ROBUSTNESS TO EXTERNAL DISTURBANCES FOR LEGGED ROBOTS USING DYNAMIC TRAJECTORY OPTIMISATION

HENRIQUE MANUEL MARTINS FERROLHO

Doctor of Philosophy
School of Informatics
University of Edinburgh

2022

Henrique Manuel Martins Ferrolho:
Robustness to External Disturbances for Legged Robots Using Dynamic Trajectory Optimisation
Doctor of Philosophy, 2022

SUPERVISORS:
Prof. Sethu Vijayakumar, Ph.D., FRSE
Prof. Michael Mistry, Ph.D.
Zhibin Li, Ph.D.

EXAMINERS:
Olivier Stasse, Ph.D.
Steve Tonneau, Ph.D.

ABSTRACT

In robotics, *robustness* is an important and desirable attribute of any system, from perception to planning and control. Robotic systems need to handle numerous factors of uncertainty when they are deployed, and the more robust a method is, the fewer chances there are of something going wrong. In planning and control, being robust is crucial to deal with uncertain contact timings and positions, mismatches in the dynamics model of the system, noise in the sensor readings and communication delays. In this thesis, we focus on the problem of dealing with uncertainty and external disturbances applied to the robot.

Reactive robustness can be achieved at the control stage using a variety of control schemes. For example, model predictive control approaches are robust against external disturbances thanks to the online high-frequency replanning of the motion being executed. However, taking robustness into account in a *proactive* way, i.e., during the planning stage itself, enables the adoption of kinematic configurations that allow the system as a whole to better deal with uncertainty and disturbances.

To this end, we propose a novel trajectory optimisation framework for robotic systems, ranging from fixed-base manipulators to legged robots, such as humanoids or quadrupeds equipped with arms. We tackle the problem from a first-principles perspective, and define a robustness metric based on the robot's capabilities, such as the torques available to the system (considering actuator torque limits) and contact stability constraints. We compare our results with other existing approaches and, through simulation and experiments on the real robot, we show that our method is able to plan trajectories that are more robust against external disturbances.

ACKNOWLEDGEMENTS

First and foremost, I want to thank my supervisor, Prof. Sethu Vijayakumar, for all the support and guidance during my time at the Statistical Learning and Motor Control (SLMC) Group. Prof. Sethu provided me with the right environment for me to develop myself as a researcher, and pushed me to not only explore my strengths but also to mend my weaknesses—and I shall forever be grateful for that. I also want to thank my second supervisors, Prof. Michael Mistry and Dr. Zhibin (Alex) Li for their valuable feedback throughout my PhD, especially during my annual review meetings.

Next, I want to express my heartfelt gratitude to my colleagues and mentors who have taught me pretty much everything I know about robotics: Vladimir Ivan, Wolfgang Merkt, Wouter Wolfslag, and Yiming Yang. I have learnt so much from you, and a big part of my successes were a direct result of your dedication and help.

I also want to thank the remaining members of the SLMC Group who have helped me somehow along my journey, in one way or the other: Agamemnon, Andreas, Carlo, Carlos, Chris, Christian, Daniel, Elle, Hsiu-Chin, Jaehyun, Jiayi, João, Lei, Marina, Matt, Ran, Ruaridh, Sanghyun, Serena, Songyan, Steve, Theodoros, Traiko.

In addition to these people, I have been fortunate enough to also have the help from people outside the University of Edinburgh, some of whom I have never even had the chance to meet in person. Thank you, François Pacaud, for our nice discussions and your help concerning nonlinear optimisation and the back-and-forths about the Knitro solver. Thank you, Romeo Orsolino, for your help and availability to discuss polytopic representations with applications to Robotics. Thank you, Twan Koolen and Robin Deits, for having developed pretty much the entire Julia robotics libraries and infrastructure I used during my research. Thank you, Chris Rackauckas, for the automatic differentiation and sparsity detection capabilities you have enabled within the Julia ecosystem, and for your help and chats through the Julia Slack/Discourse.

I would also like to thank the people from the University's workshop, who have fabricated the props for my robot experiments, such as the interchangeable wooden slabs used in my work with NASA's humanoid robot, and the wheel, lever, gate, and pulley used in my experiments with the quadruped robot equipped with an arm. So, thank you very much Douglas Howie, Gilbert Inkster, and Laura Ferguson.

I want to thank, from the bottom of my heart, all the help and support that my family has given me not only for the past few years, but all my life really. Thank you mum and dad, Aldinha and António, and thank you little sis, Ana Rita. I would also like to thank Xinnuo for all her love and support; it would have been much harder to go through the toughest and darkest times of my PhD without you by my side.

Finally, but not least, I would like to thank for the fun times I spent with friends I met in the UK: Bethany, Georges, Kai, Kseniia, Li Rui, Sharon; as well as my friends from Portugal: Hugo, Mafalda, and the TBR Group from Viseu.

PUBLICATIONS

Parts of the research leading to this thesis have previously appeared in the following peer-reviewed publications. Some passages have been quoted verbatim from the respective sources.

JOURNAL ARTICLES

- H. Ferrolho, V. Ivan, W. Merkt, I. Havoutis, S. Vijayakumar. 'RoLoMa: Robust Loco-Manipulation for Quadruped Robots with Arms'. Under review for *IEEE Transactions on Robotics (**T-RO**)*, 2022. (Chapter 5)
- C. Mastalli, W. Merkt, J. Marti-Saumell, H. Ferrolho, J. Sola, N. Mansard, S. Vijayakumar. 'A Direct-Indirect Hybridization Approach to Control-Limited DDP'. Under review, 2021.
- H. Ferrolho, W. Merkt, C. Tiseo, S. Vijayakumar. 'Residual Force Polytope: Admissible Task-Space Forces of Dynamic Trajectories'. *Robotics and Autonomous Systems (**RAS**)*, 2021. (Chapter 3)
- Y. Yang, W. Merkt, H. Ferrolho, V. Ivan, S. Vijayakumar. 'Efficient Humanoid Motion Planning on Uneven Terrain Using Paired Forward-Inverse Dynamic Reachability Maps'. In *IEEE Robotics and Automation Letters (**RA-L**)*, 2017.

CONFERENCE ARTICLES

- H. Ferrolho, V. Ivan, W. Merkt, I. Havoutis, S. Vijayakumar. 'Inverse Dynamics vs. Forward Dynamics in Direct Transcription Formulations for Trajectory Optimization'. In *IEEE International Conference on Robotics and Automation (**ICRA**)*, Xi'an, China, 2021. (Chapter 6)
- H. Ferrolho, W. Merkt, V. Ivan, W. Wolfslag, S. Vijayakumar. 'Optimizing Dynamic Trajectories for Robustness to Disturbances Using Polytopic Projections'. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (**IROS**)*, Las Vegas, USA, 2020. (Chapter 4)
- H. Ferrolho, W. Merkt, Y. Yang, V. Ivan, S. Vijayakumar. 'Whole-Body End-Pose Planning for Legged Robots on Inclined Support Surfaces in Complex Environments'. In *Proceedings of IEEE-RAS International Conference on Humanoid Robots (**Humanoids**)*, Beijing, China, 2018. (Chapter 2)

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, 2022

Henrique Ferrolho
16th August 2022

Dedicated to my sister and my parents,
who have always encouraged me no matter what.

