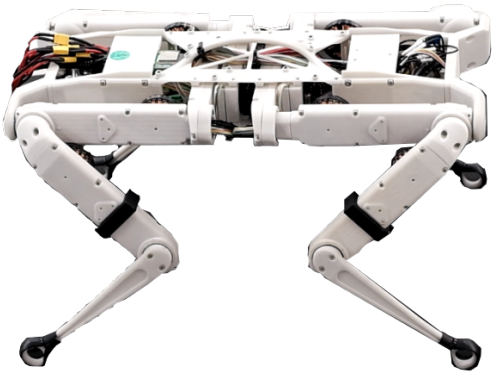


Figure 21: The Solo 12 Robot. This robot is almost entirely 3D-printed (apart from the gear assembly). The lower legs specifically are easy to modify as they have no moving parts. A supplementary video of the robot motions is available at https://youtu.be/avYdwbcu_vY.

5.1 EXPERIMENTAL DESIGN

For this chapter, we will use the framework presented in Chapter 3 using the CROCODDYL [58] motion planner. During some preliminary work on the co-design formulation we discovered a relationship between the peak torque of the control inputs and the lower leg length for trotting. Minimizing peak torque is of practical interest, as it allows the robot to carry e.g. heavier loads due to the controls being reduced below the control limits of the robot[1].

1 Although minimizing energy does also minimize peak torque, in our experiments this effect was much less (e.g. for a step length of 3 cm, the peak torque was reduced by 2.2% when minimizing energy and 14.5% when minimizing peak torque directly).

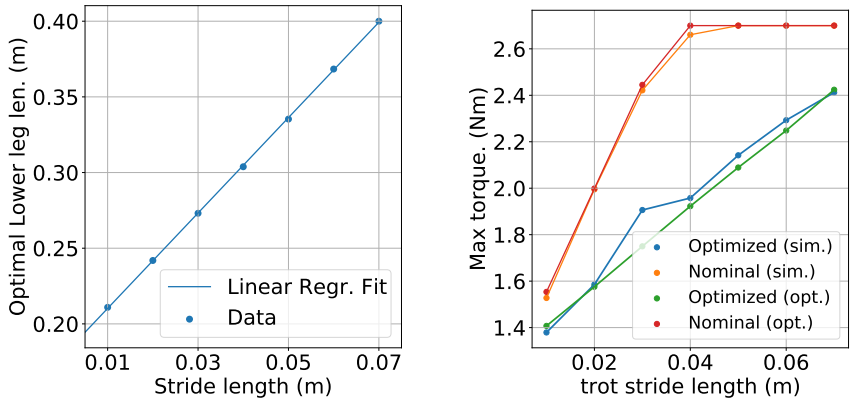
Recall that the peak torque cost involves a 'max' operator and is non-differentiable. We thus used an smooth maximum approximation in the upper-level during optimization:

$$J_{\text{UL}}(\boldsymbol{X}, \boldsymbol{U}, \rho) = \frac{1}{\alpha} \log \left(\sum_t \sum_i \exp^{|\alpha \boldsymbol{u}_t^i|} \right) \tag{70}$$

This cost sums over timesteps as well as joints (indices t and i, respectively) and is an approximation to the maximum torque the robot uses. The parameter α determines how close to the true maximum the approximation is. We used $\alpha = 100$ in our experiments. Finally, once the optimization is complete, we used the true peak torque to report the co-design improvements.

We then used a fixed step height of 0.045 m for a trotting task and we optimized the lower leg length (same for all legs) over a range of trotting step lengths – $[0.01, 0.02, \ldots, 0.07]$m for two steps taken. The mass of each lower leg is modeled to be linearly proportional to the nominal mass of 12.06 g at 19 cm and the inertia is modeled as a cylinder with a constant radius and the corresponding leg length and mass.

We plotted the resulting optimal lower leg lengths versus the stride length and fitted a linear regression model to the data (Figure 22a). The fit has an R-squared score of 0.99989 indicating that the relationship between the step length and the optimal lower leg length is almost exactly linear. Additionally, we plotted the maximum torques both in simulation and optimization (using the true maximum function) in Figure 22b). For the optimal design we notice that the robot has a maximum torque at the torque limits for stride lengths greater than 0.4 m.

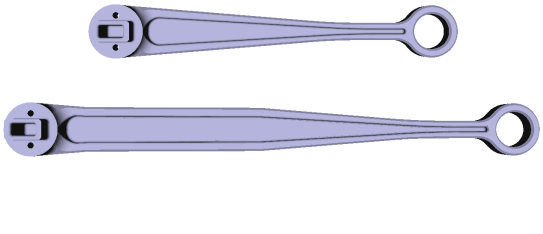


(a) Optimal lower leg length minimizing peak torque for different stride lengths. We fitted a linear regression, showing that this relationship is linear.

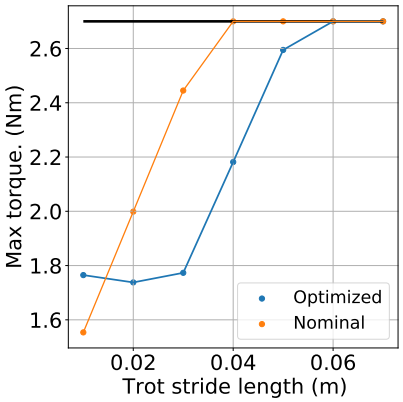
(b) Simulation and optimization results, showing maximum torque (the cost function) both from the optimizer and in simulation with a PD controller.

Figure 22: Optimization results for trotting.

Given this information, we chose to print a lower leg length of 0.27 m. This is the longest leg that we could physically print with our equipment. Any leg length above 0.4 m will have less improvement of cost for its optimal stride length as the nominal design is at the robot's torque limits in Figure 22b.



(a) Nominal and optimized longer leg design for peak torque. We designed the longer leg to be wider near the attachment point for more stability.



(b) Optimized (0.029 m) lower leg length versus nominal (0.018 m); peak torque for trotting in optimization. For longer trots than 0.01m. we see improvement until we reach the robot's torque limit at 2.7N.m..

Figure 23: Leg design for the optimal leg and peak torque reduction (in optimization) for different gaits.

5.2 RESULTS

The nominal and optimized designs are pictured in Figure 23a. We extended the robot leg, but we did not uniformly vary the thickness. Instead, we made the leg wider at its attachment point so that we ensure structural stability in the longer leg and avoid bending. Specifically we extended the 3D-model of the lower leg by cutting it at its widest and then extending the wide edge in order to achieve the desired length. Next we determined using the linear regression (plotted in Figure 22a) that the optimal stride length for this design is 0.029 m. We additionally explored different stride lengths (with motion planning) to see what torque improvement there is across stride lengths using the leg that we printed. These results are in Figure 23b and in fact we do observe peak torque reduction for this longer leg length across stride lengths with the biggest improvement being at the optimal stride length of 0.029 m. The nominal leg length (0.018 m) for the Solo robot is optimal for very short stride lengths, which is why we see it leads to less peak torque then (for stride lengths shorter than 0.02 m.

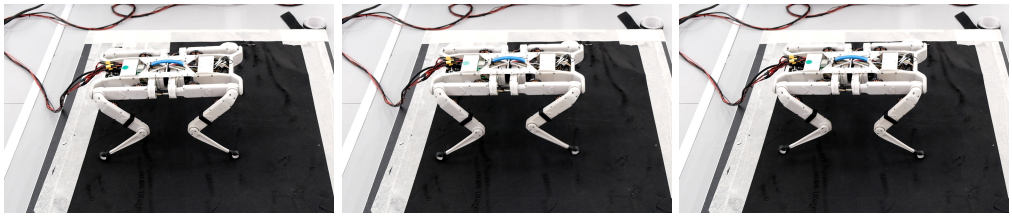


Figure 24: Snapshots from the robot trotting with its nominal lower leg design.

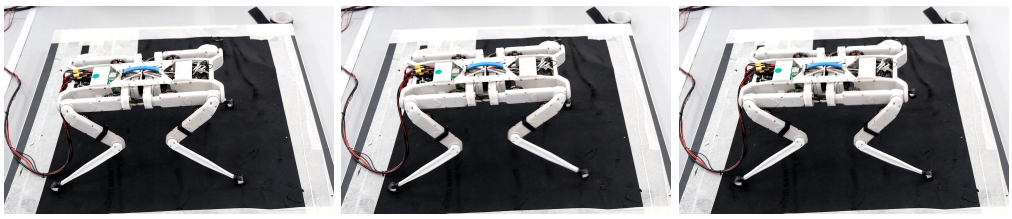


Figure 25: Snapshots from the robot trotting with the optimized lower leg design.

We next ran $N = 5$ experiments for each of the nominal leg and the longer leg on the robot. We used the same joint proportional-derivative controller that was tuned in simulation via a grid search as described in Chapter 3:

$$\boldsymbol{u} = \boldsymbol{u}^* + \boldsymbol{K_p}(\boldsymbol{q}_j^* - \boldsymbol{q}_j) + \boldsymbol{K_d}(\boldsymbol{v}_j^* - \boldsymbol{v}_j), \tag{71}$$

where $\boldsymbol{q}_t$ are the joint positions at time t, $\boldsymbol{q}_t^*$ – the optimal reference positions, $\boldsymbol{v}_t$ – the joint velocities and $\boldsymbol{v}_t^*$ – the optimal reference joint velocities, $\boldsymbol{u}^*$ – the optimal feedforward torque, $\boldsymbol{K_p}$ – the optimal position (proportional) gains and $\boldsymbol{K_d}$ – the optimal velocity (derivative) gains.

Here we also show snapshots of the first step of the robot for both designs – Figure 24 for the nominal design and Figure 25 for the optimized longer lower leg.

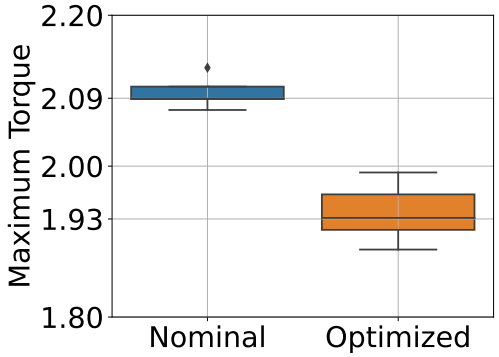


Figure 26: Boxplot of maximum torques for $N = 5$ experiments for each nominal and optimized robot design on the real hardware.

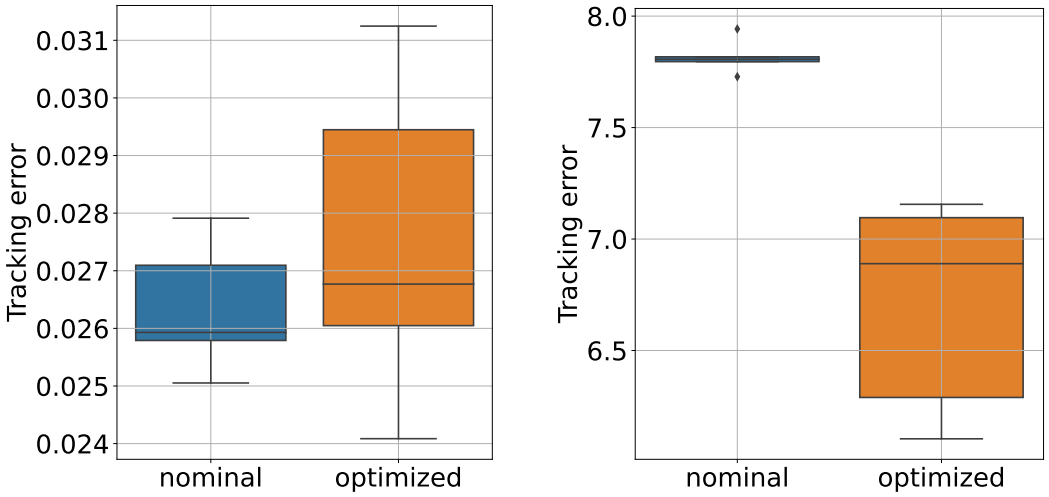
We further computed the maximum torques for both designs as recorded by the motor driver board and plotted them in two boxplots in Figure 26.

5.3 ANALYSIS

Qualitatively both designs are able to trot successfully for two steps. However, the nominal design seems more stable than the optimized one. We see some bending in the longer legs as well as more slipping and more rotation of the base. These effects are more visible in the video of the motions (`https://youtu.be/avYdwbcu_vY`).

To analyze the results quantitatively, we plotted the peak torque in Figure 26 and computed the average improvement in peak torque. In optimization the peak torque

improvement was 27.829%, in simulation it was similar at 28.271%. However, on the real robot, on average the improvement was 7.609%, which is less than both optimization and simulation. Still, we always observed a reduction in peak torque for the optimized design in all of our experiments. This shows that indeed the optimized design does reduce the maximum torque, however on the real robot the effect is less than in optimization or simulation.



(a) Position tracking error.

(b) Velocity tracking error.

Figure 27: Tracking errors on the real robot for both nominal and optimized designs.

In order to study further the reasons for this difference, we plotted the tracking error for both nominal and optimal designs in Figure 27. In the left figure we plotted the position error and in the right – the velocity error, namely:

$$E_{\text{POS}} = \sum_t |\boldsymbol{q}_t - \boldsymbol{q}_t^*|^2, \quad E_{\text{VEL}} = \sum_t |\boldsymbol{v}_t - \boldsymbol{v}_t^*|^2 \tag{72}$$

where E_{POS} is the position error, E_{VEL} is the velocity error. We can see that velocity tracking is significantly better for the optimized design, however position tracking has a larger standard deviation compared to the nominal design. To examine this further, we look at the gains of the PD-controller.

These gains were tuned in simulation over a grid search to minimize the summed tracking cost $E_{\text{POS}} + E_{\text{VEL}}$. Inspecting the optimal values we find that for the nominal design $\boldsymbol{K_p} = 3.111, \boldsymbol{K_d} = 0.182$ and for the optimized design $\boldsymbol{K_p} = 5.222, \boldsymbol{K_d} = 0.299$. With significantly larger gains it is not surprising that the optimized design has less velocity tracking error as the controller is more aggressive in correcting for it. However, the optimized design also has similar position tracking error to the nominal design, but with a larger gain on it. This does explain partially the difference in torque – with worse tracking, but a higher gain the controller for the optimized design will command higher torques to compensate.

To finish our analysis, we plotted an example of the control inputs from the simulator for both designs (Figure 28) and the same for the real robot (Figure 29). In addition we isolated the joint that exhibited the peak torque and plotted the simulation and

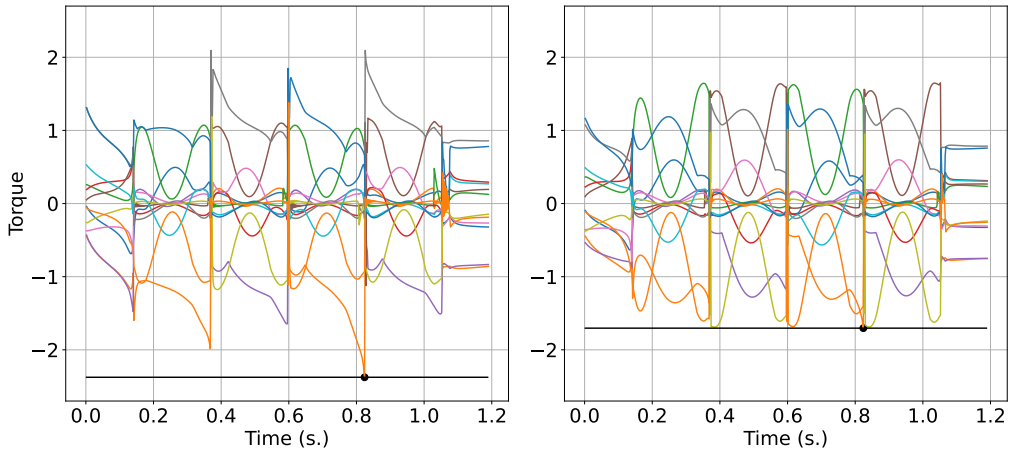


Figure 28: Control results for both designs (simulation).