CONTENTS

1 Introduction 1
- 1.1 Thesis Scope 1
- 1.2 Approach 3
  - 1.2.1 Trajectory optimiser 3
  - 1.2.2 Robustness metric 3
- 1.3 Thesis Outline 4
- 1.4 Main Contributions 5

2 Robust Whole-Body Configurations 7
- 2.1 Introduction 7
- 2.2 Related Work 9
- 2.3 Dynamic Reachability Maps Construction 11
  - 2.3.1 Upper-body iDRM 12
  - 2.3.2 Robust lower-body samples 12
  - 2.3.3 Lower-body DRM 13
- 2.4 End-Pose Planning on Inclined Terrain 14
- 2.5 Evaluation 15
  - 2.5.1 Obstacle-free benchmark 17
  - 2.5.2 Shelf benchmark 18
  - 2.5.3 Obstacle-free vs. shelf benchmark remarks 19
- 2.6 Discussion 20
- 2.7 Conclusion 20

3 Robust Trajectories for Fixed-Base Robots 23
- 3.1 Introduction 23
- 3.2 Related Work 24
- 3.3 Preliminaries 26
  - 3.3.1 Polytopes and the double description method 26
  - 3.3.2 Robot model formulation 26
  - 3.3.3 Joint force polytope and force polytope 27
- 3.4 Residual Force Polytope 27
- 3.5 Modelling Force Uncertainty 28
  - 3.5.1 Largest ball inscribed in a polytope 28
  - 3.5.2 Largest intersection with a polytope 29
- 3.6 Optimisation of Robust Trajectories 30
  - 3.6.1 Problem formulation 31
  - 3.6.2 Constraints 31
  - 3.6.3 Objectives 33
- 3.7 Experimental Results 34
  - 3.7.1 Interior-point vs. active-set methods 34
  - 3.7.2 Robustness to external disturbances 36

3.7.3 Unexpected forces vs. expected forces . . . 38
3.7.4 Summary of computational runtime . . . 38
3.8 Discussion . . . 38
3.8.1 On the scalability of our metric . . . 40
3.8.2 Mitigating the computational cost . . . 41
3.9 Conclusion . . . 41

4 Robust Trajectories for Floating-Base Robots 43
4.1 Introduction . . . 43
4.2 Related Work . . . 43
4.3 Trajectory Optimisation . . . 45
4.4 Model Formulation . . . 46
4.5 Problem Formulation . . . 47
4.5.1 Parameterization . . . 47
4.5.2 Objectives . . . 48
4.5.3 Constraints . . . 48
4.6 Robustness to Disturbances . . . 50
4.6.1 Maximum-volume ball inscribed in a polytopic projection . . . 50
4.6.2 Constraints' structure exploitation . . . 52
4.6.3 NLP reformulation . . . 52
4.7 Performance evaluation . . . 53
4.8 Experiments . . . 55
4.8.1 Robot control . . . 55
4.8.2 Description of the experiments . . . 56
4.9 Conclusion . . . 57

5 Robust Loco-Manipulation with Applications to Industry 59
5.1 Introduction . . . 59
5.2 Related Work . . . 60
5.2.1 Planning and control for quadrupeds with arms . . . 61
5.2.2 Motion robustness against disturbances . . . 62
5.3 Robust Trajectory Optimisation . . . 63
5.3.1 Robot model formulation . . . 63
5.3.2 Problem discretization . . . 64
5.3.3 System constraints . . . 65
5.3.4 Robustness against disturbances . . . 66
5.3.5 Contact switching . . . 69
5.4 Experiments . . . 70
5.4.1 System integration . . . 71
5.4.2 Repeatability test . . . 72
5.4.3 Turning a wheel and pulling a lever . . . 73
5.4.4 SUF as a tool for analysing trajectories . . . 74
5.4.5 Making and breaking contacts . . . 74
5.4.6 Robustness test with incremental weights . . . 76
5.5 Optimisation of Contact Locations . . . 77
5.6 Conclusion . . . 81

6 Inverse Dynamics as an Alternative to Forward Dynamics in Direct Transcription Formulations 83
6.1 Introduction . . . 83
6.2 Related Work . . . 84
6.3 Trajectory Optimisation . . . 85
6.3.1 Robot model formulation . . . 85
6.3.2 Problem formulation . . . 86
6.3.3 Problem constraints . . . 87
6.4 System Dynamics . . . 87
6.5 Experiments and Results . . . 89
6.5.1 Evaluation of convergence . . . 89
6.5.2 Robustness to coarser problem discretisation . . . 92
6.5.3 Optimisation with an objective function . . . 93
6.5.4 Hardware validation . . . 94
6.6 Conclusion . . . 95

7 Conclusion and Future Work 97
7.1 Framework Limitations . . . 97
7.1.1 Lack of collision avoidance . . . 97
7.1.2 Inaccurate dynamics models . . . 98
7.1.3 Too expensive for continuous re-planning . . . 98
7.2 Future Work . . . 99
7.2.1 Parallelising the framework . . . 99
7.2.2 Analysing whole-body robustness . . . 99
7.2.3 Learning the robustness metric . . . 100
7.2.4 Taking into account force-feedback during execution . . . 100
7.2.5 Maximising robustness through environment exploitation . . . 101
7.2.6 Minimising the loss of robustness . . . 101

A Appendix 103
A.1 Direct Transcription Illustrated . . . 103
A.2 Dynamics Defects Illustrated . . . 106

Bibliography 109

LIST OF FIGURES

Figure 1 NASA's humanoid robot *Valkyrie* grasping an antenna with both hands whilst standing on uneven terrain 8
Figure 2 Example of a feasible lower-body configuration for *Valkyrie* and diagram of contact forces for inclined surfaces 11
Figure 3 Pipeline overview for planning end-poses on inclined terrain . 14
Figure 4 Snapshots of *Valkyrie* walking over uneven terrain and then reaching for tools on a shelf and on a table 15
Figure 5 Inclined slabs used for our experiments with *Valkyrie* 17
Figure 6 Two arm configurations reaching the same point in task-space, and their respective force polytopes (shaped differently) 27
Figure 7 Mapping between actuation-space and task-space 28
Figure 8 Two robot applications involving force uncertainty 29
Figure 9 Different approaches for modelling force disturbances 30
Figure 10 Motion trace of trajectories computed using our framework (with different cost functions) for the *KUKA LWR* robot arm . . 34
Figure 11 Plots of the objective value and feasibility error along solver iterations for different classes of optimisation algorithms 35
Figure 12 Plots of the maximum admissible force magnitudes over time for different objective functions (IP vs. SQP) 37
Figure 13 Plots of end-effector forces that a *KUKA LWR* can counteract (regardless of the force direction) for a known motion 37
Figure 14 Profile of the test force applied to the end-effector of the *KUKA LWR* during our experiments 38
Figure 15 Plots of the total joint torques required to complete a planning task and to resist a force disturbance during execution 39
Figure 16 Picture of a legged loco-manipulation system (in our case, a quadruped robot equipped with a manipulator) 44
Figure 17 Visualisation of the contact forces and friction cones of our legged robot during a pick and place task 49
Figure 18 Terrains considered in our pick-and-place tests and their respective plots showing the robustness of different trajectories 54
Figure 19 Mean and standard deviation of the SUF at the end-effector for varying terrain inclination 54
Figure 20 Joint positions, velocities, and torques of ANYmal for a 2-seconds long trajectory on flat ground 56
Figure 21 Snapshots of our quadruped robot solving a pick-and-place task while standing on a ramp and on a skateboard 56
Figure 22 Root-mean-square error (in newtons) of the SUF for different problem discretisations 57
Figure 23 Snapshots of a quadruped robot (equipped with an arm) solving real-world tasks in an industrial setting 59

Figure 24 Summary of three different loco-manipulation NLP problems . 67
Figure 25 Snapshots of our robot grasping a hand wheel from different positions and orientations (during a repeatability test) 72
Figure 26 Pictures of the robot and plots comparing a baseline trajectory with a robust trajectory computed with our framework 73
Figure 27 Motion trace of the robot picking up an object from the floor and a plot showing how robustness is affected by each leg . . . 75
Figure 28 Plot of the SUF magnitude over time for a motion in which the robot lifts one of its legs momentarily 75
Figure 29 Snapshots of the robot lifting a bucket weighing 4.1 kg 76
Figure 30 SUF magnitude over time for a trajectory executed on the robot for lifting a heavy bucket 77
Figure 31 Comparison of different robot configurations computed with our framework version that optimises footstep locations 78
Figure 32 Feet locations, support polygons, and CoMs of trajectories optimised with different versions of our NLP problem 79
Figure 33 Snapshots of an experiment in which we disturbed the robot by pulling its end-effector 81
Figure 34 Motion traces of a *KUKA iiwa* drawing a circle, *ANYmal* jumping in-place, and *TALOS* jumping forward 90
Figure 35 Trajectory root-mean-square error for different discretisations . 92
Figure 36 Cost and feasibility error over solver iterations for the optimisation of a quadruped jump 93
Figure 37 Snapshots of a quadruped robot jumping forward 0.5 m 94
Figure 38 Snapshots of a humanoid robot performing a small jump . . . 95
Figure 39 Sketches of our robot opening two different types of doors . . 101

LIST OF TABLES

Table 1 Comparison of reachability-based end-pose planning methods 9
Table 2 Construction analysis of *Valkyrie's* upper-body map 16
Table 3 Construction analysis of *Valkyrie's* lower-body map 16
Table 4 Comparison of methods for sampling *Valkyrie's* lower-body map 18
Table 5 Analysis of *Valkyrie* end-pose planning failure 19
Table 6 Benchmark results of planning end-poses for *Valkyrie* in an environment with a shelf 19
Table 7 Analysis of end-pose planning duration and success rate in our benchmarks 20
Table 8 Solver convergence time of different objective functions in our planning framework for a *KUKA LWR* robot arm 36
Table 9 Number of function and gradient evaluations of our problems' constraints 36
Table 10 Comparison of the inference time required by each of the objective functions implemented in our planning framework . . . 40
Table 11 Inference time of computational methods for solving geometry problems relevant to our work 40
Table 12 Summary of the NLP constraints in our planning framework . 50
Table 13 Summary of the NLP constraints in our reformulation 53
Table 14 Computation time required for evaluating the function and the derivative of different NLP constraints 55
Table 15 Convergence time of different objective functions for problems discretised with a different number of mesh points 55
Table 16 Videos supplementing our work on robust loco-manipulation . 70
Table 17 Computation time and number of iterations required to solve the same planning problem for different linear solvers and for different barrier parameter update strategies in interior point methods 91
Table 18 Computation time and number of iterations required by different problem discretisations for a jumping motion on a quadruped robot 93

ACRONYMS

CoM	centre of mass
DDP	differential dynamic programming
DoF	degrees of freedom
DRM	dynamic reachability map
GIW	gravito-inertial wrench
GIWC	gravito-inertial wrench cone
iDRM	inverse dynamic reachability map
IK	inverse kinematics
IRM	inverse reachability map
LP	linear programming
MPC	model predictive control
MRP	modified rodrigues parameters
NLP	nonlinear programming
NN	neural network
RBD.jl	RigidBodyDynamics.jl
RL	reinforcement learning
RM	reachability map
RMSE	root-mean-square error
SQP	sequential quadratic programming
SUF	smallest unrejectable force
TO	trajectory optimisation
TTD	truss topology design
w.r.t.	with respect to

1

INTRODUCTION

1.1 THESIS SCOPE

Trajectory optimisation is a process that allows us to compute control trajectories as functions of time, which drive a system from an initial state towards a final state, while satisfying a given set of constraints [5]. A trajectory optimisation problem without an objective function is called a *feasibility problem*, since there is no cost function to be minimised (only a set of rules that need to be satisfied). Sometimes, a feasible solution is sufficient for solving a problem; but this is not always the case. In robotics, we often face problems where feasibility is not a sufficient criterion, and instead we are interested in finding solutions that e.g. minimise power consumption (in order to prolong battery life) or minimise the time required to complete a task. In *robust trajectory optimisation*, we are specifically looking for trajectories that are not only feasible (from a physical point of view) but also *robust* (according to a certain kind of robustness metric).

In this thesis, we tackle the problem of *robust trajectory optimisation*. More specifically, we want to be able to optimise trajectories for robots with legs and arms that need to manipulate their environment while being robust against external disturbances. This is an interesting and important problem because solving tasks in the real world involves dealing with numerous factors of uncertainty which may compromise robots' performance and integrity if left ignored.

Examples of well-known factors of uncertainty are: mismatches in the dynamics model of the system, noise in sensor readings, and communication delays. Some of these can be handled directly at the control stage thanks to clever feedback policies. However, there exist other factors of uncertainty which are more challenging to address, such as unknown external forces applied to parts of the robot—the subject of this thesis. Concrete examples of tasks where unknown external forces may cause complications are: carrying a sizeable container with liquid sloshing inside, lifting a heavy bucket with unknown contents, or turning a hand wheel which has become rusty due to exposure to the elements. In all of these examples, the actual forces applied to the robot during execution of the tasks are really hard to predict: the fluid dynamics of sloshing liquid are very complex to solve, the weight of a bucket is not easy to predict without knowing its contents, and estimating the force required to overcome the rusty axis of a hand wheel is also not trivial. Therefore, if we aim to successfully deploy robots for solving complex tasks such as these, we need methods that can handle and withstand those kinds of uncertainty.

Previous work tackling the issue of robustness against external disturbances can be split into two categories, *reactive* or *proactive*, depending on the stage at which robustness is considered. In a reactive approach, robustness is considered at the control stage; whereas in a proactive approach, robustness is considered during the planning stage, i.e., ahead of motor command execution.

Del Prete *et al.* [19] proposed a solution to improve the robustness to joint-torque tracking errors by modelling deterministic and stochastic uncertainties in joint torques within their control framework. Xin *et al.* [79] proposed a hierarchical controller in which external forces are estimated directly, with the goal of minimizing actuator torques while enforcing constraints on the contact forces. Other approaches, like [21, 42, 54], have looked at the problem from a different perspective: they see disturbances as something that the robot can handle by continuously replanning its actions, in a model predictive control (MPC) fashion. According to our categorisation, we classify all these approaches as *reactive*, because they handle robustness at the control level.

In [13], Caron *et al.* proposed the gravito-inertial wrench cone (GIWC), a feasible region used as a general stability criterion for legged robots. They used the GIWC to plan stable whole-body configurations for humanoid robots (bipedal platforms which have to actively balance on their two feet). Later, Orsolino *et al.* [55] proposed to extend the properties of the GIWC by incorporating system torque limits, i.e., the lower and upper torque bounds of the motors at the joints. They employed their method for planning stable locomotion for quadrupedal robots, thereby ensuring that the system's actuation limits were not violated during execution on the real robot. Both [13] and [55] are examples of approaches that we classify as *proactive*, since robustness is considered a priori during planning, rather than at the control stage.

Despite the progress achieved with these previous approaches, there is still plenty of room for improvement. The robustness at the control stage shown in [19] comes from deterministic and stochastic uncertainties at the joint torque level; instead, it would be nice to derive a robustness metric from the actual hardware capabilities of the system, such as the actuation limits and contact stability criteria. In [79], the actuation limits of the robot are not enforced, and there is no planner for computing elaborate whole-body motions considering full-order system dynamics. The representation proposed in [13] is efficient for testing the robust static equilibrium of legged robots, but it also neglects the system's actuation limits. Finally, the stability region presented in [55] considers actuation limits, but the technique is so expensive to compute that in their robot experiments it was only used once for the entire duration of each leg's stepping sequence.

In this thesis, our goal is to tackle these limitations. We want to be able to quantify the robustness of robots against external disturbances based on their hardware capabilities, such as actuation limits and friction cone limits. Additionally, we want to tackle the problem in a *proactive* manner, i.e., during the planning stage, and without sacrificing the ability of finding complex whole-body behaviours that take advantage of the full-order system dynamics. Next, we describe our approach to tackle this problem.

1.2 APPROACH

The key components of our approach are twofold:

1. A trajectory optimisation framework capable of computing dynamic trajectories for a variety of robots (ranging from fixed-base manipulators to floating-base quadrupeds and humanoids); and
2. An explainable metric for evaluating the robustness of robot trajectories, which can be employed in trajectory optimisation problems through an objective function (for maximising the robustness of trajectories being optimised).

1.2.1 *Trajectory optimiser*

Fundamentally, we are trying to solve trajectory optimisation problems, i.e., we are trying to find trajectories that satisfy a set of constraints (imposed by robots' hardware and the environment in which they operate) and that can be executed (not only in simulation but also on the actual robots). Therefore, we need a framework which allows us to formulate and solve trajectory optimisation problems. More specifically, we need to be able to model robots and the dynamics that govern their motion, a model for the environment in which the robots operate, and the description of the tasks that we want robots to solve within those environments.

1.2.2 *Robustness metric*

Once we have a trajectory optimisation framework that is capable of computing dynamic trajectories, we want to be able to quantify the robustness of those trajectories, for any kind of robot platform. More specifically, for a given (feasible) robot trajectory, we want to be able to evaluate the system's ability of counteracting forces applied to it throughout the motion represented by the trajectory. Therefore, we need a metric (ideally, derived from first principles) which takes the state and control commands of the robot at any given time of the trajectory and evaluates how robust the robot is against disturbances at that instant.