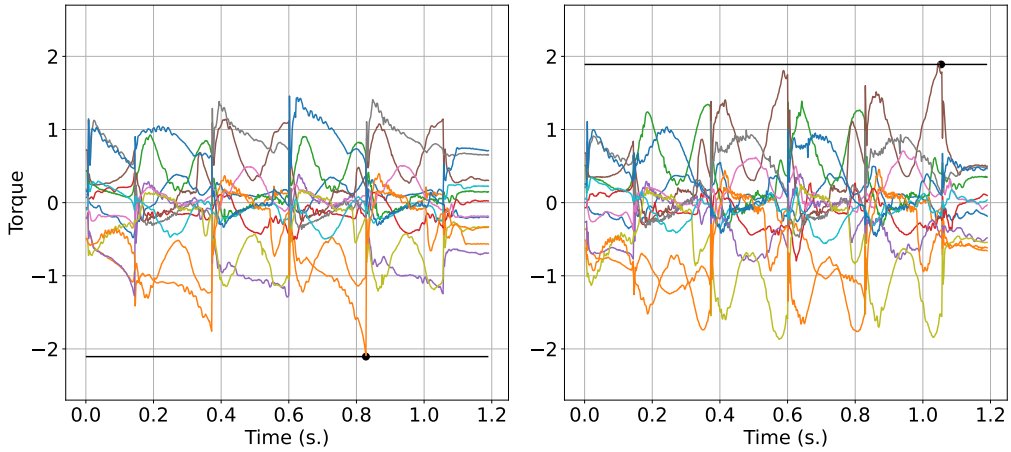


Figure 29: Control results for both designs (hardware).

hardware results on one plot for both designs (Figure 30). These plots give us valuable insight into exactly what the difference between simulation and reality is. In simulation we can see that the nominal design leads to controls with peaks in torque greater than 2N.m.. These peaks are observed at every contact the robot makes with ground. The optimal design reduces those peaks in torque, however it does so at the cost of more control input over time. This makes sense as still the same amount of energy is needed for the same motion – in fact for the optimized design more energy would be needed, as the legs are also heavier because they are longer.

On the real robot (Figure 29), however, we see that the controller for the nominal design does not track the commanded torque peaks well and in fact has a similar smoothing effect. In this particular case this does not lead to the robot falling over, however it does reduce the peak torque, which is our metric. The controller for the optimized leg length largely accomplishes reducing those peaks in controls and smoothing the control profile.

To quantify this we can look at the difference in peak torque between simulation and on the real robot. On average the nominal design exhibits 0.301N. less peak torque

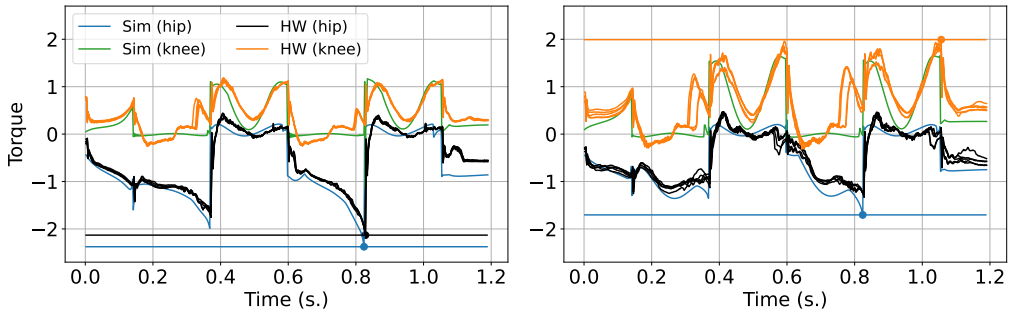


Figure 30: Control results for the nominal (left) and optimized (right) designs (simulation and hardware). We highlight the peak torque with a horizontal line. For the nominal design (left) the peak is less than in simulation and for the optimized design (right) the peak appears at a different link and is more than in simulation.

across all experiments and the optimized design exhibits 0.207N. more. Hence indeed what is happening is that the nominal design leads to a control profile that is not tracked well on the real robot with the peaks being smaller and the optimized design is tracked better, however at the cost of a smaller increase in peak torque because of the higher gains. Both of these effects together are what lead to the difference we see between simulation and reality. We note here that these gains are necessary for the robot to trot and are not interchangeable – the optimized designs with lower gains fails even in simulation and the nominal design with higher gains leads to jerky and noisy control inputs. These gains were tuned to be optimal in that sense and are a function of both the motion and the design.

5.4 CONCLUSIONS

The experiments we presented in this chapter provide insight into applying co-design on real hardware. Firstly, we want to answer the research questions as posed at the start of the Chapter.

- *Does the robot trot?* The robot qualitatively is able to trot for both designs – it does not fall over in any of the experiments and is able to successfully execute the motion plan.
- *How well does the robot track the motion plan?* There is tracking error for both designs. Although the optimized design has lower tracking error, this can be explained by the higher gains needed by the controller, rather than a property of the design itself.
- *What are the design improvements in hardware and how do they compare to optimization and simulation?* On the real robot there is a cost improvement. On average the improvement in cost was 7.609%, which is smaller than in optimization and simulation.

While both designs (nominal and optimized) were able to perform the motion successfully, the improvements in co-design cost were reduced (though we always

observed an improvement in peak torque). This is due to the following factors that in future work need to be taken in consideration:

- When modeling link lengths, such as the lower legs of the robot, we need to include a factor that scales the thickness of the legs for a given material to avoid bending. This was the case for the longer legs we printed saw bending which led to poorer tracking.
- We observed significant slipping in our experiments, even though we worked on a high-friction surface and had rubber attachments to the legs. This is difficult to mitigate. Legs with built-in high-friction surface at the contact location are a way to address this. The rubber bands we used were only attached via zip-ties and were not fixed to the bottoms of the legs.
- Both of the above additionally suggest that a more sophisticated controller is necessary – in particular one that is based for instance on model-predictive-control (MPC). Such a controller can compensate for the deviations from the model we use in optimization (we discuss such a controller for a different task and robot in Chapter 7).
- Finally, the actuators of the robot might not be able to track the control profile that is requested by the motion planner or the controller. To account for this actuator modeling during the optimization process is necessary. This will prevent the motion planner from generating trajectories that can not be physically tracked (e.g. Figure 29). In the case of minimizing properties of the torque (like in this chapter) this will be beneficial as torque output is directly related to the motor and how well it can output for instance torque peaks.

The modeling of links bending fits into our framework as additional constraints and implicit relationships e.g. between thickness and leg length. The slipping can be included in the motion planning or it can be reduced physically by building more structurally stable lower legs. Lastly, the actuator models can be included in our framework as differentiable functions that map commanded torque to applied torque at the joints without any other modifications to the robot model.

Finally, this chapter concludes our discussion on co-design and the first part of this thesis. In *Part 2*, we present our work on motion planning. In the next chapter we present our work on differential dynamic programming (DDP) that introduces sparsity costs on controls and presents an approach to tuning their weights. This allows the solver to make discrete choices in control-space by enforcing sparsity on the control variables. We show two applications of this – (i) for satellite thruster control and (ii) for automatic joint selection on the Valkyrie humanoid robot. Controlling satellite thrusters is related to choosing contact forces, but simpler, as thrusters do not have the constraint of having to be in contact with the ground when they fire.

Part 2
Motion Planning

6 SPARSITY-INDUCING OPTIMAL CONTROL VIA DIFFERENTIAL DYNAMIC PROGRAMMING

Throughout Chapters 3–5 we focused on systems with predefined contact timings and locations, which is a common assumption in the literature [59, 60]. In our co-design approach we assumed fixed, pre-specified contact locations and timings. Ideally we would like the motion planner to choose the contacts instead. This gives the solver more flexibility, as different contact timings and locations might be optimal for robots with different designs. Additionally, optimizing the gait parameters in the motion planning layer can improve the co-design process, as the design is only optimal with respect to (w.r.t.) the motion plan itself. Pre-specifying the timing and location still has its applications in the cases where the user has specific constraints on those parameters, but is less general in other cases.

Choosing contacts is a discrete problem which requires selecting surfaces and timings – i.e. when and where to place the feet. This leads to sparse control inputs – that is control inputs that are exactly zero when the feet are not in contact in the ground. In this Chapter we study how to achieve sparsity in control-space using differential dynamic programming (DDP) for two related, but simpler problems and systems.

The first system is a satellite, where the thrusters, which are similar to legs, need to be controlled in a discrete fashion. This is a simpler problem than selecting contact locations and timings for a quadruped, but it is related as thrusters are similar to legs – they can be either on or off, much like a quadruped can be in contact or not in contact. However they are simpler to control, since there is no constraint to be in contact with the ground.

The second application is an automatic joint selection task for the Valkyrie humanoid robot, where we can use sparsity in control space to automatically select the required number of joints to be moved for a low dimensional task.

To solve these problems we propose to use a sparsity-inducing penalty term in the control cost. This serves to both switch off unnecessarily small control inputs in the case of high-degrees of freedom (DoF) robots and to discover thruster-like behavior for satellites.

6.1 RELATED WORK

A common optimality criterion for control problems is energy optimization (Chapter 2 as well as Chapter 3), which is traditionally a squared cost on the control inputs (L_2 norm regularization). In practice, this leads to smooth control profiles and has been widely applied to canonical dynamic systems, computation of flight trajectories, as well as to synthesize highly dynamic maneuvers for legged robots [58, 68]. However, as no sparsity is introduced, on redundant systems this frequently leads to moving many joints even if not all joints are required to complete the task (cf. Figure 39).

Sparsity in control inputs for planning was studied in the context of satellite motion planning [43]. There the authors used the L_1 norm, also known as *Lasso* model. Similarly, the authors in [53] applied an L_1 penalty using an alternating direction method of multipliers (ADMM) approach separating the problem into an optimal control update and a soft thresholding update. Whereas previous work considered an L_1 penalty applied to the force at the center of mass of the satellite, here we model the thruster behavior directly. Finally, the authors in [57] obtained thruster controls by optimizing the timing of thruster pulses, which are modeled as on-off controls. Here we use smooth L_1 costs to penalize the otherwise continuous thruster forces. As a result, our approach is directly applicable in standard optimal control frameworks.

Recently, the concept of sparsity has gained additional attention in the trajectory optimization and control community for terrestrial/traditional robotics applications such as manipulation. [45], for instance, investigated the use of Mixed-Integer and Lasso regression to reduce joint motion on humanoid robots in a hierarchical inverse dynamics control scheme. Nonetheless, enforcing sparsity for planning over longer horizons continues to be a challenge.

It is well known in the machine learning community that L_1 introduces a discontinuity at 0 that makes it non-differentiable [79]. Several differentiable metrics have been proposed to deal with this issue [46, 53]. In this chapter, we study two such costs when applied to DDP: the SmoothL1 [28] and Huber differentiable approximations [46] to L_1. Using such sparsity-inducing cost terms with DDP raises two challenges requiring careful consideration: i) ensuring numerical stability/conditioning as efficient implementations assume positive-definiteness of the control Hessian, and ii) trading off achievements of desired tasks with sparsity of solutions through control regularization. Note that these challenges do not arise when using direct transcription/collocation where tasks are enforced with hard constraints and solved using off-the-shelf nonlinear programming (NLP) solvers.

6.2 CONTROL REGULARIZATION

We begin by reviewing the use of L_2 penalties in the optimal control literature. The L_2 loss is defined as:

$$L_2(x) \triangleq x^2.$$

The L_2 loss is used to regularize solutions by penalizing large positive or negative control inputs in the optimal control setting or features in machine learning.

In contrast, the L_1 loss is used to penalize solutions for sparsity, and as such, it is commonly used for feature selection in the machine learning community [88]. The L_1 loss is defined as the absolute value of its argument:

$$L_1(x) \triangleq ||x||$$

When L_1 is used with gradient-based optimizers as a regularization, it drives its argument to exactly 0 as opposed to small values. This is explained by the condition that the gradients of the regularization parameter and the task cost must be parallel.

Using the L_1 loss directly in gradient-based optimization is difficult due to the discontinuity at $x = 0$ where the gradient is undefined. Smooth approximations to the

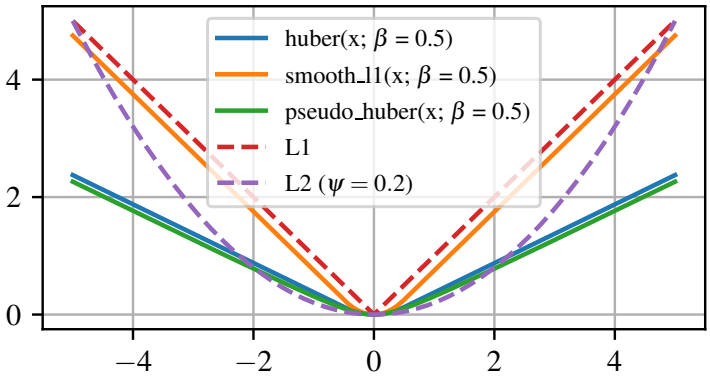


Figure 31: Comparison of the different regularization schemes for the control cost using a range of hyper-parameters: L_2, L_1, $L_{1\text{-smooth}}$, L_{Huber}, and $L_{\text{pseudo-Huber}}$.

L_1 function can be used in place of the true L_1 penalty. In this chapter, we consider a smooth L_1 function from [28], which combines L_1 and L_2 losses, defined as:

$$L_{\text{smooth-l1}}(x) = \begin{cases} 0.5x^2/\beta & \text{if } ||x|| \leq \beta \\ ||x|| - 0.5\beta & \text{otherwise} \end{cases} \tag{73}$$

where β defines where the function switches from an L_1 to an L_2 cost. For small $x \leq \beta$, *SmoothL1* switches to L_2, since L_2 has a gradient at 0.

We study another variant of combining L_1 and L_2 regularization, namely the Huber loss [46]:

$$L_{\text{huber}}(x) = \begin{cases} 0.5x^2 & \text{if } |x| \leq \beta \\ \beta(||x|| - 0.5\beta) & \text{otherwise} \end{cases} \tag{74}$$

where β is again a shape parameter. The Huber loss has a variable slope, controlled by β in addition to mixing L_1 and L_2. This can be seen in Figure 31, where for $\beta = 0.5$ the Huber cost has a lower slope than SmoothL1.

Finally, we consider a smooth approximation to the Huber loss, the Pseudo-Huber loss, as defined in [44, Appendix 6]:

$$L_{\text{pseudo-huber}}(x) = \beta^2 \left(\sqrt{(1 + \frac{x^2}{\beta})} - 1 \right). \tag{75}$$

We illustrate the considered losses for different settings of their hyper-parameters in Figure 31. The shape parameters of the smooth variants control how closely they approximate the true L_1 and Huber losses, respectively. The choice of the control regularization and its parametrization has an impact on convergence and sparsity of the output of the optimal control formulation.

6.3 Optimal Control

We consider the robot as a dynamic system described by state $\boldsymbol{x}$ composed of generalized coordinates $\boldsymbol{q}$ and generalized velocities $\boldsymbol{v}$. The system evolves under applied

control inputs $\boldsymbol{u}$ according to the state transition function $\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t)$ which incorporates the differential dynamics as well as an integration scheme. Here, we use a geometric representation of the configuration manifold of floating-base systems ($\mathbb{SE}(3)$) with its geometric integrators along with an energy-conserving symplectic integration scheme of the differential dynamics.

To describe a discrete optimal control problem with a fixed horizon, we additionally specify the integration time step Δt and time horizon T and the number of discretization knots N. This yields a state trajectory $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ and control trajectory $U = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{N-1}\}$. Tasks and constraints are enforced by minimizing a cost function:

$$J(\boldsymbol{X}, \boldsymbol{U}) = h(\boldsymbol{x}_N) + \sum_{t=1}^{N-1} \ell(\boldsymbol{x}_t, \boldsymbol{u}_t). \tag{76}$$

Shooting methods in particular minimize $J(\cdot)$ with respect to control inputs only:

$$U^* = \underset{U}{\operatorname{argmin}} \; J(\boldsymbol{X}, \boldsymbol{U})$$

where U^* is the optimal open-loop control trajectory. The corresponding state trajectory is obtained by performing a forward roll-out using the state transition function.

In this chapter we will be using differential dynamic programming (DDP) to solve the above optimal control problem. For a thorough explanation of how DDP works, we refer the reader to Chapter 4. In this chapter we will not be modifying the internal DDP algorithm, but rather we will be introducing sparsity cost terms to the formulation.

6.4 Enforcing Sparsity with L_1 and Huber Costs

DDP minimizes a general cost function $J(\boldsymbol{X}, \boldsymbol{U})$ of the form in (76). In this work, we propose adding an additional cost term for each control input $\boldsymbol{u}_t$ that induces sparsity. The new cost function thus becomes:

$$J(\boldsymbol{X}, \boldsymbol{U}) = h(\boldsymbol{x}_N) + \sum_{t=0}^{N-1} \left[\ell(\boldsymbol{x}_t, \boldsymbol{u}_t) + \psi \; l_s(\boldsymbol{u}_t)\right] \tag{77}$$

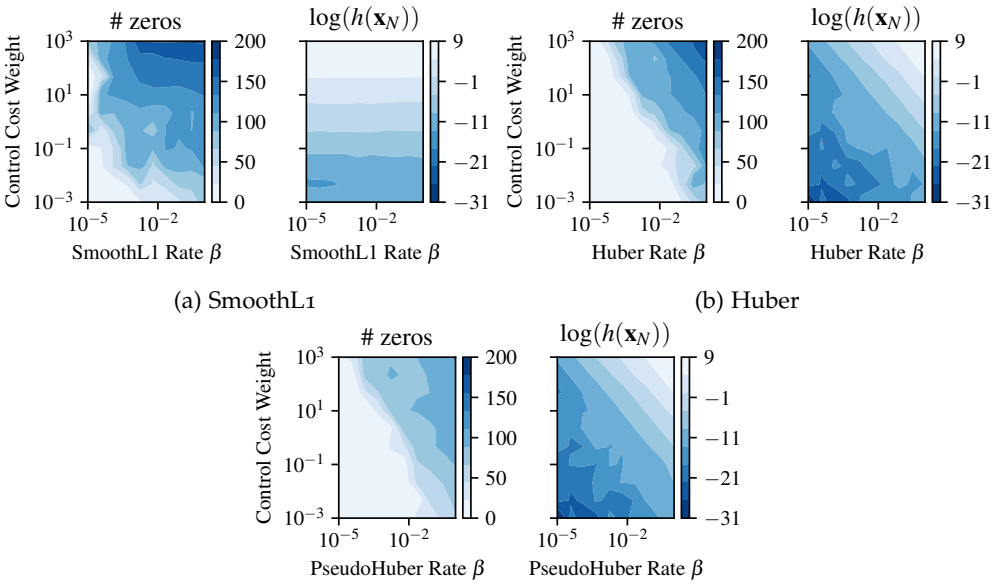
where $l_s(\boldsymbol{u}_t)$ is one of the sparsity-inducing losses described in Section 6.2 and ψ is a strength parameter, which controls the relative effects of the regularization loss and the objective loss.

For the cartpole and satellite examples, we use quadratic state costs of the following form:

$$\begin{aligned} h(\boldsymbol{x}_N) &= (\boldsymbol{x}_N - \boldsymbol{x}^*)^T Q_f (\boldsymbol{x}_N - \boldsymbol{x}^*) \\ \ell(\boldsymbol{x}_t, \boldsymbol{u}_t) &= (\boldsymbol{x}_t - \boldsymbol{x}_t^*)^T Q (\boldsymbol{x}_t - \boldsymbol{x}_t^*) \end{aligned} \tag{78}$$

where Q is a diagonal weighting matrix for each of the NDX dimensions of $\boldsymbol{x}_t$ and the matrix Q_f is a weighting term for $\boldsymbol{x}_N$, respectively. For the Valkyrie example, we demonstrate the use of nonlinear task cost functions such as end-effector position, stability cost, and joint limits from [47].

The parameters ψ together with β are hyper-parameters and their values define the interaction between the sparsity loss and the optimization criterion (task). We study their effect on the convergence of the solution in detail using the canonical cartpole in the following section.



(a) SmoothL1

(b) Huber

(c) Pseudo Huber

Figure 32: Comparison of different sparsity-inducing control cost terms on the cartpole system: Sparsity of solutions and final costs for a grid-search over hyper-parameters β (shape) and ψ (strength). Darker color indicates more sparsity and lower final cost respectively.

We firstly study the effects of sparsity-inducing costs on a one-dimensional problem—the swing-up of a cartpole, which is a canonical optimal control problem where a pendulum is attached to a cart moving on an infinite friction-less track. The goal is to swing the pendulum upright and move the cart to the origin. The problem is underactuated—the control inputs are linear forces on the cart, whereas the pendulum joint is not controlled. The time horizon is $T = 200$ with $\Delta t = 0.01$s, resulting in a 2 s trajectory. The control limits are ± 30N and $Q_f = 100\ \mathbb{I}_4$, where $\mathbb{I}_4$ is the 4×4 identity matrix.

6.5.1 *Effects of weight and shape parameters on sparsity*

Firstly, we examined the effect of the weighting term ψ and shape parameter β on sparsity. β for all functions controls where the switching between an L_2 cost (for small $x < \beta$) and an L_1 cost (for $x \geq \beta$) occurs. We consider controls to be zero when they are within $[-\beta, \beta]$.

The results of a grid search over β and ψ are in Figure 32a for SmoothL1, Figure 32b for Huber, and Figure 32c for PseudoHuber. We plotted both the number of zeros and the final task cost—the latter tells us how close we are to the goal state.

A desired property of sparsity costs is that sparsity should increase with the weight term ψ. As expected, we see a trade-off between sparsity and task cost—the more regularization, the more sparsity, however at a higher task cost. This is in fact what the grid search shows. Another important property we observe is that for lower tolerances β, a much higher weight is required to achieve sparsity. Thus we can extract a criterion

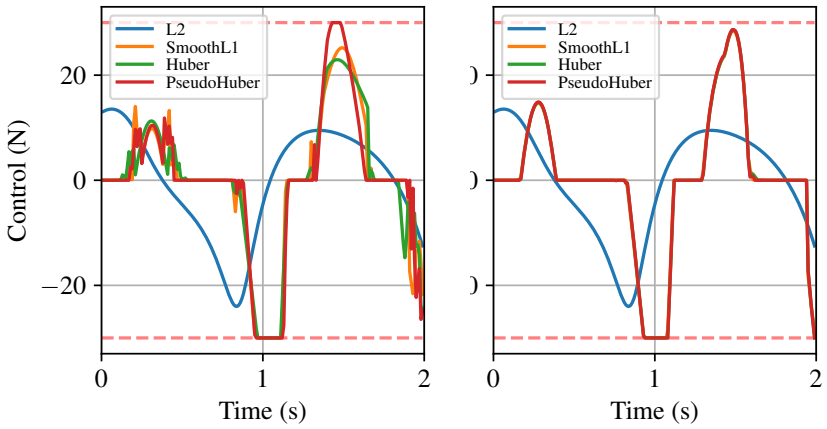


Figure 33: Sparse control solutions for the cartpole system: Left with control artifacts and right with smooth controls.

for choosing sparsity—1) pick the largest β parameter according to how much noise tolerance the system has and 2) adjust the control cost weight until the desired amount of sparsity is achieved. The noise tolerance of the system is the maximum value of controls beyond which rapid fluctuations can not be tolerated.

In the solutions for $\beta = 10^{-3}$ in Figure 33(left), all solutions achieve a final task cost of less than 10^{-5} with 73, 58 and 86 zero controls for SmoothL1, Huber and PseudoHuber, respectively, yet we observe artifacts showing rapid control changes. Increasing the weight (from 7, 9, and 0.01 to 25, 25, and 0.023 for Huber, PseudoHuber and SmoothL1, respectively) can reduce these artifacts, while also increasing sparsity. In Figure 33(right) the solutions have 101, 102 and 89 zero controls and produce smoother control profiles. However, this comes with an increase of final state cost from less than 10^{-5} to less than 10^{-4}.

In general, sparsity-inducing costs together with control/actuator limits produce so-called "bang-bang" control. On a real system, aggressive bang-bang control requires the actuator to change the output torque rapidly at each time-step. This is usually not possible due to actuator dynamics, for example, when using electric motors on the cart pole in our example. However, in some domains, namely satellite control, the underlying physical system and actuators are only capable of bang-bang control and this in fact is a desirable property.

6.6 THRUSTER CONTROL FOR SATELLITES

The propulsion systems of orbital satellites have a unique control limitation. In many cases, they use impulsive cold gas or bi-propellant thrusters incapable of or ineffective at low rates of firing. The resulting control then relies on fewer longer bursts of thrust to generate a constant amount of force over time while keeping the thrusters off between the bursts.

The required control inputs are then a sequence of binary on/off commands that are activated sparsely throughout the motion [57]. We call this type of control *sparsity-inducing*, referring to the sparse use of control inputs throughout the trajectory. Such control will prefer zero control (off) followed by a high control action (on) to continuous corrective commands applied throughout the whole trajectory.

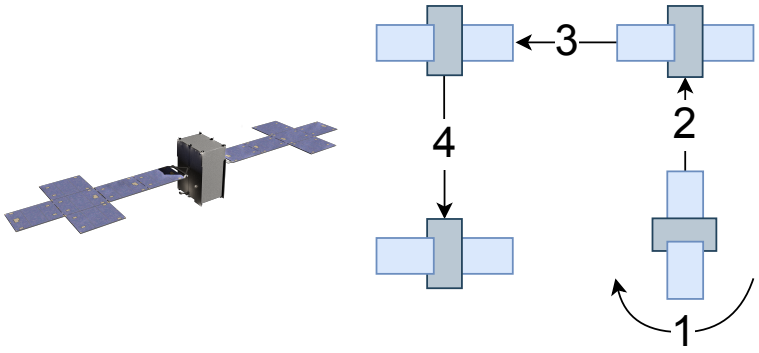


Figure 34: SSL-1300 satellite model and satellite maneuver.

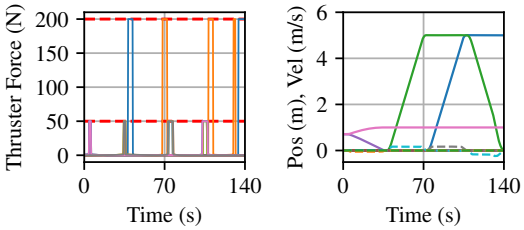


Figure 35: Satellite thruster trajectory (SmoothL1) and corresponding state trajectory.

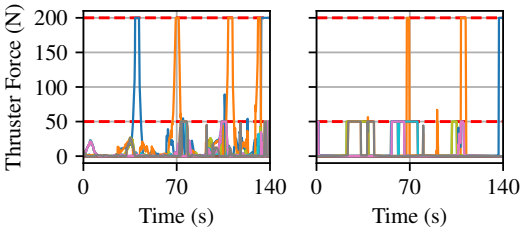


Figure 36: Satellite control trajectories for weights 10^{-5} and 10^{-1} and a PseudoHuber loss.

We now consider a satellite as a floating rigid body in $\mathbb{SE}(3)$ actuated through forces produced by two primary propulsion thrusters on the front and back of the satellite and four steering thrusters on each of its remaining four sides ($NX = 13$, $NU = 18$). We generated a center of mass tracking trajectory with 4 discrete stages solved in a single optimization problem, as illustrated in Figure 34. The problem was discretized with $\Delta t = 0.1$ with $T = 1400$ knots for a 140-second trajectory. Thruster forces were constrained in $[0, 200]$ N for the front and back and $[0, 50]$ N for the side thrusters, since there are four for each direction.

The resulting control (thrust) and state trajectories are shown in Figure 35. The different colors correspond to different thrusters being activated at the corresponding times. We see thruster peaks at the start and end of each of the stages and reach the corresponding set output forces. The right side of the figure shows the position and linear velocity trajectories in state space.

6.6.1 *Effects of weight parameters*

We next examine the effects of the different weights on the satellite problem. In Figure 36 we plotted the control trajectories for different weights—$\psi = 10^{-5}$ and $\psi = 10^{-1}$. The plot for the correctly tuned $\psi = 10^{-3}$ is in Figure 35. We see a similar trend as in the cart-pole system—if the control weight is too small, we observe artifacts in solution space. Tuning the weight produces sparse solutions with no artifacts and the desired bang-bang control profile. Finally, we observe a different result when increasing the weight on the satellite example—this leads to longer thruster bursts on the thrusters with smaller limit, which in fact leads to less sparsity—23649 zero controls compared to 24509 when tuned. This can be explained by the reduction

of peaks at 200 N, which are penalized more, which in turn leads to the solver compensating by switching on the 50 N thrusters.

6.6.2 *Effects on convergence*

Finally, we examine the effects on convergence for L_1 costs. A plot of the cost evolution is shown in Figure 37 and a plot of the time to convergence for all considered cost terms is shown in Figure 38. We computed this over a grid of weights $\psi \in [10^{-5}, 10^{-1}]$ for $\beta = 1$. Generally, time to convergence is increased for all sparse costs with Huber being slowest and Pseudo-Huber fastest to converge on average. This is expected as sparsity-inducing cost terms have less steep gradients further away from zero, where the gradient for an L2 loss would be larger.

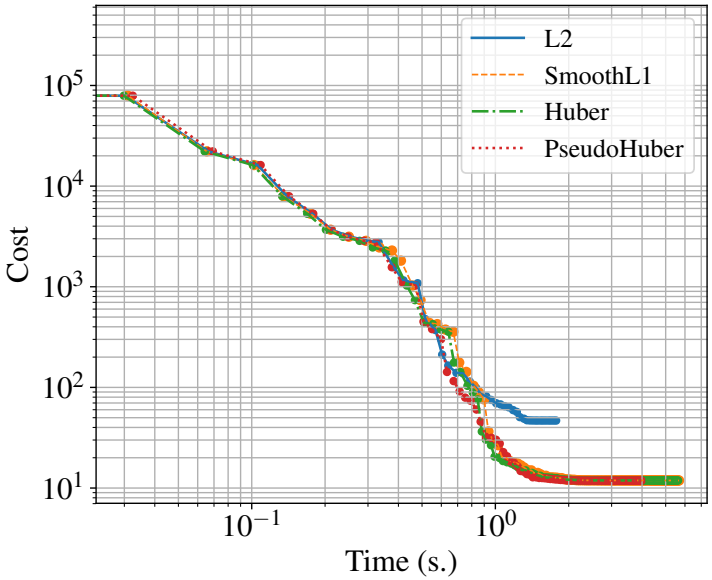


Figure 37: Cost plot for the Satellite example: Average per-iteration time is 0.04 seconds for all costs, but it takes more iterations for sparse costs to converge. Each dot represents an iteration.

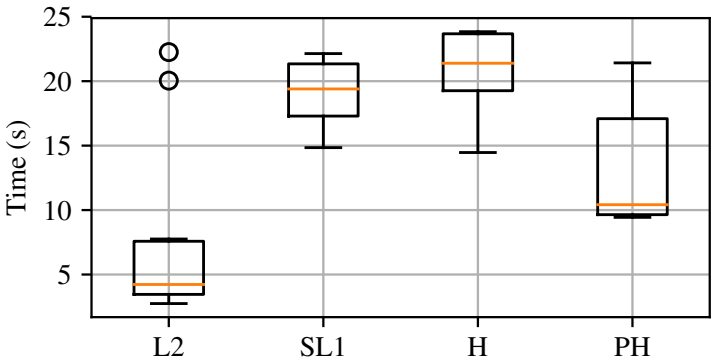


Figure 38: Timing analysis on the satellite example: Using sparse costs leads to slower overall convergence.