Let us understand how it all ties together. Once we have these two key components (i.e., the trajectory optimiser and the robustness metric) we are then able to: plan trajectories that can be executed on real robots, and evaluate the robustness of those trajectories. The interesting bit, however, comes from the *combination* of these two components, i.e., employing the robustness metric within the objective function of trajectory optimisation problems, such that we can plan trajectories that are not only feasible for execution, but that also maximise the robustness of robots against external disturbances while they move. Now, *that* is what this thesis is about!

1.3 THESIS OUTLINE

While reading this thesis, it is recommended to follow the chapter order. All chapters include a brief introduction and discussion, which get the main scientific ideas and contributions across without going into the very fine details of each work. The connecting thread between chapters is summarised in the paragraphs below. Enjoy!

Our journey starts in Chapter 2, where we look at the problem of planning whole-body configurations for a humanoid robot while taking into account torque-tracking errors. This allows us to plan robust pre-grasp bimanual whole-body poses that can serve as goals for motion and footstep planners. However, due to the serial nature of this approach, we have no means of guaranteeing robustness of the motion overall while the robot moves—only the robustness of the final quasi-static configuration.

To tackle this issue, in Chapter 3 we resort to trajectory optimisation as a means of planning motion for fixed-base robots, and we introduce a novel metric called the smallest unrejectable force (SUF) for evaluating the robustness of the robot against external disturbances. We can compute the SUF value for any point along the kinematic chain of a robot and, in short, it represents the largest force magnitude which the robot is able to counteract, assuming that force can be applied from any given direction. By defining the objective of our optimisation problem as the maximisation of the SUF over time, we show that the resulting trajectories are able to counteract larger force disturbances when compared with other approaches. Nonetheless, this approach has a limitation: it relies on numerical routines from computational geometry which are expensive to evaluate, and therefore it does not scale well to robots with legs, such as quadrupeds and humanoids.

In Chapter 4, we leverage a new scheme from computational geometry as a way of scaling our approach from fixed-base robots to legged robots. Instead of computing an exact representation of the force-rejection capabilities of the robot, we use an accurate approximation which allows faster inference of the SUF value for a particular state. Furthermore, we derive a new set of mathematical constraints to formulate a nonlinear optimisation problem where we can not only evaluate the SUF for a given trajectory, but also optimise the trajectory itself to further maximise its robustness. We show experimental results of our approach on a quadruped robot equipped with a robot arm, for a set of pick and place tasks that do not require taking steps.

In Chapter 5, we continue to develop our planning framework, increasing its complexity but ultimately extending its functionality. We add a new set of constraints to handle contact switching, therefore enabling robust *loco-manipulation* (simultaneous locomotion and manipulation). We show the robot completing a set of real-world tasks through various hardware experiments in a mock-up industrial site. We also show how our framework can optimise robot footstep locations.

In Chapter 6, we improve the core performance of our planning framework by reformulating the set of mathematical constraints responsible for enforcing physical realism in robot motion. We revisit the equations of motion that govern actuated rigid-body systems that make and break contacts with the environment, and we review the forward and inverse dynamics problems for solving the equations of motion.

Most importantly, we replace the forward dynamics constraints in our approach with inverse dynamics constraints, which significantly decreases the time and number of iterations required to find a solution to a motion planning problem.

Finally, in Chapter 7 we conclude with a list of ideas for improving our approach and interesting avenues for future research work.

1.4 MAIN CONTRIBUTIONS

The main contributions of this thesis are summarised below:

- A fast framework for planning robust and collision-free whole-body robot configurations in cluttered environments with uneven terrain. (Chapter 2)
- The *residual force polytope* — an exact representation for the set of forces that a robot can counteract, considering the full-order dynamics of articulated rigid-body systems. (Chapter 3)
- The *smallest unrejectable force (SUF)* — a robustness metric representing the magnitude of the largest force that a robot can counteract, assuming that force can be applied from any given direction. (Chapter 3 and Chapter 4)
- A trajectory optimisation framework for computing robust loco-manipulation robot trajectories, considering the whole-body dynamics of the system and the maximisation of the SUF. (Chapter 4 and Chapter 5)
- A direct transcription formulation that uses inverse dynamics to enforce physical consistency, for constrained trajectory optimisation in domains with rigid contacts. (Chapter 6)
- Experimental validation through numerous tests in full-physics simulations and on the real robot (all chapters), including deployment in a realistic scenario mimicking an industrial offshore platform (Chapter 5).

2

ROBUST WHOLE-BODY CONFIGURATIONS

Planning balanced whole-body reaching configurations is a fundamental problem in humanoid robotics on which manipulation and locomotion planners depend on. While finding valid whole-body configurations in free space and on flat terrains is relatively straightforward, the problem becomes extremely challenging when obstacle avoidance is taken into account, and when balancing on more complex terrains, such as inclined supports or steps. Previous work using paired forward-inverse dynamic reachability maps demonstrated fast end-pose planning on flat terrains at different heights by decomposing the kinematic structure and leveraging combinatorics.

In this chapter, we present an efficient whole-body end-pose planning framework capable of finding collision-free whole-body configurations in complex environments and on sloped support regions. Our main contributions are twofold: (i) the integration of contact property information of support regions into both precomputation and online planning stages, including whole-body static equilibrium robustness, and (ii) the proposal of a more informed and meaningful sampling strategy for the lower-body. We focus on humanoid robots throughout the chapter, but all the principles can be applied to legged platforms other than bipedal robots. We demonstrate our method on the NASA Valkyrie humanoid platform with 38 degrees of freedom (DoF) over inclined supports. Analysis of the results indicate both higher success rates—greater than 95% and 80% on obstacle-free and highly cluttered environments, respectively—and shorter computation times compared to previous methods.

2.1 INTRODUCTION

Humanoid robots are complex systems designed to perform dexterous tasks in environments designed and engineered for people (cf. Figure 1). While their key advantage is the ability to operate in uneven terrain and unstructured environments such as disaster sites or outdoor environments, they require active control to maintain balance, thus rendering fast planning and control challenging problems.

Directly planning motion which includes locomotion and manipulation in a single formulation is non-trivial as switching contacts and balance have to be taken into account, and available contact-implicit trajectory optimisation formulations can easily take hours to compute. Thus, it is a common approach to decompose the overall planning into subproblems [72, 81], e.g., to first plan a pre-manipulation stance and configuration (*end-pose planning* [81]) and use this as a goal for footstep [18] or acyclic contact planners [72] to generate a guide trajectory along with a sequence of contacts that navigates to the pre-manipulation stance. The final configuration further serves as an input to a whole-body motion planner [80] to compute manipulation tasks with the feet assumed to be stationary.

As such, with a focus on manipulation in complex environments, the success of these planning pipelines hinges on the quality of the final whole-body configuration. However, finding an appropriate pre-manipulation stance and configuration is not

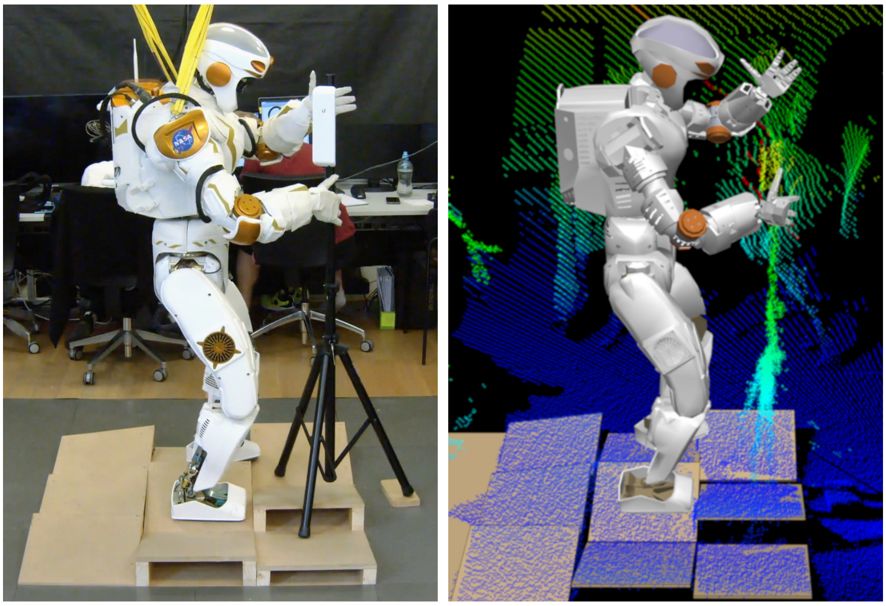


Figure 1: End-pose planning for a bimanual manipulation task where the robot has to reach for an antenna in a complex environment of sloped support surfaces. Left: photo of the robot in a pre-grasp stance. Right: visualization of the perceived 3D point cloud, fitted terrain model, and the planned whole-body configuration. Our method automatically adapts to the constraints imposed by the environment and chooses collision-free statically-balanced stance locations and whole-body configurations. These can later be used as an input to locomotion and motion planners.