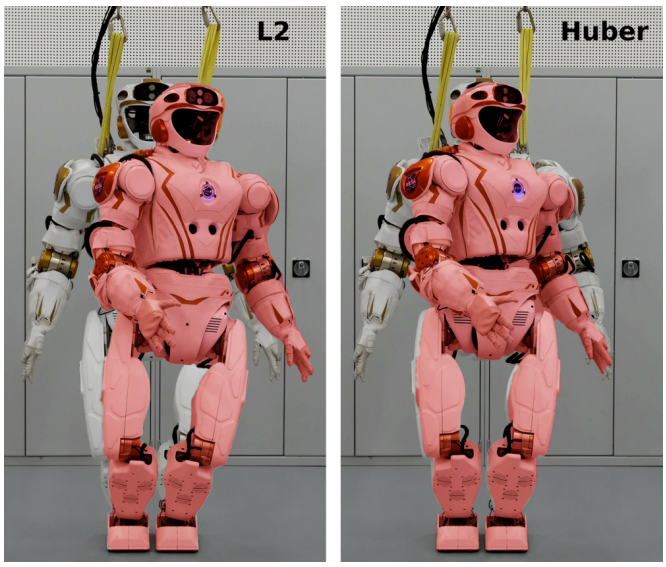


Figure 39: Reaching to a position target: Using L_2, the robot reaches the target while moving a majority of the joints by small amounts. A Huber loss induces sparsity improving tracking and maintaining an equal contact force distribution.

6.7 Active Joint Selection for Lower-Dimensional Tasks on Redundant Systems

Selecting the required degrees of freedom (DoF) of a redundant system such as a humanoid robot is another application of *sparsity-inducing* control. Instead of deactivating the control inputs, the planner deactivates unnecessary joints. Consider a reaching task for the 38-DoF Valkyrie humanoid robot, where the goal is to extend the hand (end-effector) forward to point at a target. Solving this via motion planning involves finding suitable control inputs for the entire humanoid such that it balances itself and extends the hand. With traditional control-penalty methods, the planner readily discovers a motion where all the joints are simultaneously moving (cf. Figure 39). Such a motion is arguably unnecessary and in fact undesirable as it requires more complex control to coordinate the motion. This can result in trajectories that are more difficult to execute and track by the low-level controller and may require more energy.

We applied the sparse costs to a reaching task on a 38-DoF humanoid robot. We use acceleration-based linear system dynamics and nonlinear general task costs. In this case, the three-dimensional reaching task requires only two joints to move (the shoulder and the hip). This is illustrated in Figure 39. The goal is to reach to $\boldsymbol{x}^* = [0.5, 0.2, 0.9]$, a point directly in front of the robot at hip-height. We discretized into $T = 20$ knots with $\Delta = 0.1$ s for a 2 s trajectory.

The resulting state and control plots for L_2 are in Figure 40. Solving the problem with L_2 control regularization produces a solution that moves more joints than necessary. This is easily seen in the corresponding state plots (Figure 40 and 41), showing the positions and velocities of the joints that move.

Applying a sparsity cost (Pseudo-Huber) to the problem leads to the solver choosing to move only the required joints. The resulting trajectories in Figure 41 show this clearly. However, due to the approximation of the L_1 costs, numerically the robot is

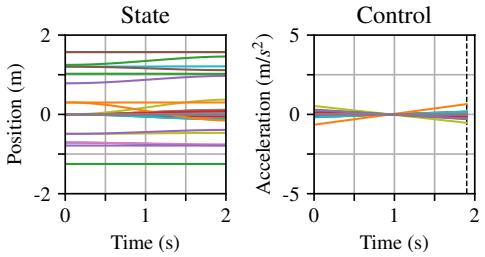


Figure 40: Valkyrie reaching task. L_2 uses 25 joints.

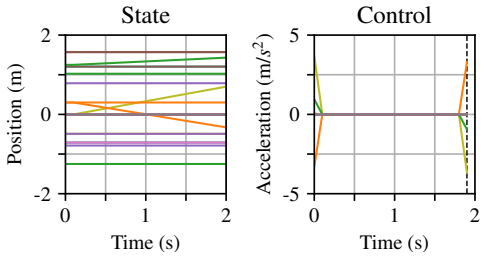


Figure 41: Valkyrie reaching task. Pseudo Huber uses 7 joints.

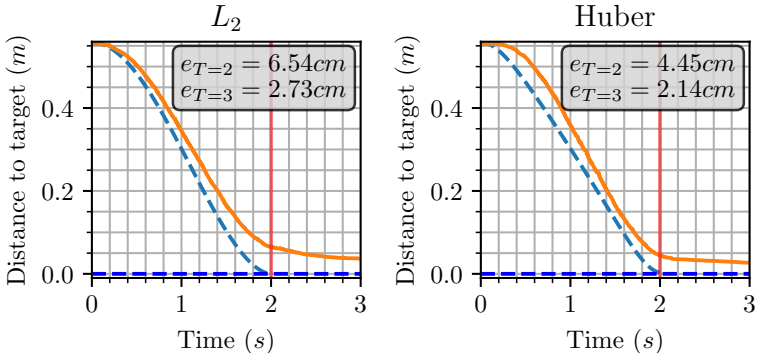


Figure 42: Tracking results for the Valkyrie reaching task. The red line highlights the end of the commanded trajectory. We highlight the task-space error at the end of the trajectory ($T = 2$s) and the end of the experiment. This illustrates that the whole-body controller adds a delay to the motion execution, and that the tracking error is lower for the sparsity-induced trajectory.

not moving 3 joints, as is apparent, but rather 7 have velocities greater than 10^{-3}. Compared with L_2, which moves 26 joints, this is nonetheless a significant reduction.

Finally, we executed the motion plans on the physical robot. The trajectories are tracked using an inverse dynamics based whole-body controller. We compared a solution with an L_2 cost and a Huber cost. We plot the tracking results (distance of end-effector to target) in Figure 42. For the Huber sparse trajectory, tracking is better both during and at the end of the trajectory.

6.7.1 *Supplementary Video*

A supplementary video outlining the method and showing videos of the motions is available at `https://youtu.be/YMXRZjFsqhc`.

6.8 DISCUSSION AND CONCLUSIONS

In this chapter we studied the effects of using an L_1 cost for the control of dynamic systems using optimal control. Since L_1 is not continuously differentiable, we studied three approximations: the SmoothL1, Huber, and PseudoHuber losses.

On a simple cartpole problem, L_1 costs lead to sparsity in control space by making a subset of the controls 0 and producing peaks that resemble square waves. We analyzed the performance of L_1 costs over a grid of values for the shape parameter β, which

thresholds switching between L_1 and L_2, and the control cost weight ψ. For smaller values of β a much larger control weight is required to achieve sparsity. Larger control weights, however, lead to higher task costs. We thus propose picking the largest value of β according to the system's noise tolerance and then fitting the weight *psi* until the desired level of sparsity is achieved. We further motivate this approach by showing that relying on sparsity and final task cost alone can lead to non-smooth control trajectories with visible artifacts.

Scaling L_1 costs to real-world robots presents new challenges. We successfully applied L_1 to a kino-dynamics optimal control problem on the Valkyrie robot to select a subset of active joints for a low-dimensional reaching task. While L_2 losses use more joints than necessary, L_1 can automatically reduce the number of joints by setting the corresponding controls to 0. Sparse controls can be in practice tracked better. However, sparse controls also result in bang-bang control with higher commanded accelerations that can damage the hardware.

On the other hand, this is a desired control profile for thruster control for satellites. We were able to achieve thruster-like behavior with L_1 costs in order to track a multi-stage center of mass trajectory. We further analyzed the timing performance of L_1 costs, showing that there is an increase in convergence time when using L_1 costs with Huber being the slowest, followed by SmoothL1 and Pseudo-Huber. Finally, we showed that the weight parameter ψ has a similar effect for satellite control as it does on the cartpole—low values of ψ lead to artifacts and high values of ψ lead to high task costs. In the satellite problem, however, high ψ did not numerically lead to more sparsity, as it produced longer thruster peaks for the lower control-limit thrusters, instead penalizing the high-limit thrusters more.

Finally, we note that it is important to enforce control limits when using sparsity-inducing losses. For unconstrained methods (DDP [61], FDDP [58]) clamping of applied controls in the forward-pass works in practice but results in slower convergence and often leads to getting stuck in local minima. We tested the losses with active-set control-limited DDP [86], and in particular BoxFDDP [60] in our experiments due to its greater generalization without an initial guess.

In the next chapter we focus on planning and control of a wheel-legged robot. We discuss modeling the physics of the robot and a more complex controller than we have so far discussed based on model predictive control (MPC). We verify in simulation that our models produce the desired motion and test our controller in a noisy and rough environments.

7 MODELING AND CONTROL OF A HYBRID WHEELED JUMPING ROBOT

Often the motion planning part of the co-design process is too complex for our current tools to differentiate. This was the case in Chapter 4, where taking the derivatives of dynamics for the quadruped was not possible with the tools at our disposal. Automatic differentiation was simply too slow for practical applications and finite differences was too inaccurate. In this chapter we take the approach of simplifying dynamics models using template models. These models capture the essential dynamic interactions of the robot while not taking into account the full dynamics. The benefit of using template models is that derivatives are often fast to compute analytically. They are only usable, however, if they can solve the motion planning problems we are interested in. As such from the perspective of co-design they can enable our tools to work in complex environments with the tradeoff of not capturing the full physics of the robot, thus limiting what attributes of the design we can optimize to only the ones included in the template model.

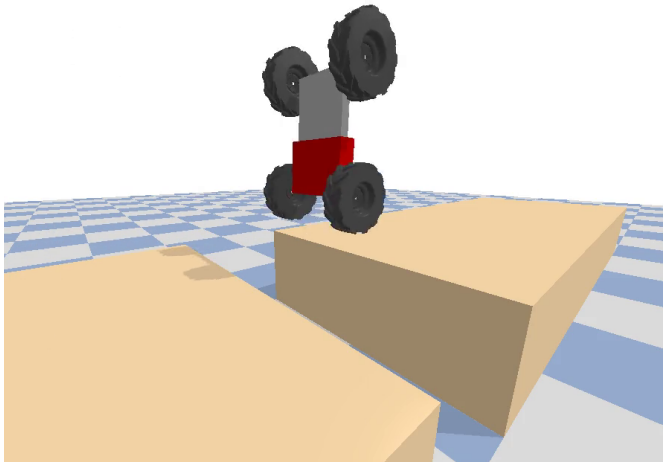


Figure 43: The wheeled jumping robot jumps across a gap. Please find the accompanying video at https://youtu.be/j2sIWL8m2pQ

We study a difficult control problem for a complex system – a wheel-legged hybrid robot Figure 43. The robot we consider has four wheels and a base. The wheels are actuated and the base has a prismatic joint in the middle, which is also actuated. We are specifically interested in controlling this robot when balancing on two of its wheels. This is a difficult robot to control as it needs to always actively balance, even when driving across (on two wheels). Instead of using the complex rigid-body dynamics of the system, we simplify using a template centroidal model that nonetheless captures the important interaction between wheels and base. We provide an analytical derivation of the system dynamics using this model, which we then use inside a model predictive controller (MPC). We study the behavior of the model and

demonstrate highly dynamic motions such as swing-up and jumping. We evaluate the controller on a variety of tasks and uneven terrains in a simulator.

7.1 INTRODUCTION

For model-based control, a model of the system is needed. It should be able to handle the rolling contact of the wheels as well as the discrete contact changes typical for stepping and jumping motions of legged robots. However, instead of adding complexity to a legged robot model by adding the rolling contact, we choose to focus on developing a simplified model with only one prismatic joint to 'mimic' the capability of a leg. This provides us with a better tool for modeling the robot behavior than some of the more complex models.

Most wheel-legged robots ([7, 8, 29, 31, 50]) are high degree of freedom nonlinear systems which do not lend themselves readily to online whole-body trajectory optimization (TO). Previous work [31, 33, 66] explored kinematic motion planning where the robot is controlled like a mobile vehicle and the legs act as a suspension. By definition, these techniques do not consider the dynamics of the system.

The authors of [50] use a linear quadratic regulator (LQR) for balancing the Ascento robot. They linearize the nonlinear system dynamics around the fixed point at the upright configuration. This provides an efficient approximation but it limits the range of motion around the fixed point [87, Chapter 3]. By contrast, ANYmal [8] uses a centroidal dynamics model [71] in a two-stage controller: One for the center of mass (CoM) and one for the wheels. This approach, therefore, fails to exploit potential combined synergies of the wheel and body dynamics.

Skaterbots [29] solve the full optimization. However, due to the complexity of the resulting nonlinear program, the constraints are expressed as cost terms and optimized using Newton's method. Compared to Skaterbots, we are able to solve the nonlinear problem due to the simple template model. Additionally, here, we use a Model Predictive Controller scheme, compared to the PD control used in Skaterbots.

Finally, the authors in [89] propose a Quadratic Programming (QP)-based approach to wheel-legged locomotion. In order to make the problem tractable, the z-component of the trajectory, as well as the CoM trajectory are taken as inputs to the QP-solver. In contrast, we directly optimize the z-component of the robot, thus allowing the optimizer to discover jump-like motions.

The Handle and the Flea by Boston Dynamics show remarkably dynamic motions. Flea is capable of jumping to a height of 10 m using combustive propulsion. Handle shows jumping motion while driving forward, as well as traversing rugged terrain. However, methods used to control these systems have not been published.

Instead of considering the full rigid body dynamics, one can model the system using simpler template models that capture the essence of the robot dynamics.

A popular template model for legged robots is the Spring-Loaded Inverted Pendulum (SLIP) [75]. SLIP has been used for high speed running and jumping [94]. For wheeled balancing systems, Wheeled Inverted Pendulum models (WIP) have been extensively used as template models [17]. Template models based on SLIP and WIP can be applied to wheel-legged systems, however they will consider either only the leg or only the wheel, respectively.

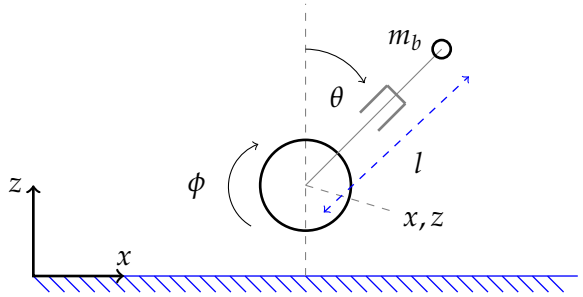


Figure 44: The proposed Variable-Length Wheeled Inverted Pendulum (VL-WIP) model.

In a similar fashion in this chapter we propose the simplest template model of wheel-legged robots. Our system consists of two sets of wheels connected by a prismatic joint (see Figure 43).

7.2 DYNAMICS MODEL

We begin by modeling the system behavior using a template model we call a Variable-Length Wheeled Inverted Pendulum (VL-WIP). Our model is based on the WIP model which is used to control balancing robots [17]. WIP consists of a wheel and a pole. The pole is modeled as a point mass a fixed distance away from the wheel (see the derivation in [23]). We extend this model to include a prismatic joint and a floating base described by a z-coordinate as shown in Figure 44.

The dynamics model $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ describes the system behavior in terms of its state $\boldsymbol{x} = [\boldsymbol{q}^T \ \boldsymbol{v}^T]^T$ and controls $\boldsymbol{u}$, where $\boldsymbol{q}$ is the generalized position and $\dot{\boldsymbol{q}}$ is the generalized velocity. The generalized position is $\boldsymbol{q} = [x \ z \ \phi \ l \ \theta]^T$, where x, z are the coordinates of the center of the wheel, ϕ is the wheel's angle along its axis, l is the distance from the center of the wheel to the body point mass and θ is the rotation of the body from the inertia z-axis. Controls are $\boldsymbol{u} = [\tau \ f]^T$, where τ is actuation torque of the wheel and f is the linear force in the prismatic joint. The explicit Equations of Motion (EoM) for the system under contact can be written as:

$$\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{v}} + \boldsymbol{C}(\boldsymbol{q}, \boldsymbol{v})\boldsymbol{v} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{S}^T\boldsymbol{u} + \boldsymbol{J}_C^T\boldsymbol{\gamma} \tag{79}$$

where $\boldsymbol{M}$ is the mass matrix, $\boldsymbol{C}$ is the Coriolis matrix, $\boldsymbol{G}$ is the gravity vector, $\boldsymbol{S}$ is a selection matrix, $\boldsymbol{J}_C$ is the contact Jacobian and $\boldsymbol{\gamma}$ stands for the contact forces. Note that, this dynamics equation also applies when the robot is in flight and the contact forces vanish. We use the Lagrangian method to derive the dynamics of the system [65, Chapter 6]. First, we define the position x_b, z_b of the mass m_b and its velocity $\dot{x}_b, \dot{z}_b$:

$$\begin{aligned} x_b &= x + l \ \sin(\theta) \\ z_b &= z + l \ \cos(\theta) \\ \dot{x}_b &= \dot{x} + \dot{l} \ \sin(\theta) + l \ cos(\theta) \ \dot{\theta} \\ \dot{z}_b &= \dot{z} + \dot{l} \ \cos(\theta) - l \ sin(\theta) \ \dot{\theta} \end{aligned} \tag{80}$$

Lagrange's method states that for a system with total kinetic energy T and potential energy U:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_i} - \frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_i} = \boldsymbol{f}_{ext}, \tag{81}$$

where $\mathcal{L} = T - U$ is the system's Lagrangian and $\boldsymbol{f}_{ext}$ are external forces applied to the system. We now need to compute the system's kinetic and potential energy.

In general, every link will have a rotational and translational kinetic energy component. For the wheel we include a rotational kinetic energy term $I_w\dot{\phi}^2$ where $I_w = m_w R_w^2$ is the moment of inertia of the wheel. Since the point mass has a zero moment of inertia, it only has a translational kinetic energy $m_b \boldsymbol{v}_b^T \boldsymbol{v}_b$, where $\boldsymbol{v}_b$ is the velocity of the point mass.

The only potential energy component is due to gravity g acting on the wheel and the point mass. This leads to:

$$T = \frac{1}{2}(I_w\ \dot{\phi}^2 + m_w\dot{x}^2 + m_w\dot{z}^2 + m_b\dot{x}_b^2 + m_b\dot{z}_b^2) \tag{82}$$

$$U = m_w g z + m_b g z_b \tag{83}$$

$$\mathcal{L} = T - U \tag{84}$$

Inputting Equations (82), (83), and (84) into Equation (81) leads to a system of five equations, the solution of which is the equations of motion (85)-(87):

$$\mathbf{M}(\boldsymbol{q}) = \begin{bmatrix} m_t & 0 & 0 & m_b s_\theta & m_b l c_\theta \\ 0 & m_t & 0 & m_b c_\theta & -m_b l s_\theta \\ 0 & 0 & I_w & 0 & 0 \\ m_b s_\theta & m_b c_\theta & 0 & m_b & 0 \\ m_b l c_\theta & -m_b l s_\theta & 0 & 0 & m_b l^2 \end{bmatrix} \tag{85}$$

$$\mathbf{C}(\boldsymbol{q}, \boldsymbol{v}) = \begin{bmatrix} 0 & 0 & 0 & 2m_b c_\theta \dot{\theta} & -m_b l s_\theta \dot{\theta} \\ 0 & 0 & 0 & -2m_b s_\theta \dot{\theta} & -m_b l c_\theta \dot{\theta} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -m_b l \dot{\theta} \\ 0 & 0 & 0 & 2m_b l \dot{\theta} & 0 \end{bmatrix} \tag{86}$$

$$\mathbf{G}(\boldsymbol{q}) = \begin{bmatrix} 0 & g m_t & 0 & g m_b c_\theta & -g m_b l s_\theta \end{bmatrix}^T \tag{87}$$

where $m_t = m_b + m_w$ is the total mass which sums up the body mass m_b and the wheel mass m_w, I_w is the inertia of the wheel, $s_\theta = \sin(\theta)$ and $c_\theta = \cos(\theta)$.

The matrix $\mathbf{S}$ translates controls τ and f into the generalized coordinates of the system. f directly maps to the prismatic joint. The torque τ applied on the wheel will lead to a counter reaction torque $-\tau$ on the body of the robot. We write this down as:

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{88}$$

Finally, the contact Jacobian relates the velocity of the contact point $\dot{\boldsymbol{r}}_C = [\dot{x}_c \; \dot{z}_c]^T$ to the generalized velocities of the robot:

$$\dot{\boldsymbol{r}}_C = \boldsymbol{J}_C \boldsymbol{v} = \begin{bmatrix} 1 & 0 & -R_w & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{v} \tag{89}$$

where R_w is the radius of the wheel.

Because of our system's simplicity, we have derived an analytic dynamics model. Compared to the full-body dynamics of complex systems, e.g. computed using recursive algorithms, analytic dynamics and their derivatives are faster to evaluate. Using the analytic dynamics allows us to plan motions for both the body and the wheels in a unified way. A unified planning approach allows the solver to plan for motions for which the interaction between wheels and base is key.

7.3 PROBLEM FORMULATION AND CONTROL

This section looks at how the derived system dynamics can be incorporated into a planning and control pipeline. For motion planning, we use an optimal control formulation, namely direct transcription [48]. To account for model mismatch and perturbations during execution, we use Model Predictive Control [77], which calls the planner iteratively to replan the motion from the current robot state.

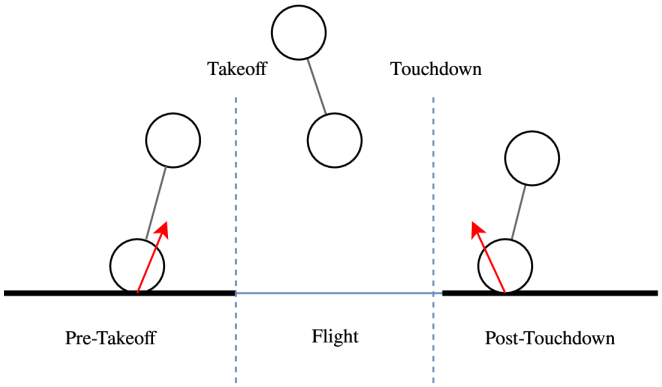
7.3.1 *Planning*

The generic planning problem formulation is illustrated in Figure 45. We formulate a nonlinear optimization problem (NLP) that finds trajectories $X = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, $U = \{\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots \boldsymbol{u}_{N-1}\}$, and $\Gamma = \{\gamma_0, \gamma_1, \ldots \gamma_{N-1}\}$ that satisfy the problem constraints and minimize the total energy, weighted by a normalizing term W. Here, $\boldsymbol{x}$, $\boldsymbol{u}$, and γ refer to the robot state, controls and contact forces respectively.

Firstly, we specify the duration T of the motion in seconds and discretize it into N knot points. At each knot point the optimizer finds states $\boldsymbol{x}$, controls $\boldsymbol{u}$, and contact forces γ that satisfy the system dynamics $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \gamma_t)$ derived in Section 7.2.

Next we setup task specific constraints. The start state $\boldsymbol{x}_0^*$ and the target state $\boldsymbol{x}_N^*$ are specified for knots $\boldsymbol{x}_0$ and $\boldsymbol{x}_N$. At each knot point, state limits $[\mathbf{x}^-, \mathbf{x}^+]$, and control limits $[\mathbf{u}^-, \mathbf{u}^+]$ are enforced. These limits ensure the physical feasibility of the motion generated. For different tasks the state and control limits will differ. For instance, when jumping over a gap, the state limits for the three phases will indicate where the "flight" phase begins and ends. In general, a jumping motion involves three phases: a "pre-takeoff" phase, where the robot is on the ground, a "flight" phase where the robot is in the air, and a "post-touchdown" phase where the robot lands and re-balances. We specify the duration of each phases by setting the time of takeoff T_{tf}, and the time of touchdown T_{td}.

Finally, we specify ground contact constraints. Contact forces must be zero ($\gamma_t = \mathbf{0}$) while the robot is in the flight phase. For the ground phases, they should stay inside the friction cone ($|\gamma_t^x| \leq \mu \gamma_t^z$, where μ is the friction coefficient) and have a positive vertical component ($\gamma_t^z \geq 0$). For kinematics, we enforce a no-slip constraint while the robot is on the ground: $\dot{x}_t = R_w \dot{\phi}_t$, where R_w is the wheel radius. For motions



$$\min_{X,U,\Gamma} \sum_i W \boldsymbol{u}_i^T \boldsymbol{u}_i \qquad \text{(Minimum Energy Cost)}$$

$$\begin{aligned}
\text{s.t. } & \mathbf{x}_0 = \mathbf{x}_0^* && \text{(Start State)}\\
& \mathbf{x}_N = \mathbf{x}_N^* && \text{(Goal State)}\\
& \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\gamma}_t),\ t \in [0, N] && \text{(Dynamics)}\\
& \mathbf{x}^- \leq \mathbf{x}_t \leq \mathbf{x}^+,\ t \in [0, N] && \text{(State Bounds)}\\
& \mathbf{u}^- \leq \mathbf{u}_t \leq \mathbf{u}^+,\ t \in [0, N-1] && \text{(Control Bounds)}\\
& \text{if } t \in [T_{\text{tf}}, T_{\text{td}}] \text{ (Flight Phase)} &&\\
& \quad \boldsymbol{\gamma}_{\text{t}} = \mathbf{0} && \text{(No Ground Force)}\\
& \text{if } t \notin [T_{\text{tf}}, T_{\text{td}}] \text{ (Ground Phases)} &&\\
& \quad |\gamma_t^x| \leq \mu \gamma_t^z, (\mu : \text{friction coefficient}) && \text{(Friction Cone)}\\
& \quad \gamma_{\text{t}}^z \geq 0 && \text{(Unilateral Force)}\\
& \quad \dot{x}_t = R_w \dot{\phi}_t && \text{(No Slip on Ground)}
\end{aligned}$$

Figure 45: Problem formulation. On the top we illustrate the three-phases involved in the planning problem. Below is the mathematical formulation of the problem.

without jumping, we use a similar formulation by removing the flight phase related constraints.

The planner finds an admissible trajectory for the control task. However, for executing it on a real system or in simulation, a controller is needed.

7.3.2 *Control*

With open-loop control, the planned trajectory cannot be tracked precisely due to tracking error introduced by model mismatch and external perturbations. Thus, we use Model Predictive Control [77] to handle the errors. Figure 46 illustrates the scheme. At each timestep, the planner re-plans the trajectory starting at the observed robot state $\boldsymbol{x}_{obs}$ obtained from the simulator. Then, the first action of the plan $\boldsymbol{u}_0$ is executed. After that, it recedes the time horizon T by the elapsed time.

A natural problem arises with the recession of T. Ideally we would keep the number of knot points N constant—this enables saving the optimizer's state and avoids costly

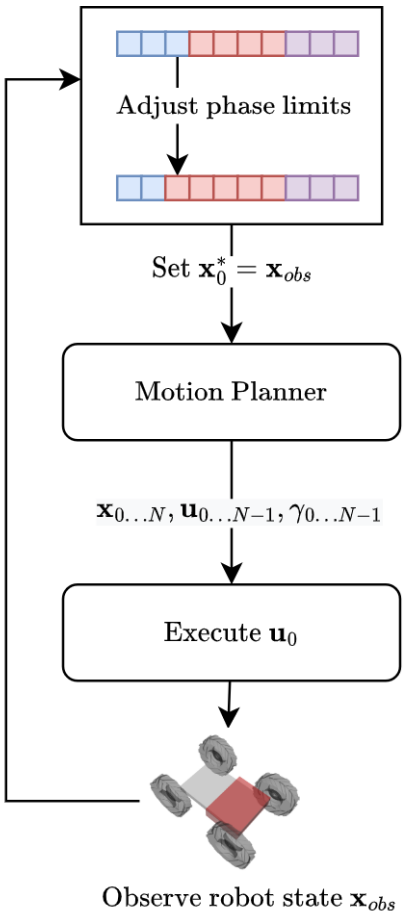


Figure 46: MPC control pipeline. The controller re-plans a motion every timestep by setting the start state $\boldsymbol{x}_0^*$ to the current state in simulation $\boldsymbol{x}_{obs}$. It then executes the first control $\boldsymbol{u}_0$ and updates the phases and limits of the planner.

re-initialization. The solution requires adjusting the limits of the problem so that each phase always has the same duration. Thus, we maintain the ratio of pre / flight / post phases by shifting the limits of the problem while keeping N the same.

To implement the optimal control problem, we used CasADI [4]. As the underlying solver, we used KNITRO and selected the Interior/Conjugate-Gradient algorithm that is well-suited for large-scale sparse optimal control problems [13].

7.4 SIMULATION RESULTS

In order to validate our approach, we investigated several tasks, namely swing-up, balancing and jumping in the physics simulator PyBullet [19]. Please find the accompanying video at https://youtu.be/j2sIWL8m2pQ.

The simulated robot weighs 6 kg including the wheels and the dimensions of the base are $0.3 \times 0.4 \times 0.1$m. The radius of the wheels is 0.17 m and wheels are 0.15 m wide.

7.4.1 *Swing-up and Balance*

Firstly, we demonstrate that the robot can switch from a driving mode on four wheels to a balancing mode on two wheels. This demonstrates that the robot can switch between modes without intervention. We used a two-stage controller for this task. Since controlling the robot in driving mode is outside the scope of this chapter, we drive forward with a constant acceleration until we reach a velocity of $\dot{x} = 3\text{m}\,\text{s}^{-1}$. Then we switch to the proposed MPC scheme.

The trajectory duration (and MPC horizon) was $T = 5\text{s}$ with $N = 20$ knot points. We set control limits to $\pm 10\,\text{N}\,\text{m}$ and $\pm 200\,\text{N}$ for τ and f, respectively. Figure 47 shows the resulting motion. The controller planned a motion where the robot coordinates its prismatic joint and wheel to swing-up.

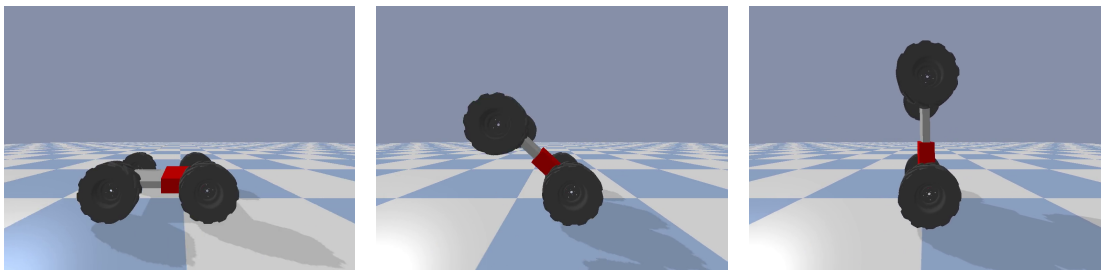


Figure 47: Swing-up motion.

7.4.2 *Driving Upright*

Next we demonstrate that, in the upright mode, the robot can balance and drive forward using the proposed approach. In this experiment, the goal is to drive the robot forward 1 m while balancing upright. We defined the same limits on the states $[\boldsymbol{x}^- \; \boldsymbol{x}^+]$, and the controls $[\boldsymbol{u}^- \; \boldsymbol{u}^+]$ for the entire trajectory. We set torque limits for the wheels to $\pm 10\,\text{N}\,\text{m}$ and force limits for the prismatic joint to $\pm 200\,\text{N}$. The total time (MPC horizon) was $T = 3\text{s}$ and $N = 50$ knot points.