trivial due to the necessity of considering collision avoidance in close proximity, contact support properties, and robot manipulability. Therefore, traditionally, a pre-grasp whole-body configuration was either provided by a human operator or based on inverse kinematics without collision avoidance, requiring the operator to manually confirm validity [24]. This often resulted in little exploration of the redundancy of high-DoF platforms and did not leverage repositionability, making it unsuitable for complex environments. Furthermore, such human-in-the-loop processes become a limiting factor for autonomous operation.

Direct optimisation-based formulations in this setting are unlikely to succeed: the problem is highly discontinuous and non-convex, with many local minima. Thus, a good initialization seed is required, especially since many constraints and objectives are expensive to compute, do not provide gradient information, or are difficult to replace with proxy constraints.

In order to exploit the redundancy of humanoids, prior work has focused on storing valid, balanced configurations along with an encoding of manipulability and reachability information during an offline preprocessing step [10, 81, 82]. However, these works are limited to flat terrains by using simple stability criteria. Furthermore, for work leveraging kinematic splits and combinatorics, no consideration is taken to ensure that recombined samples are valid and satisfy constraints, increasing requirements for online checking and planning times. Finally, the questions of required dataset sizes and good sampling strategies are not addressed.

To this end, we extend the prior work by taking into account support contact properties during both the offline preprocessing and online planning stages to enable whole-body, bimanual end-pose planning on sloped surfaces. We estimate the static equilibrium robustness of a whole-body configuration during the preprocessing stage through an informed approximation of the upper-body to ensure that recombined samples provide good initialization seeds. We further extensively evaluate different dataset sizes and sampling criteria settings in a complex, random benchmark. Finally, we embed our algorithm in a planning pipeline with multiple failure recovery mechanisms, ensuring that a valid pre-grasp configuration will be found if it exists within the dataset. With these contributions, our reachability encoding enables us to exploit the null space of the robot to efficiently compute collision-free whole-body configurations in cluttered environments. A comparison with related end-pose planning methods is shown in Table 1.

Table 1: Comparison of reachability-based end-pose planning methods.

Method	IRM [10], iDRM [81]	Paired Forward- -Inverse DRM [82]	Our method
Task constraint	Single hand ($1 \times 6D$)	Bimanual ($2 \times 6D$)	Bimanual ($2 \times 6D$)
Feet placement	x, y, yaw for mid-feet ($1 \times 3D$)	x, y, z, yaw for each foot ($2 \times 4D$)	$x, y, z, roll, pitch, yaw$ for each foot ($2 \times 6D$)
Assumptions	No slip; Horizontal support surface; Feet with constant displacement and zero *yaw* between each other.	No slip; Horizontal support surface ($roll = pitch = 0$).	No slip.

The planning framework has been validated in full-physics simulation with a model of the NASA Valkyrie humanoid robot with 38-DoF, demonstrating that the proposed method is able to find feasible pre-grasp whole-body configurations in complex environments with inclined support regions. We have further carried out hardware validation experiments of the simulated scenarios. An accompanying video is available at `https://youtu.be/tt6oYKuPI_A`.

2.2 RELATED WORK

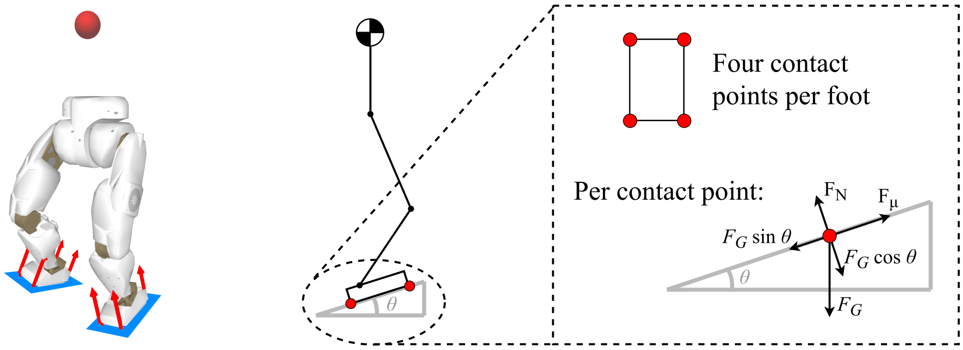
As a robotic system cannot reach every part of its workspace equally well, research has focused on characterizing the manipulability of workspace areas to find the best floating base placement by precomputing a map of the reachable workspace. The reachability map (RM) [84] records the reachable workspace regions of a fixed-base robot, allowing efficient query of whether a pose is reachable by the end-effector. Similarly, the inverse reachability map (IRM) [74] encodes feasible stances for a mobile robot given an end-effector pose. Taking into account constraints such as stability and kinematic loop closure, Burget and Bennewitz [10] extended the IRM to humanoids. They used a dense coverage of the sampling space for a single-arm reaching task through deterministic sampling in $\mathcal{C}$-space and used the results directly without post-processing, allowing violation of reach constraints according to coverage/quality of samples and requiring online collision checking. However, these methods only store the kinematic reachability with collision checking performed online, significantly contributing to long planning times.

Yang et al. [81] introduced the inverse dynamic reachability map (iDRM), which addresses this limitation by computing a custom mapping between $\mathcal{C}$-space and occupied workspace, therefore offloading collision checking to an offline preprocessing stage. This mapping was further used to initialize a reduced non-linear optimisation problem. In order to achieve good dataset coverage, the authors used deterministic sampling in the workspace. Notwithstanding, iDRM relies on two limiting assumptions: (i) the terrain of planning scenarios solely consists of a horizontal support surface; and (ii) the stance configuration always has the feet set parallel to each other and with fixed separation. These assumptions allow the explicit encoding of robust balance by only storing whole-body configurations that are in static equilibrium, i.e., configurations whose centre of mass (CoM) projection falls within their support polygon, akin to [10]. A key limitation, however, is the trade-off between problem complexity and memory available to store enough samples in order to densely cover the $\mathcal{C}$-space.

More recently, Yang et al. [82] presented a novel end-pose planning algorithm which allows covering a larger or less constrained $\mathcal{C}$-space without exhausting available memory by decomposing the kinematic structure at the pelvis link and adding a recombination of upper- and lower-body samples valid for the selected environments. The authors explored both deterministic and uniform sampling techniques. The kinematic split allows leveraging the strengths of both forward and inverse dynamic reachability maps and creates a combinatorial pool of candidate whole-body configurations. The evaluation presented in [82] provided enough evidence to support that splitting the kinematic structure at the pelvis link is the most practical approach, considering the trade-off between coverage, planning success rate, and algorithm runtime. The combinatorics of the modular maps hereby increased $\mathcal{C}$-space coverage, thus enabling the algorithm to compute pre-grasp whole-body configurations for problems requiring the feet to be placed on horizontal supports at different heights, as well as to freely place the feet relatively to each other. The latter is particularly useful for reaching tasks in environments that include small obstacles, e.g., boxes below waist height, resulting in expressive poses using the redundancy and flexibility of legged platforms, such as lunges and support steps. Nevertheless, this approach is limited to horizontal supports at different heights.

A key criterion for keeping underactuated systems balanced, especially on inclined terrains, is the robustness of the static equilibrium of contact forces. In assessing the static equilibrium of whole-body configurations, a key distinction is whether the support surfaces are flat or sloped, as the gravitational force decomposes into an orthogonal and a tangential component for non-zero inclinations (cf. Figure 2). For flat ground ($\theta = 0°$), there is no tangential component and a system is said to be in static equilibrium if the vertical projection of its CoM lies within the convex hull of the support polygon. To account for state estimation and modelling errors (e.g., elasticity in the legs) and robustness to small disturbances, a common approach is to shrink the contact polygon, thus creating the so-called support polygon.

For non-flat terrain, the CoM projection must lie within a non-linear convex set defined by the properties of each contact limb placement [9]. Different measures have been proposed in order to compare the robustness of the static equilibrium under arbitrary contacts. Caron et al. [13] proposed the capacity of a system to generate CoM accelerations within a polytope in the axial plane without a change in angular momentum. Barthélemy and Bidaud [2] proposed a robustness measure based on



(a) (b)

Figure 2: (a) Example of a feasible lower-body configuration. The red sphere above the pelvis represents an approximation of the upper-body as a lumped point mass. The red arrow markers represent the contact normals for each contact point between the feet and their virtual support region (represented in blue). (b) Diagram showing the forces involved with contact on inclined surfaces: the gravitational force decomposes into an orthogonal and tangential forces. Slippage occurs when the tangential component exceeds the frictional force $F_\mu = \mu F_N$, where μ is the friction coefficient between the foot sole and the contact surface materials.

the radius of the largest hypersphere centred at the gravito-inertial wrench (GIW) and fully contained inside the GIW cone. Subsequently, Del Prete et al. [20] proposed to account for robustness to errors in the contact-force tracking, i.e., to prevent the forces necessary to maintain equilibrium from being too close to the boundaries of the friction cones. The authors proved that the CoM-projection method extends to *quasi-flat* terrains but comes with a larger number of false negatives as, for example, the height difference between the contact points increases. Additionally, they propose an algorithm that outperforms (in terms of computation time) previous approaches by approximating friction cones with polytopes defined by a set of linear inequalities.

A robustness metric is essential to assess stability on uneven terrains. We will now describe how such a metric can be integrated with the reachability map at creation time to filter out the unstable poses and to reduce the map size.

2.3 DYNAMIC REACHABILITY MAPS CONSTRUCTION

A reachability map encodes the reachable poses of a robot link with respect to a given frame. The forward and inverse reachability maps, i.e., RM and IRM, thus only differ in the frame by which they are defined: the former in base frame, and the latter in end-effector frame. In simple terms, the forward RM encodes *how well a robot can reach different regions of its workspace*, whereas the IRM encodes *where the robot base can be placed, given a grasping target*. The major distinction between RM / IRM and their *dynamic* versions is the additional mapping information concerning *occupation* and *reach* lists. Both the dynamic reachability map (DRM) and iDRM involve an initial step where the workspace is discretized into a bounded 3D voxel grid, $\mathbb{V}$. The resolution at which discretization is applied depends on the application's final purpose, and ultimately, represents a trade-off between memory usage, computation times, and planning accuracy.

Similarly to previous work, henceforth we will designate each map's reference frame link as *root link*, i.e., the base link for DRM, and one of the end-effector links for the iDRM. We will designate the mapped frames as *tip links*, i.e., the end-effector links (feet) for the lower-body DRM, and the base link and the remaining end-effector link for iDRM.

We discretize the reachable workspace during the offline preprocessing stage, and we create two distinct lists for each voxel $v \in \mathbb{V}$: the *occupation list*, O_v, which maps to samples intersecting with the voxel v, and the *reach list*, R_v, containing the indices of the samples for which one of the *tip links* falls within the voxel v. These lists are then used online to efficiently invalidate samples that are in collision or cannot reach the target. For further detail, please refer to [82].

2.3.1 *Upper-Body iDRM*

The same sampling process for the upper-body iDRM from [82] is used. There exist two distinct variants of upper-body datasets: constrained and unconstrained. The unconstrained variant samples robot configurations within the full scope of the $\mathcal{C}$-space—this includes upper-body configurations where, in the specific case of a humanoid, the arms reach behind the robot. The constrained variant uses rejection sampling to discard samples whose tip links are not comfortably reaching a bounded region of space in front of the robot—the rationale behind this being favouring the front side of the robot for increased manipulability where most sensor data is captured.

2.3.2 *Robust Lower-Body Samples*

A limiting assumption in our previous work [82] was that support regions would always be flat, even though they could be positioned at different heights, e.g., steps. As such, the lower-body dataset in that work consisted solely of configurations in which both feet were constrained to be horizontal (i.e., $roll = pitch = 0$). Nonetheless, in order to eliminate the need for that assumption, a lower-body dataset comprising non-horizontal feet is required.

A robustness measure for the equilibrium of a specified CoM position can be computed as proposed in [20] by solving the following linear programming (LP) problem:

$$\begin{aligned} \text{find} \quad & \boldsymbol{b}, b_0 \\ \text{maximize} \quad & b_0 \\ \text{subject to} \quad & G\boldsymbol{b} = D\boldsymbol{c} + \boldsymbol{d}, \\ & \boldsymbol{b} - \mathbb{1} b_0 \geq \boldsymbol{0}, \end{aligned} \tag{1}$$

where $\boldsymbol{b}$ is a vector of coefficients of the contact force generators ($f = G\boldsymbol{b}$), $b_0 \in \mathbb{R}$ is a scalar parameter proportional to the robustness measure, $\boldsymbol{c} \in \mathbb{R}^3$ is the CoM position, G is the matrix whose columns are the GIW generators, D is the matrix mapping the CoM position to GIW, and $\boldsymbol{d}$ is the 6D vector containing the gravity component of the GIW. These variables can be computed from kinematic and dynamic properties of the robot model.

Since the kinematic structure has been split into upper- and lower-body parts, a whole-body configuration is not available during preprocessing, and consequently, neither a CoM position—albeit the masses of the platform's parts are known. In order to circumvent this problem, a lumped point mass can be added above the pelvis level to the lower-body model of the robot to approximate the upper-body CoM position (cf. Figure 2a). The position of the lumped point mass can be approximated by averaging the CoM positions of all upper-body samples contained in an upper-body dataset created beforehand:[1]

$$\boldsymbol{p}_{\text{lumped mass}} = \frac{1}{n} \sum_{i=1}^{n} f_c(\mathrm{Y}_i) \tag{2}$$

where $\boldsymbol{p}$ is the approximated position of the lumped point mass being calculated, Y is an upper-body dataset with n samples, and f_c is a function which returns the CoM position of a dataset entry Y_i. At last, after this addition, a whole-body CoM position can be estimated, in turn allowing the calculation of an approximated robustness of a whole-body configuration static equilibrium. Despite such setup being only an approximation, it indeed provides a close estimation of the likely CoM for the whole-body robustness.

It is important to note that the equilibrium approximation depends on the direction of the gravity vector with respect to the floating base. Thus, in order to reuse the approximation computed offline during online recombination, the *roll* and *pitch* components of the pelvis are set to zero while maintaining the *yaw* component. This reduces the uncertainty and mismatch of the actual (vs. approximated) robustness measure when recombining individual upper- and lower-body samples to a whole-body candidate configuration.

2.3.3 *Lower-Body DRM*

The lower-body sampling process described in [82], similarly to [10], is carried out deterministically by stepping through joint range using fixed increments while constraining the feet to the flat surface. Instead, we use a pseudo-random rejection sampling procedure coupled with the static equilibrium robustness measure described in 2.3.2 to generate the datasets analysed in this study. A sampled lower-body configuration is admitted for storage in the dataset being generated if and only if its static equilibrium robustness is greater than or equal to a certain threshold, R_{min}. This is in order to assure that (a) the forces necessary to maintain the equilibrium of the stored samples are not too close to the boundaries of the linear approximation of the friction cones, and (b) the amount of torque each joint is allowed to exert in order to achieve the pose is limited. We explored different R_{min} values during our evaluation and present the results in Table 4.

Additionally, the sampling strategy we suggest also considers the distance between feet before admitting a configuration to be stored in a dataset. This is due to the walking controller employed in the task-planning framework we use: limiting the maximum distance between the feet ensures that the whole-body configurations returned by the end-pose planner can be transitioned into. Since the method we present in this chapter can be generalized to any legged platform, even beyond bipedal systems, this decision variable is left out to be dictated by the targeted robot platform.[2]

---

[1] We used the Y_{4M_c} upper-body dataset presented in Table 2 to compute an approximation of the lumped point mass position.