The resulting motion is shown in Figure 48. We noticed that the optimizer first chooses to extend the prismatic joint before moving forward. This behavior can, for instance, be explained by noticing that it is easier to balance a body with a higher center of mass, which is achieved through extending the prismatic joint. Note that we did not in any way encode this behavior—the planner discovered it from first principles of optimization.

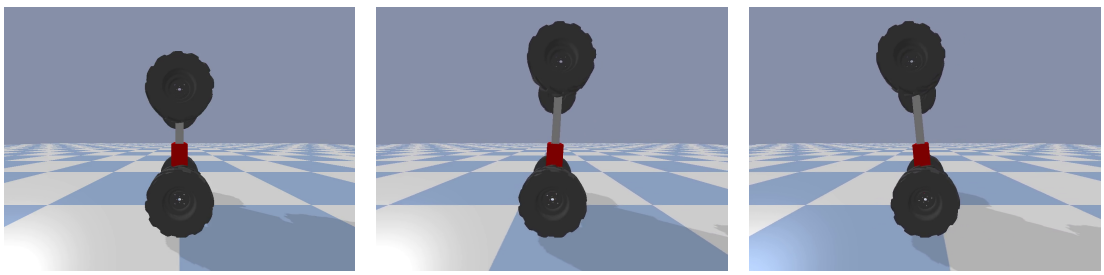


Figure 48: Moving forward while balancing.

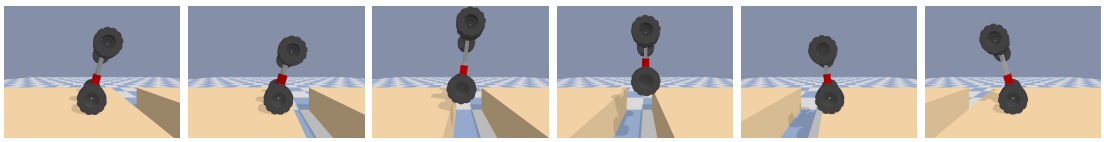


Figure 49: Jumping over a gap.

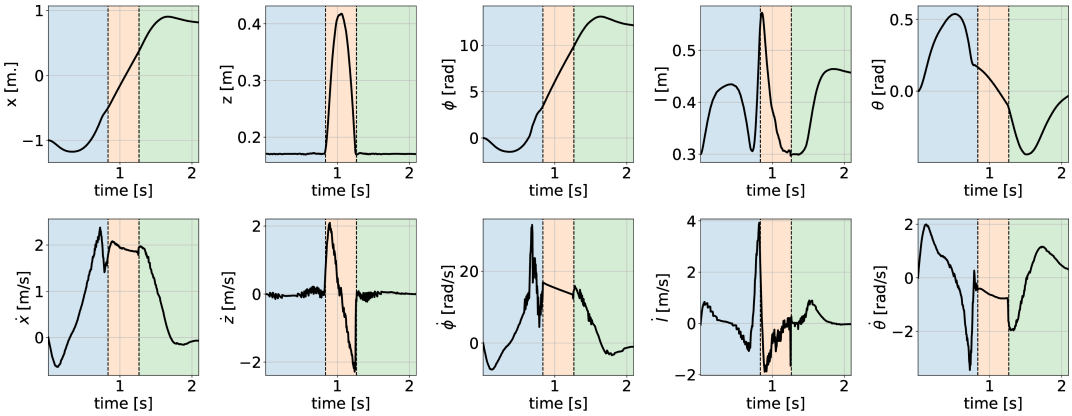


Figure 50: State evolution for the jumping motion in Figure 49. Phases are—"pre-takeoff" in blue, "flight" in orange, and "post-touchdown"—in green.

7.4.3 *Jumping Over a Gap*

Finally, we consider jumping over a gap, which uses the full multi-phase optimization described in Section 7.3. This task is more complex since different phases are involved in the motion.

We can entirely describe the jumping motion via state and control limits. In addition to torque limits $\pm 10\,\mathrm{N\,m}$ and force limits $\pm 200\,\mathrm{N}$, we set the state limits $\boldsymbol{q}_{lim}$ for each of the phases. Letting g^-, g^+ specify the bounds of the gap in the x direction, we obtain:

$$\boldsymbol{q}_{lim}^{\textit{pre-tf}} = \begin{bmatrix} -\infty & R_w & -\infty & l^- & -\pi/2 \\ g^- & R_w & \infty & l^+ & \pi/2 \end{bmatrix} \tag{90}$$

$$\boldsymbol{q}_{lim}^{\textit{flight}} = \begin{bmatrix} -\infty & R_w & -\infty & l^- & -\pi/2 \\ \infty & z^+ & \infty & l^+ & \pi/2 \end{bmatrix} \tag{91}$$

$$\boldsymbol{q}_{lim}^{\textit{post-td}} = \begin{bmatrix} g^+ & R_w & -\infty & l^- & -\pi/2 \\ \infty & R_w & \infty & l^+ & \pi/2 \end{bmatrix} \tag{92}$$

where l^- and l^+ are the limits of the prismatic joint, R_w is the wheel radius, and z^+ limits the jumping height. In order to set up the length of the trajectory , we ran time optimization within bounds $T^- = 0.3, T^+ = 5$ and knot points $N_{\textit{pre-tf}} = 40, N_{\textit{flight}} = 20, N_{\textit{post-td}} = 40$. This ensures that the optimizer chooses the appropriate flight duration that is physically feasible within the set time limits. In this sense, $T = 2.3\mathrm{s}$ was the optimal time and MPC horizon.

The resulting motion is shown in Figure 49 and its state and control history is plotted in Figure 50. The optimizer first "contracts" the body of the robot by retracting the prismatic joint before takeoff and extends it just before landing to absorb impact. We can see the preparatory movement in the optimized motion in Figure 49 for x, ϕ and θ.

7.5 ROBUSTNESS

In the following experiments, we evaluate the robustness of the MPC approach as compared to a Proportional-Derivative (PD) controller with feed-forward torque.

The PD controller has the following control equation:

$$\tau = \tau_{\text{des}} - K_p^{\theta} e(\theta) - K_d^{\theta} e(\dot{\theta}) - K_p^{x} e(x) - K_d^{x} e(\dot{x}) \tag{93}$$

$$f = f_{\text{des}} + K_p^{f} e(f) + K_d^{f} e(\dot{f}) \tag{94}$$

where $e(\cdot)$ is the error between desired and measured state variables, $K_p^{\theta} = 15$, $K_d^{\theta} = 2$, $K_p^{f} = 1,000$, $K_d^{f} = 100$, $K_p^{x} = 1.25$, $K_d^{x} = 2$ are the PD gains we tuned by hand. We add a term dependent on x in the first equation so that position error is taken into account. Note, the larger PD gains for the prismatic joint—this is due to the scale difference in the length and force needed. We tuned the PD controller so that it achieves similar performance when driving on flat terrain as the MPC controller.

7.5.1 *Robustness to Sensor Noise*

This experiment has the same setup as Section 7.4.2. We compare the effects of sensor noise on both the MPC and the PD controllers. We added Gaussian noise to sensor readings at each timestep of the simulation:

$$\mathbf{x}'_{\text{obs}} = \mathbf{x}_{\text{obs}} + \mathcal{N}(\mathbf{0}, \sigma\mathbb{I}) \tag{95}$$

where σ is the noise standard deviation, which was varied in the $0 - 0.4$ range and $\mathbb{I}$ is the identity matrix. We computed the L2-distance from the observed state $\mathbf{x}_{\text{obs}}$ to the goal state $\mathbf{x}^*$ at the end of the simulation. For this experiment we ran 20 trials for each value of σ, computing the means as well as the standard deviations of the L2-distance. The results are in Figure 51.

We notice similar performance for the PD controller at lower noise strengths. However, at $\sigma \geq 0.2$ the PD controller has a significant drop in performance. Additionally, as the noise strength increases, the variability in performance for the PD controller becomes larger. In contrast, the MPC controller shows better mean performance and lower performance variability across σ values.

7.5.2 *Rough Terrain Locomotion*

For this experiment we consider the task of driving forward on two wheels, as outlined in Section 7.4.2. For the MPC controller we set a target velocity $\dot{x}^* = 1\text{m}\,\text{s}^{-1}$ and horizon $T = 1\text{s}$. The PD controller has no feed-forward torque reference; it has the same desired velocity target as well as a balance target $\theta^* = 0$. Neither have knowledge of the terrain and the overall duration of each experiment was $T = 5\text{s}$.

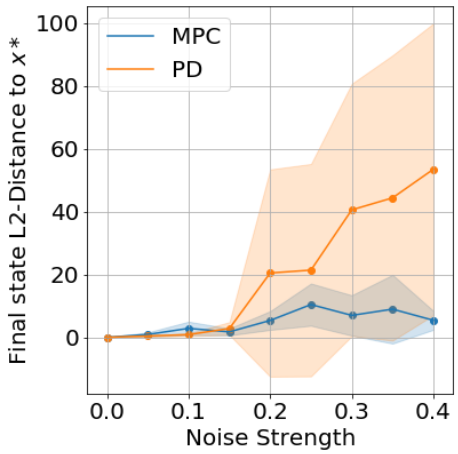
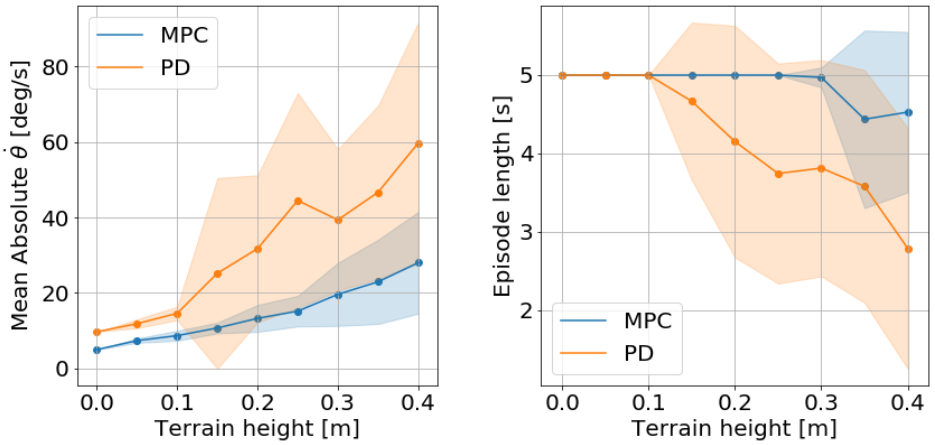


Figure 51: Driving with sensor noise. L2-distance to x^* and one standard deviation.

We ran 20 experiments for varied terrain heights in the range 0 m to 0.4 m. We generated random terrain using Perlin noise for every experiment while keeping the same random number generator seed between the PD and MPC. Example generated terrain for heights 0.1 m and 0.4 m are shown in Figure 53a and Figure 53b, respectively. We computed the mean absolute angle velocity $\dot{\theta}$ for the entire trajectory for each of the terrain heights and its standard deviation. We also show the episode duration as well as one standard deviation. The episode is terminated when the angle of the robot is more than 85 degrees. The results are in Figure 52a and Figure 52b, respectively.

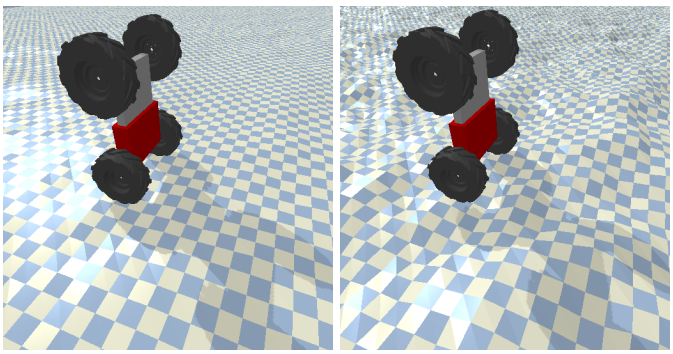


(a) Mean absolute angle velocity $\dot{\theta}$ and one standard deviation.

(b) Episode duration and one standard deviation.

Figure 52: Rough Terrain Locomotion Results

For flat terrain and for terrain height up to 0.1 m, MPC and PD show similar results. As the terrain height increases, however, MPC is better at stabilizing the robot with much less variability in performance.



(a) Rough terrain—height 0.1m.

(b) Rough terrain—height 0.4m.

Figure 53: Examples of rough terrain used.

7.6 DISCUSSION AND FUTURE WORK

The results demonstrated in this chapter show that the VL-WIP model, regardless of its simplicity, is sufficient for generating complex dynamic motions. We consider the combined motion planning pipeline developed here as a first step towards more dynamic locomotion of hybrid wheel-legged systems. Future work will apply several connected VL-WIP models, one for each limb, to wheel-legged bipeds and quadrupeds, using the standard motion planning and control pipeline outlined here. This approach is similar to using a SLIP model for each limb of a biped [56, 92, 93]. Compared to single rigid-body models, like the one in [95], the stacked VL-WIP model will model leg mass and inertia properties, allowing for more complex in-air maneuvers as well as not requiring inertia compensation.

Furthermore, for all the motions the planner exploited the dynamics of the system without being guided to do so explicitly. Most notably for jumping it discovered the preparatory motion from optimization alone—including contracting and extending the robot before take off, absorbing the impact upon landing, and extending the prismatic joint for easier balancing.

Currently for swing up and jumping we have predefined the switching times. In the future these can be automatically discovered, for instance via phase-based parametrization [85, 95].

When transferring to a real system, we note that work needs to be done to increase the speed of the computation, for instance by warm-starting the solver using trajectory libraries [63] or using policy learning. Our MPC controller ran at 109 ± 64 Hz for balancing, 18 ± 37 Hz for jumping, and 36 ± 13 Hz for swing-up on a Intel Core i9-9980HK at 2.4 GHz with 16 GB of DDR4 RAM at 2667 MHz. We have indeed noticed that the solver did not take an equal number of iterations for all initial conditions and it sometimes did not converge. For this reason the speed was significantly lower on the more complex three-phase optimization for the jumping task. The convergence of the motion planner and the impact of warm starts is outside of the scope of this chapter but it presents an exciting direction for future research.

Finally, this chapter concludes *Part 2* of this thesis. In the next chapter we discuss limitations, future work and conclude the thesis.

Part 3
Conclusion and Future Work

8 CONCLUSION AND FUTURE WORK

In this thesis we presented our research in co-design and motion planning. In order to conclude this thesis, we first summarize the research questions and direction we outlined in Chapter 1. Our hypothesis is that we can include the full complexity of design constraints within a bi-level optimization approach for co-design and that this method can solve the co-design problem successfully (converge to a feasible design with a lower cost). Furthermore, we specifically aimed to use the full constrained motion planning problem in the lower-level and compute its derivative. We further set out to obtain a computational speed and scalability benefit with respect to (w.r.t.) existing approaches by leveraging the fast computation speed of specialized motion planners. We additionally hypothesized that the co-design process is more stable for bi-level methods than simultaneous methods in the sense that the motion is always optimal thus simplifying the entire optimization.

Below are our findings for each of the questions:

- Firstly, we were indeed able to solve co-design problems for (i) a quadruped trotting and jumping with fixed contacts (Chapter 3, Chapter 5), (ii) a single- and double-pendulum (Chapter 4) and for a Kinova manipulator and a reaching task (Chapter 4).
- Secondly, we did indeed obtain a computational advantage compared to (i) covariance matrix adaptation evolutionary strategy (CMA-ES) – between 1.8 and 8.4 times faster on a quadruped jumping/trotting task respectively (Chapter 3) and (ii) a single NLP – 54 times faster on a Kinova-arm reaching task (Chapter 4).
- Although the simultaneous approaches in our experiments did not diverge, bi-level optimization achieved a similar co-design cost (Chapter 4) and always had a feasible motion (due to differential dynamic programming (DDP)). This was indeed not the case for simultaneous approaches where the problem was more complex in the sense that the design vector influenced all constraints at every timestep. As such, while the bi-level approach was not more "stable", as the single NLP always converged, we did observe a significant 54-times speed improvement.

All of these findings enable the development of interactive co-design tools. Indeed we developed such a tool for quadrupeds (Chapter 3) that shows our vision for how co-design can be used to aid prototyping and quickly evaluating designs on multiple environments.

Moreover, we showed how to analytically differentiate optimal control problems via DDP(Chapter 4), which is necessary to improve scalability for more complex dynamic systems. Though this is still hindered by the lack of availability of dynamics derivatives w.r.t. design variables, we still obtained a speed improvement (as outlined above).

We also verified our approach on real hardware and observed an improvement in co-design cost (Chapter 5).

Moreover, we proposed solutions to two outstanding problems in motion planning. We formulated an extension to DDP where sparsity-inducing L_1 costs are used for controlling for discrete on-off controls and an application to satellite thruster control as well as an application for automatic joint selection of a humanoid robot for low-dimensional tasks. Thrusters are discrete in nature, which makes them difficult to solve for, especially in the context of full rigid-body dynamics. We proposed using differentiable L_1 approximations to tackle this problem. Lastly, we discussed the modeling and control of a wheel-legged robot via a model-predictive-control (MPC) approach. Wheel-legged robots are inherently unstable dynamic systems, which need actively balance in addition to moving forward or jumping. By using a simple template model, we were able to capture only the necessary dynamic interactions (between base and wheel) and integrate this into an model predictive control (MPC) framework.

In summary, we have taken the first step towards an interactive co-design tool that can handle arbitrary design constraints (in the upper level) and uses a constrained motion planner in the lower level. However, we have not fully solved this problem. Among the current limitations are (i) the fixed contact locations and timing (ii) the lack of fully analytical derivatives up to and *including* the system dynamics and (ii) the lack of support for multiple tasks evaluated at once. In the next section we discuss these and other limitations as well as directions for future work in line with our vision.

8.1 LIMITATIONS AND FUTURE WORK

Finally, in this sections we discuss some of the limitations of our work. Some of these limitations also present exciting directions for future work, which we outline.

8.1.1 *Differentiability of Motion Planners*

One of the main issues with differentiating nonlinear programs, such as Motion Planners, is the lack of differentiability guarantees. If the motion planning is convex and converges to the same "mode" of motion planning, in practice differentiating the motion planner we believe is likely to be feasible. This in part is what we observe in Chapter 3, where we have heavy regularization terms on the motion planning of the quadruped. This regularization serves two purposes in practice – (i) it ensures that when running motion planning in a receding horizon fashion on the real robot, the motion planner will converge to the same "mode" for variations in state space and (ii) for variations in design it does the same. Mode here refers informally to the quality of the motion – if for instance we encode a trotting motion and for some designs or some starting state the motion plan is something else than a trot, then this would be a different mode.

Thus multi-modal solutions to the motion planning problem would be challenging for gradient-based design optimization, as the optimizer can get stuck in local minima for one of the modes. If, on the other hand, there are simply too many modes and the robot exhibits vastly different behaviors for different designs, this could make the cost landscape non-differentiable.

One way of addressing this empirically is to simply run the optimization for several iterations and assess the optimization. As we showed in the discussion section of

Chapter 3 our strategy achieves feasible designs (design constraints satisfied) after only a few iterations and we see cost improvements after only minutes of optimization. Although this is not rigorous in nature, in practice we can quickly iterate between motion planning formulations if undesired behavior is encountered.

As a concrete example, we in fact encountered this issue while developing our method in Chapter 3. For the jumping task once the robot design was optimized for a few iterations, the motion planner would converge to a behavior where the base would be moved forward and then the legs would move to the target position without actually jumping. We then added a cost to penalize such non-jumping behavior in the motion planning level.

8.1.2 *Analytical Second-Order Gradients of Dynamics*

In Chapter 4 we presented an approach that differentiates DDP. However, our approach is analytical only up to the motion planner, or solver, itself. In fact the second-order dynamics derivatives of the system and derivatives w.r.t. design variables, that are required, were all computed using automatic differentiation (AD). While this is not an issue for small systems, such as the pendulum or double pendulum, where AD is very efficient, for the Kinova arm this is the limiting factor in terms of speed of computation. Rigid-body dynamics libraries do not support for derivatives w.r.t. design variables, however that is mostly a software limitation, as we show we can compute them with automatic differentiation. If we can do so, it therefore means we can also analytically compute the same expressions. This requires creating a standard way of parameterizing the robot models, where symbolic variables need to be used when defining the robot. Normally this is done using unified robotics description format (URDF). A symbolic version of the URDF will need to be introduced that allows for parametric link lengths, attachment points (joint positions) as well as parametric inertias and masses. This potentially can be done in a similar way to Xacro (XML Macros), which is commonly used to generate URDF from parametric Xacro files.

Recently in the robotics community there have been results showing the benefits of using second-order dynamics in DDP to solve optimal control (OC) problems as well. This involves the second-order derivatives we introduced in Chapter 4 and an efficient way to compute the relevant tensor product [81, 82]. A concrete direction for future work is in the use of the efficient algorithms developed in [82] to speed up the computation of derivatives of dynamics as we discuss above. Future work lies in developing such algorithms in order to enable online deployment of differentiable solvers for learning and system identification on real robots.

8.1.3 *Too Expensive for Real-Time Learning or Reconfigurable Design*

One of the exciting applications of co-design is for reconfigurable robots, where some of the design parameters can be changed during the motion. An example of such a robot is DyRET [69], which can change the lengths of its upper and lower legs while the robot is balancing in place. The technology for reconfigurable robots like DyRET is still in its infancy – DyRET can only execute slow walking gaits and not dynamic

motions. However, it is an exciting platform for applying co-design algorithms to find the best design during the motion execution from terrain sensor measurements.

However, this is only possible if the speed of computation is improved. Real-time speed for reconfigurable robots is not strictly required, as the robot can stop and reconfigure itself, however interactive speeds are at least required, in the orders of seconds.

If the speed of computation is improved further to real-time, differentiable solvers can be used for online system identification or imitation learning, even from only limited observations (only states or only controls). Learning the dynamics model of the robot or learning the policy on-the-fly requires faster computation as small errors in modeling or deviations in policy will quickly accumulate for more dynamic motions, which can lead to failure.

8.1.4 *Footstep Optimization*

One of the potential applications of co-design is in optimizing the foothold locations of the robot in addition to choosing timings (e.g. Chapter 6). Including footstep positions (and potentially orientations) as design variables does allow for them to be optimized by differentiating through the solver. However, this is challenging, as footsteps are the main source of discontinuity in the motion planning and can potentially be a source of discontinuity for differentiable solvers too. Care must be taken when defining the contact model and contact constraints if differentiable optimal control (DOC) is used for foothold optimization so that the derivatives in the upper level are not noisy and the costs are differentiable.

8.1.5 *Differentiable Simulation*

Another future direction lies within the use of differentiable simulation. Two recent works in the field stand out – Xu *et. al.* [96] used a differentiable simulator to optimize morphologies of grippers and their motions and De Vincenti *et. al.* [20] optimized the leg design for quadrupeds using a differentiable simulator and a quadratic programming (QP)-based controller and planner. These approaches still use simple motion planning that does not consider the full rigid-body dynamics, however they consider the more complex contact encountered by the robot through the simulator, which is hard to model during planning. The algorithms developed in this thesis are differentiable and as such can be included in a pipeline with a differentiable simulator as the motion planning algorithm to enable more complex and dynamic motions.

BIBLIOGRAPHY

[1] Defense Advanced Research Projects Agency. *DARPA Robotics Challenge*. https://www.darpa.mil/program/darpa-robotics-challenge. 2016 (cit. on p. 2).

[2] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots and J. Zico Kolter. 'Differentiable MPC for End-to-end Planning and Control'. In: *Advances in Neural Information Processing Systems*. 2018 (cit. on pp. 17–19, 21, 39, 42, 44–46, 48).

[3] Brandon Amos and J. Zico Kolter. 'OptNet: Differentiable Optimization as a Layer in Neural Networks'. In: *International Conference on Machine Learning*. 2019 (cit. on pp. 16, 17, 19).

[4] Joel Andersson, Johan Åkesson and Moritz Diehl. 'CasADi: A Symbolic Package for Automatic Differentiation andOptimal Control'. en. In: *Recent Advances in Algorithmic Differentiation*. Berlin, Heidelberg: Springer, 2012, pp. 297–307 (cit. on p. 85).

[5] John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010 (cit. on p. 19).

[6] Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B Shah. 'Julia: A fresh approach to numerical computing'. In: *SIAM Review* (2017) (cit. on p. 48).

[7] Marko Bjelonic, C. Dario Bellicoso, Yvain de Viragh, Dhionis Sako, F. Dante Tresoldi, Fabian Jenelten and Marco Hutter. 'Keep Rollin' - Whole-Body Motion Control and Planning for Wheeled Quadrupedal Robots'. In: *IEEE Robotics and Automation Letters* 4.2 (Apr. 2019), pp. 2116–2123 (cit. on p. 80).

[8] Marko Bjelonic, Prajish K. Sankar, C. Dario Bellicoso, Heike Vallery and Marco Hutter. 'Rolling in the Deep – Hybrid Locomotion for Wheeled-Legged Robots using Online Trajectory Optimization'. In: *arXiv:1909.07193 [cs, eess]* (Jan. 2020) (cit. on p. 80).

[9] Gabriel Bravo-Palacios, A. Del Prete and Patrick M. Wensing. 'One Robot for Many Tasks: Versatile Co-Design Through Stochastic Programming'. In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020). Conference Name: IEEE Robotics and Automation Letters, pp. 1680–1687 (cit. on pp. 12, 20).

[10] R. Budhiraja, J. Carpentier, C. Mastalli and N. Mansard. 'Differential Dynamic Programming for Multi-Phase Rigid Contact Dynamics'. In: *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. 2018, pp. 1–9 (cit. on p. 29).

[11] Gabriele Buondonno, Justin Carpentier, Guilhem Saurel, Nicolas Mansard, Alessandro De Luca and Jean-Paul Laumond. 'Actuator design of compliant walkers

via optimal control'. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 705–711 (cit. on p. 12).

[12] Richard H Byrd, Jorge Nocedal and Richard A Waltz. 'Knitro: An integrated package for nonlinear optimization'. In: *Large-scale nonlinear optimization*. Springer, 2006, pp. 35–59 (cit. on pp. 32, 53).

[13] Richard H. Byrd, Jorge Nocedal and Richard A. Waltz. 'Knitro: An Integrated Package for Nonlinear Optimization'. en. In: *Large-Scale Nonlinear Optimization*. Vol. 83. Boston, MA: Springer US, 2006, pp. 35–59 (cit. on pp. 11, 85).

[14] Alexandre Campeau-Lecours, Hugo Lamontagne, Simon Latour, Philippe Fauteux, Véronique Maheu, François Boucher et al. 'Kinova modular robot arms for service robotics applications'. In: *Rapid Automation: Concepts, Methodologies, Tools, and Applications*. IGI global, 2019 (cit. on pp. 7, 51).

[15] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse et al. 'The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives'. In: *IEEE/SICE International Symposium on System Integration (SII)*. 2019 (cit. on pp. 30, 46, 52).

[16] Michael Chadwick, Hendrik Kolvenbach, Fabio Dubois, Hong Fai Lau and Marco Hutter. 'Vitruvio: An Open-Source Leg Design Optimization Toolbox for Walking Robots'. In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 6318–6325 (cit. on pp. 14, 22).

[17] Ronald Ping Man Chan, Karl A. Stol and C. Roger Halkyard. 'Review of modelling and control of two-wheeled robots'. en. In: *Annual Reviews in Control* 37.1 (Apr. 2013), pp. 89–103 (cit. on pp. 80, 81).

[18] European Commission. *Memory of Motion*. https://cordis.europa.eu/project/id/780684. 2022 (cit. on p. 2).

[19] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016–2021 (cit. on pp. 32, 85).

[20] Flavio De Vincenti, Dongho Kang and Stelian Coros. 'Control-Aware Design Optimization for Bio-Inspired Quadruped Robots'. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 1354–1361 (cit. on p. 96).

[21] Ruta Desai, Beichen Li, Ye Yuan and Stelian Coros. 'Interactive Co-Design of Form and Function for Legged Robots using the Adjoint Method'. In: *arXiv:1801.00385 [cs]* (Apr. 2018). arXiv: 1801.00385 (cit. on pp. 17, 22, 32).

[22] Krishnamanaswi M. Digumarti, Christian Gehring, Stelian Coros, J. Hwangbo and Roland Siegwart. 'Concurrent optimization of mechanical design and loco-

motion control of a legged robot'. In: *Mobile Service Robotics*. World Scientific, 2014 (cit. on pp. 4, 14).

[23] Ye Ding, Joshua Gafford and Mie Kunio. 'Modeling, Simulation and Fabrication of a Balancing Robot'. en. In: *Advanced System Dynamics and Control* 151 (2012) (cit. on p. 81).

[24] Gabriele Fadini, Thomas Flayols, Andrea del Prete, Nicolas Mansard and Philippe Souères. 'Computational design of energy-efficient legged robots: Optimizing for size and actuators'. In: *2021 International Conference on Robotics and Automation (ICRA)*. 2020 (cit. on pp. 14, 27, 33).

[25] Farbod Farshidian, Michael Neunert, Alexander W. Winkler, Gonzalo Rey and Jonas Buchli. 'An Efficient Optimal Planning and Control Framework For Quadrupedal Locomotion'. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)* (May 2017). arXiv: 1609.09861, pp. 93–100 (cit. on p. 4).

[26] R. Featherstone. *Rigid Body Dynamics Algorithms*. Berlin, Heidelberg: Springer-Verlag, 2007 (cit. on p. 29).

[27] Henrique Ferrolho, Vladimir Ivan, Wolfgang Merkt, Ioannis Havoutis and Sethu Vijayakumar. 'Inverse dynamics vs. forward dynamics in direct transcription formulations for trajectory optimization'. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 12752–12758 (cit. on pp. 4, 11).

[28] Cheng-Yang Fu, Mykhailo Shvets and Alexander C. Berg. 'RetinaMask: Learning to predict masks improves state-of-the-art single-shot detection for free'. In: *arXiv:1901.03353 [cs]* (Jan. 2019). arXiv: 1901.03353 version: 1 (cit. on pp. 68, 69).

[29] Moritz Geilinger, Roi Poranne, Ruta Desai, Bernhard Thomaszewski and Stelian Coros. 'Skaterbots: optimization-based design and motion synthesis for robotic creatures with legs and wheels'. en. In: *ACM Transactions on Graphics* 37.4 (July 2018), pp. 1–12 (cit. on pp. 16, 17, 21, 26, 32, 80).

[30] Moritz Geilinger, Sebastian Winberg and Stelian Coros. 'A Computational Framework for Designing Skilled Legged-Wheeled Robots'. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3674–3681 (cit. on pp. 17, 21, 22, 32).

[31] Markus Giftthaler, Farbod Farshidian, Timothy Sandy, Lukas Stadelmann and Jonas Buchli. 'Efficient kinematic planning for mobile manipulators with non-holonomic constraints using optimal control'. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 3411–3417 (cit. on p. 80).

[32] Philip E. Gill, Walter Murray and Michael A. Saunders. *SNOPT: An SQP Algorithm For Large-Scale Constrained Optimization*. 2002 (cit. on p. 12).