2.4 END-POSE PLANNING ON INCLINED TERRAIN

Figure 3 highlights the pipeline we are proposing for whole-body end-pose planning on inclined terrains. Two essential prerequisites for the pipeline to function are two datasets generated offline, storing upper- and lower-body samples. Their computation has been addressed in 2.3.1 and 2.3.3.

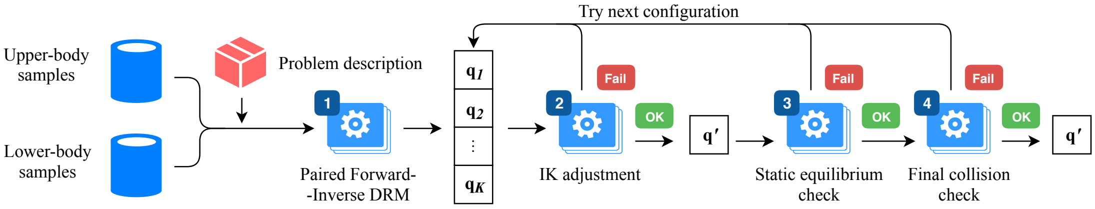


Figure 3: Overview of the proposed planning pipeline. The numbered blocks with a gear represent the key *stages* along the pipeline.

A planning request is triggered when a problem description is fed into the pipeline. This includes a bimanual grasping target and the environment information—including scene obstacles and support regions. Firstly, in *Stage 1*, we calculate a set of candidate whole-body configurations that are collision-free and satisfy the task constraints. This calculation is performed by the Paired Forward-Inverse DRM module (see [82] for more details). After the set is complete, the candidates are sorted according to the following cost function:

$$f(\boldsymbol{q}) = w_T ||T_{rhand}(\boldsymbol{q}^*) - T_{rhand}(\boldsymbol{q})|| + \frac{w_R}{R(\boldsymbol{q})}, \tag{3}$$

where $R(\boldsymbol{q})$ is the static equilibrium robustness of the whole-body configuration $\boldsymbol{q}$ (i.e., b_0 in optimisation problem 1), w_T is a weight for the distance between a configuration's grasping tip $T_{rhand}(\boldsymbol{q})$ and the target pose $T_{rhand}(\boldsymbol{q}^*)$, and w_R is another weight for the robustness measure of the static equilibrium of configuration $\boldsymbol{q}$. Once sorted, candidate configurations are tested in order. Planning terminates when a whole-body candidate successfully reaches the end of the pipeline, or the candidate set is exhausted.

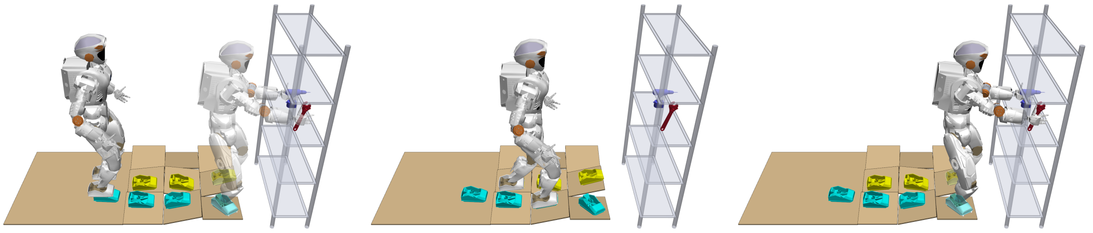
Stage 2 consists of an inverse kinematics (IK) adjustment to ensure the target reaching constraints are satisfied and that the feet are in perfect contact with the support regions.[3] Because the robust static equilibrium constraint is not part of the IK formulation problem, the adjusted configuration, $\boldsymbol{q}'$, must go through *Stage 3*, where its static equilibrium robustness is computed. Finally, a full collision check has to be performed over $\boldsymbol{q}'$ in *Stage 4*. The reason for this being that, if the IK adjustment involves a considerable kinematic displacement, the occupancy encoding of candidate $\boldsymbol{q}$ (which lead to $\boldsymbol{q}'$) might no longer hold. Stages 2, 3 and 4 are repeated for each candidate configuration selected in *Stage 1* until one of them successfully passes all the stages.

---

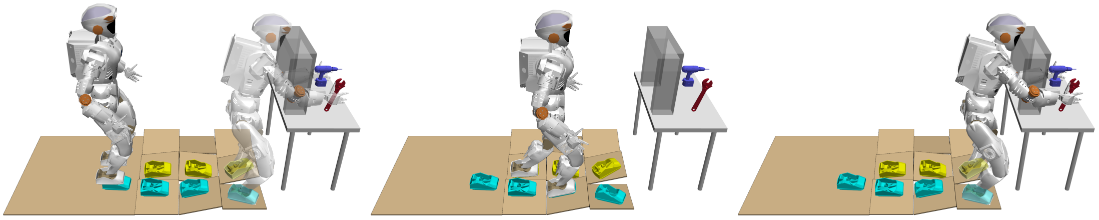
[2] For the purpose of this study, we have opted to use a threshold of 0.5 m for the maximum x-y distance between feet of the NASA Valkyrie humanoid robot.

[3] Here, we use the optimisation-based IK from Drake [68].

Figure 4 shows task snapshots of the results obtained after embedding our proposed planning pipeline into our higher-level control framework. Given a bimanual grasping target, the previously described pipeline finds a valid stance location and a reachable whole-body configuration. Afterwards, the feet locations of this result are passed on to a footstep planner [18] to generate a walking trajectory, bringing the robot to the desired stance. Finally, after having arrived at the computed standpoint, a whole-body motion planner [80] is invoked to generate a collision-free whole-body motion to reach the desired pre-grasping configuration.



(a) Grasping a drill and a wrench in a shelf compartment on complex terrain with multiple support regions at different inclinations.



(b) Grasping a drill and wrench similarly to the previous task but through a narrow frame atop a table. This scenario is purposefully built in such a way that the set of feasible solutions is comprised *only* of whole-body configurations where the left arm of the robot passes through the frame on top of the table.

Figure 4: Snapshots of task stage progression in time on two scenarios with the same terrain but different surroundings. The robot needs to reach for a drill and a wrench in a shelf compartment and atop a table, respectively. The spatial location of the targets are the same in both scenarios. Without change, our method automatically adapts to the different surroundings and returns suitable whole-body configurations.

2.5 EVALUATION

Based on our previous research and on the trade-off between memory consumption and mapping completeness, we have chosen to discretize the workspace into 10 cm voxels. Furthermore, at *Stage 1*, a whole-body configuration, $\boldsymbol{q}$, must respect the following constraints in order to be admitted as a candidate: (i) the z-distance between each foot and its support region must be less than half the discretized workspace resolution—5 cm in this case; (ii) the orientation difference between each foot and its support region, i.e., the angle measured between the normal vector of the support region and the foot's normal, must be less than a certain tolerance—we used 0.25 rad. At *Stage 2*, we set the following task-specific tolerances for satisfying the equality constraints: 1 mm for hands and feet positions; 10^{-5} rad for hand orientations; and

10^{-3} rad for feet orientation. At *Stage 3*, we use four generators per contact and the contact friction coefficient μ is set to 0.3—which is comparable to half the friction between rubber and dry concrete on clean and dry surfaces. These are the same parameters used for computing the lower-body dataset. The work presented in this chapter was implemented using EXOTica [39] and all evaluations carried out in a single-threaded process on a 4.00 GHz Intel Core i7-6700K CPU with 32 GB 2133 MHz RAM.

Table 2: Upper-body map construction analysis.

Designation	Constrained	No. of samples	Build time (hh:mm:ss)	Size (MB)
Υ_{500K_c}	Yes	5×10^5	02:54:35	645
Υ_{1M_c}	Yes	1×10^6	05:39:21	1290
Υ_{2M_u}	No	2×10^6	04:20:23	2183
Υ_{2M_c}	Yes		10:45:49	2186
Υ_{4M_u}	No	4×10^6	08:40:46	4366
Υ_{4M_c}	Yes		21:34:39	4372

Table 2 shows the details of the upper-body datasets we have generated for this work. An upper-body dataset is designated by the symbol Υ followed by the number of samples it contains and its type (c - constrained, u - unconstrained) in subscript. The actual size of datasets vary for different robot models as the majority of space is used to encode workspace occupancy. The sizes shown in the table correspond to the model of the 38-DoF NASA Valkyrie humanoid robot. It is clear that both the time it takes to build the dataset and the size it occupies in disk grows with the number of samples. Moreover, it is noticeable that, due to the nature of the rejection sampling process, *constrained* dataset variants take longer to be built than the *unconstrained* ones.