[33] P. Robuffo Giordano, M. Fuchs, A. Albu-Schaffer and G. Hirzinger. 'On the kinematic modeling and control of a mobile platform equipped with steering

wheels and movable legs'. In: *2009 IEEE International Conference on Robotics and Automation*. May 2009, pp. 4080–4087 (cit. on p. 80).

[34] Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz and Edison Guo. 'On differentiating parameterized argmin and argmax problems with application to bi-level optimization'. In: *arXiv:1607.05447* (2016) (cit. on pp. 16, 17, 39).

[35] Felix Grimminger et al. 'An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research'. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3650–3657 (cit. on pp. 27, 57).

[36] Sehoon Ha, Stelian Coros, Alexander Alspach, James M Bern, Joohyung Kim and Katsu Yamane. 'Computational design of robotic devices from high-level motion specifications'. In: *IEEE Transactions on Robotics* 34.5 (2018), pp. 1240–1251 (cit. on p. 20).

[37] Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim and Katsu Yamane. 'Task-based limb optimization for legged robots'. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 2062–2068 (cit. on pp. 4, 14).

[38] Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim and Katsu Yamane. 'Computational co-optimization of design parameters and motion trajectories for robotic systems'. In: *The International Journal of Robotics Research* 37 (2018) (cit. on pp. 17, 22, 32).

[39] Nikolaus Hansen. 'Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed'. In: *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*. 2009, pp. 2389–2396 (cit. on pp. 5, 14, 35).

[40] Nikolaus Hansen. 'CMA-ES: A Function Value Free Second Order Optimization Method'. In: *PGMO COPI*. 2014 (cit. on p. 14).

[41] Nikolaus Hansen and Stefan Kern. 'Evaluating the CMA evolution strategy on multimodal test functions'. In: *International conference on parallel problem solving from nature*. Springer. 2004, pp. 282–291 (cit. on pp. 5, 14, 35).

[42] Nikolaus Hansen, yoshihikoueno, ARF1, Gabriela Kadlecová, Kento Nozawa, Luca Rolshoven, Matthew Chan, Youhei Akimoto, brieglhostis and Dimo Brockhoff. *CMA-ES/pycma: r3.3.0*. Version r3.3.0. Jan. 2023 (cit. on p. 34).

[43] Edward N. Hartley, Marco Gallieri and Jan M. Maciejowski. 'Terminal spacecraft rendezvous and capture with LASSO model predictive control'. In: *International Journal of Control* 86.11 (Nov. 2013), pp. 2104–2113 (cit. on p. 68).

[44] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. en. Cambridge University Press, 2003 (cit. on p. 69).

[45] E. M. Hoffman, M. P. Polverini, A. Laurenzi and N. G. Tsagarakis. 'A Study on Sparse Hierarchical Inverse Kinematics Algorithms for Humanoid Robots'. In: *IEEE Robotics and Automation Letters* 5.1 (2020), pp. 235–242 (cit. on p. 68).

[46] Peter J. Huber. *Robust Statistics*. en. John Wiley & Sons, 2004 (cit. on pp. 68, 69).

[47] Vladimir Ivan, Yiming Yang, Wolfgang Merkt, Michael P. Camilleri and Sethu Vijayakumar. 'EXOTica: An Extensible Optimization Toolset for Prototyping and Benchmarking Motion Planning and Control'. In: *Robot Operating System (ROS): The Complete Reference (Volume 3)*. Cham: Springer International Publishing, 2019, pp. 211–240 (cit. on p. 70).

[48] M. Kelly. 'An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation'. In: *SIAM Review* 59.4 (Jan. 2017), pp. 849–904 (cit. on p. 83).

[49] Yeonju Kim, Zherong Pan and Kris Hauser. 'Mo-bbo: Multi-objective bilevel bayesian optimization for robot and behavior co-design'. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 9877–9883 (cit. on p. 21).

[50] Victor Klemm et al. 'Ascento: A Two-Wheeled Jumping Robot'. en. In: *2019 International Conference on Robotics and Automation (ICRA)*. Montreal, QC, Canada: IEEE, May 2019, pp. 7515–7521 (cit. on p. 80).

[51] Jonas Koenemann, Andrea Del Prete, Yuval Tassa, Emanuel Todorov, Olivier Stasse, Maren Bennewitz and Nicolas Mansard. 'Whole-body model-predictive control applied to the HRP-2 humanoid'. In: *IEEE International Conference on Intelligent Robots and Systems (IROS)*. 2015 (cit. on pp. 7, 39).

[52] Twan Koolen and Robin Deits. 'Julia for robotics: Simulation and real-time control in a high-level programming language'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019 (cit. on pp. 52, 55).

[53] Simon Le Cleac'h and Zachary Manchester. 'Fast Solution of Optimal Control Problems With L1 Cost'. In: *AAS/AIAA Astrodynamics Specialist Conference* 904 (2019), pp. 1–11 (cit. on p. 68).

[54] Q. Li, W.J. Zhang and L. Chen. 'Design for control-a concurrent engineering approach for mechatronic systems design'. In: *IEEE/ASME transactions on mechatronics* 6.2 (2001), pp. 161–169 (cit. on pp. 11, 25).

[55] Weiwei Li and Emanuel Todorov. 'Iterative linear quadratic regulator design for nonlinear biological movement systems.' In: *ICINCO (1)*. 2004, pp. 222–229 (cit. on pp. 39, 41).

[56] Yiping Liu, Patrick M. Wensing, David E. Orin and Yuan F. Zheng. 'Dynamic walking in a humanoid robot based on a 3D Actuated Dual-SLIP model'. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. May 2015, pp. 5710–5717 (cit. on p. 90).

[57] Danylo Malyuta, Taylor Reynolds, Michael Szmuk, Behcet Acikmese and Mehran Mesbahi. 'Fast Trajectory Optimization via Successive Convexification for Spacecraft Rendezvous with Integer Constraints'. en. In: *AIAA Scitech 2020 Forum*. American Institute of Aeronautics and Astronautics, Jan. 2020 (cit. on pp. 68, 72).

[58] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar and N. Mansard. 'Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. May 2020, pp. 2536–2542 (cit. on pp. 7, 25, 29, 30, 46, 51, 57, 67, 77).

[59] C. Mastalli, W. Merkt, G. Xin, J. Shim, M. Mistry, I. Havoutis and S. Vijayakumar. 'Agile Maneuvers in Legged Robots: a Predictive Control Approach'. In: *arXiv:2203.07554* (2022) (cit. on pp. 25, 67).

[60] Carlos Mastalli, Wolfgang Merkt, Josep Marti-Saumell, Henrique Ferrolho, Joan Solà, Nicolas Mansard and Sethu Vijayakumar. 'A feasibility-driven approach to control-limited DDP'. In: *Autonomous Robots* 46.8 (2022), pp. 985–1005 (cit. on pp. 4, 30, 43, 53, 67, 77).

[61] David Mayne. 'A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems'. In: *International Journal of Control* (1966) (cit. on pp. 41, 77).

[62] DAVID Q. Mayne. 'Differential Dynamic Programming–A Unified Approach to the Optimization of Dynamic Systems'. In: *Control and Dynamic Systems*. Ed. by C. T. Leondes. Vol. 10. Academic Press, Jan. 1973, pp. 179–254 (cit. on p. 11).

[63] Wolfgang Merkt, Vladimir Ivan and Sethu Vijayakumar. 'Leveraging Precomputation with Problem Encoding for Warm-Starting Trajectory Optimization in Complex Environments'. In: *IEEE International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 5877–5884 (cit. on p. 90).

[64] Katja Mombaur. 'Using optimization to create self-stable human-like running'. In: *Robotica* 27 (2009) (cit. on pp. 5, 12).

[65] David Morin. *Introduction to classical mechanics: with problems and solutions*. Cambridge University Press, 2008 (cit. on p. 81).

[66] Kenta Nagano and Yasutaka Fujimoto. 'The stable wheeled locomotion in low speed region for a wheel-legged mobile robot'. In: *2015 IEEE International Conference on Mechatronics (ICM)*. Mar. 2015, pp. 404–409 (cit. on p. 80).

[67] M. Neunert, M. Stauble, M. Giftthaler, C. D. Bellicoso, J. Carius, C. Gehring et al. 'Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds'. In: *IEEE Robotics and Automation Letters* (2018) (cit. on pp. 7, 39).

[68] Michael Neunert, Farbod Farshidian, Alexander W. Winkler and Jonas Buchli. 'Trajectory Optimization Through Contacts and Automatic Gait Discovery for

Quadrupeds'. In: *IEEE Robotics and Automation Letters* 2.3 (July 2017), pp. 1502–1509 (cit. on p. 67).

[69] Tønnes F Nygaard, Charles P Martin, Jim Torresen and Kyrre Glette. 'Self-modifying morphology experiments with dyret: Dynamic robot for embodied testing'. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 9446–9452 (cit. on p. 95).

[70] Mohammad Nabi Omidvar and Xiaodong Li. 'A comparative study of CMA-ES on large scale global optimisation'. In: *Australasian Joint Conference on Artificial Intelligence*. Springer. 2010, pp. 303–312 (cit. on pp. 5, 14).

[71] David E. Orin, Ambarish Goswami and Sung-Hee Lee. 'Centroidal dynamics of a humanoid robot'. en. In: *Autonomous Robots* 35.2 (Oct. 2013), pp. 161–176 (cit. on p. 80).

[72] Alex Oshin, Matthew D Houghton, Michael J Acheson, Irene M Gregory and Evangelos A Theodorou. 'Parameterized Differential Dynamic Programming'. In: *arXiv preprint arXiv:2204.03727* (2022) (cit. on p. 13).

[73] Adam Paszke, Gross et al. 'Pytorch: An imperative style, high-performance deep learning library'. In: *Advances in Neural Information Processing Systems* (2019) (cit. on p. 46).

[74] Leszek Pecyna, Angelo Cangelosi and Alessandro Di Nuovo. 'A Deep Neural Network for Finger Counting and Numerosity Estimation'. In: *IEEE Symposium Series on Computational Intelligence (SSCI)*. 2019 (cit. on p. 32).

[75] Ioannis Poulakakis and Jessy W. Grizzle. 'The Spring Loaded Inverted Pendulum as the Hybrid Zero Dynamics of an Asymmetric Hopper'. In: *IEEE Transactions on Automatic Control* 54.8 (Aug. 2009), pp. 1779–1793 (cit. on p. 80).

[76] Joose Rajamäki, Kourosh Naderi, Ville Kyrki and Perttu Hämäläinen. 'Sampled differential dynamic programming'. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 1402–1409 (cit. on p. 11).

[77] James Blake Rawlings, David Q. Mayne and Moritz Diehl. *Model predictive control: theory, computation, and design*. Vol. 2. Nob Hill Publishing Madison, WI, 2017 (cit. on pp. 83, 84).

[78] J. Revels, M. Lubin and T. Papamarkou. 'Forward-Mode Automatic Differentiation in Julia'. In: *arXiv:1607.07892* (2016) (cit. on p. 48).

[79] Mark Schmidt, Glenn Fung and Rómer Rosales. 'Fast Optimization Methods for L1 Regularization: A Comparative Study and Two New Approaches'. en. In: *Machine Learning: ECML 2007*. Vol. 4701. ISSN: 0302-9743, 1611-3349 Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 286–297 (cit. on p. 68).

[80] C Semini, N G Tsagarakis, E Guglielmino, M Focchi, F Cannella and D G Caldwell. 'Design of HyQ - a hydraulically and electrically actuated quadruped robot'. In: *Journal of Systems and Control Engineering* 225 (2011) (cit. on p. 25).

[81] Shubham Singh, Ryan Russell and Patrick M Wensing. 'Efficient analytical derivatives of rigid-body dynamics using spatial vector algebra'. In: *IEEE Robotics and Automation Letters* (2022) (cit. on pp. 56, 95).

[82] Shubham Singh, Ryan P Russell and Patrick M Wensing. 'Closed-Form Second-Order Partial Derivatives of Rigid-Body Inverse Dynamics'. In: *arXiv:2203.01497* (2022) (cit. on pp. 56, 95).

[83] Ankur Sinha, Pekka Malo and Kalyanmoy Deb. 'A review on bilevel optimization: from classical to evolutionary approaches and applications'. In: *IEEE Transactions on Evolutionary Computation* 22.2 (2017), pp. 276–295 (cit. on pp. 13, 15).

[84] Andrew Spielberg, Brandon Araki, Cynthia Sung, Russ Tedrake and Daniela Rus. 'Functional co-optimization of articulated robots'. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 5035–5042 (cit. on pp. 5, 12, 13, 21, 22, 53–55).

[85] Theodoros Stouraitis, Iordanis Chatzinikolaidis, Michael Gienger and Sethu Vijayakumar. 'Dyadic collaborative Manipulation through Hybrid Trajectory Optimization.' In: *CoRL*. 2018, pp. 869–878 (cit. on p. 90).

[86] Y. Tassa, N. Mansard and E. Todorov. 'Control-limited differential dynamic programming'. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 1168–1175 (cit. on pp. 11, 77).

[87] Russ Tedrake. *Underactuated Robotics. Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. 2023 (cit. on pp. 6, 48, 50, 80).

[88] Robert Tibshirani. 'Regression Shrinkage and Selection via the Lasso'. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1 (1996). Publisher: [Royal Statistical Society, Wiley], pp. 267–288 (cit. on p. 68).

[89] Yvain de Viragh, Marko Bjelonic, C. Dario Bellicoso, Fabian Jenelten and Marco Hutter. 'Trajectory Optimization for Wheeled-Legged Quadrupedal Robots Using Linearized ZMP Constraints'. In: *IEEE Robotics and Automation Letters* 4.2 (Apr. 2019). Conference Name: IEEE Robotics and Automation Letters, pp. 1633–1640 (cit. on p. 80).

[90] Andreas Wächter and Lorenz T. Biegler. 'On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming'. en. In: *Mathematical Programming* 106.1 (Mar. 2006), pp. 25–57 (cit. on pp. 11, 52).

[91] Kevin Wampler and Zoran Popović. 'Optimal gait and form for animal locomotion'. In: *ACM Trans. on Graph.* 28 (2009) (cit. on pp. 4, 14).

[92] Patrick M. Wensing and David E. Orin. 'High-speed humanoid running through control with a 3D-SLIP model'. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nov. 2013, pp. 5134–5140 (cit. on p. 90).

[93] Patrick M. Wensing and David E. Orin. '3D-SLIP steering for high-speed humanoid turns'. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sept. 2014, pp. 4008–4013 (cit. on p. 90).

[94] Patrick M. Wensing and David E. Orin. 'Development of high-span running long jumps for humanoids'. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, pp. 222–227 (cit. on p. 80).

[95] Alexander W. Winkler, C. Dario Bellicoso, Marco Hutter and Jonas Buchli. 'Gait and Trajectory Optimization for Legged Systems Through Phase-Based End-Effector Parameterization'. In: *IEEE Robotics and Automation Letters* 3.3 (July 2018), pp. 1560–1567 (cit. on p. 90).

[96] Jie Xu, Tao Chen, Lara Zlokapa, Michael Foshey, Wojciech Matusik, Shinjiro Sueda and Pulkit Agrawal. 'An end-to-end differentiable framework for contact-aware robot design'. In: *arXiv preprint arXiv:2107.07501* (2021) (cit. on p. 96).

[97] Jie Xu, Andrew Spielberg, Allan Zhao, Daniela Rus and Wojciech Matusik. 'Multi-objective graph heuristic search for terrestrial robot design'. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 9863–9869 (cit. on p. 20).

[98] Yevgeniy Yesilevskiy, Zhenyu Gan and C David Remy. 'Energy-optimal hopping in parallel and series elastic one-dimensional monopeds'. In: *Journal of Mechanisms and Robotics* 10.3 (2018), p. 031008 (cit. on p. 27).

[99] Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus and Wojciech Matusik. 'Robogrammar: graph grammar for terrain-optimized robot design'. In: *ACM Transactions on Graphics (TOG)* 39.6 (2020), pp. 1–16 (cit. on p. 20).