Table 3: Lower-body map construction analysis.

Designation	R_{min}	No. of samples	Build time (hh:mm:ss)	Size (MB)
Π_{1K}	10	1 000	00:07:14	1
Π_{10K_1}	0	10 000	00:30:53	7
Π_{10K_2}	10		01:11:24	7
Π_{10K_3}	20		04:27:00	8
Π_{50K_1}	0	50 000	02:38:17	38
Π_{50K_2}	10		05:59:24	37
Π_{11K}	-	11 812	00:01:45	7
Π_{151K}	-	151 503	00:22:17	94

Table 3 shows the details of the lower-body datasets we have generated for this study. A lower-body dataset is designated by the symbol Π followed by the number of samples it contains in subscript. Datasets Π_{11K} and Π_{151K} have been generated in a deterministic fashion, by incrementally stepping through joint values and maintaining the feet horizontal. All datasets other than Π_{11K} and Π_{151K} have been generated using our new lower-body sampling strategy. A correlation between the time taken and the strategy used to create a dataset can be observed: the proposed rejection sampling method takes longer, as large quantities of low quality samples are rejected. Moreover, the higher the threshold for R_{min}, the longer the dataset generation time. However, as sampling takes place offline, dataset build times can often be neglected.

Figure 5 shows the testbed we have modelled in order to carry out evaluation tests of our planning method. It consists of 18 support regions with different inclinations. The layout is reconfigurable, enabling us to change the arrangement of the support regions if necessary.

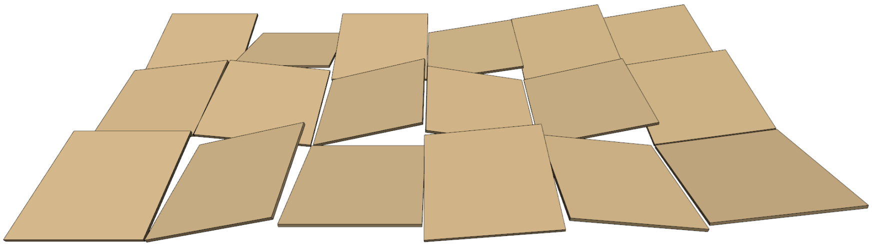


Figure 5: Testbed of inclined support regions used during the benchmarking tests of our planning method. The testbed is organized in three rows, each with six supports. Each support is 40 cm × 40 cm and has an inclination of either 10° or 15°.

2.5.1 *Obstacle-Free Benchmark*

We created a benchmark to evaluate the performance of our method in obstacle-free environments on the aforementioned testbed (cf. Figure 5). The benchmark tests a total of 1000 planning requests. Each request is generated by sampling a pseudo-random upper-body configuration using the same constraints as during dataset creation. Afterwards, a random *yaw* is applied to the configuration. Finally, the pose is translated to a random location in the testbed, subject to the condition of its pelvis projection lying within the shrunk x-y boundaries of the testbed.

Table 4 shows the benchmark results of different lower-body datasets while using the same constrained upper-body dataset, Υ_{2M_c}. The two bottom rows of the table (i.e., Π_{11K} and Π_{151K}) concern lower-body datasets generated according to the methodology of our previous work. The remaining rows concern lower-body datasets generated with our most recent approach. The benchmark results show that our sampling strategy significantly outperforms the previous, which is reflected under the request success rate column. The number of "Total Candidates" shows how well the dataset matches the request criteria during the benchmark. The smallest dataset tested, Π_{1K}, produces very few candidates, which results in short computation times but also reduces planning success rate. Moreover, the quality of the candidates is reflected in the number of "Rejected Candidates", which indicates how many candidate solutions were discarded further down the pipeline until an actual feasible solution was found. The rejection rates of the datasets generated using the proposed method (i.e., Π_{10K_i}

and Π_{50K_i}) are an order of magnitude lower than the ones generated in our previous work (i.e., Π_{11K} and Π_{151K}). Finally, the results also show that the minimum robustness threshold R_{min} does not affect the overall request success rate. However, the robustness threshold does affect the resilience of the system to external disturbances. Yet, this was not evaluated in this benchmark.

Table 4: Comparison of different lower-body sampling methods using a constrained upper-body dataset, Y_{2M_c}.

Dataset	No. of samples	R_{min}	Success rate (%)		Candidates		Durations (ms)		
			Stage 1	Request	Total	Rejected	UB filter	IK	Request
Π_{1K}	1 000	10	82.9	72.5	9.4 ± 20.5	0.69 ± 1.38	87 ± 17	18.8 ± 2.3	191 ± 77
Π_{10K_1}	10 000	0	97.5	**94.0**	106.5 ± 304.1	2.01 ± 3.79	90 ± 18	18.2 ± 1.9	436 ± 303
Π_{10K_2}		10	97.3	**93.8**	116.1 ± 344.3	1.91 ± 4.45	89 ± 19	18.4 ± 2.6	457 ± 369
Π_{10K_3}		20	96.9	**93.4**	113.9 ± 245.6	1.75 ± 4.05	86 ± 17	18.0 ± 2.2	421 ± 268
Π_{50K_1}	50 000	0	99.4	**98.6**	441.5 ± 1017.7	2.01 ± 3.74	88 ± 17	18.1 ± 3.4	1094 ± 898
Π_{50K_2}		10	99.3	**98.9**	567.3 ± 1874.2	2.37 ± 4.04	87 ± 18	18.0 ± 2.3	1244 ± 1528
Π_{11K}	11 812	-	26.1	23.8	544.7 ± 1690.1	19.27 ± 118.30	84 ± 17	18.1 ± 2.9	1370 ± 3948
Π_{151K}	151 503	-	28.6	27.4	7300.8 ± 25 004.4	108.86 ± 1198.77	85 ± 16	17.5 ± 1.5	9344 ± 29 810

Table 5 provides some valuable insight regarding the pipeline stages at which a test candidate got rejected. That is, from all the “Rejected Candidates” in Table 4, Table 5 breaks down at which point in the pipeline a candidate was rejected. With this information, we can understand why the candidates failed and which is the most predominant factor for rejecting a candidate. Each entry contains a colour bar with, from left to right, the percentage of test-candidates rejection during IK adjustment (*Stage 2*, purple), static equilibrium check (SEC) (*Stage 3*, pale blue), and final collision check (FCC) (*Stage 4*, dark blue). Failure due to the IK adjustment is minimal (0.8 %, in the Π_{50K_1} entry, is the greatest percentage in the tests we carried out). The most frequent stage at which a sample gets rejected is the SEC. Furthermore, a correlation exists between the minimum robustness threshold, R_{min}, and the SEC-FCC rejection distribution: as the robustness storage criteria gets more demanding, rejection starts to shift from occurring during the SEC to the FCC stage (cf. Π_{10K_1}, Π_{10K_2}, and Π_{10K_3}). The conclusion to draw from this table is that the bottleneck is due to the adjustment performed by the IK solver during *Stage 2*, which returns a whole-body configuration that is no longer in robust static equilibrium.

2.5.2 *Shelf Benchmark*

We created a second benchmarking test to evaluate the performance of our method in environments cluttered with obstacles. For that, we make use of the same testbed terrain of the previous benchmark, with the addition of a shelf. The benchmark routine translates the shelf in incremental steps about the testbed, three steps in the y-axis direction, and 21 steps in the x-axis direction. Finally, for each position of the shelf, 16 bimanual requests located inside one of the shelf compartments are passed on as inputs to our planning pipeline. This amounts to a total of 1008 requests during a whole benchmark session.

Table 5: End-pose planning failure analysis using a constrained upper-body dataset.

Name	Failure stage decomposition (%)
Π_{1K}	SEC: 58.0% FCC: 41.5%
Π_{10K_1}	SEC: 79.3% 20.7%
Π_{10K_2}	SEC: 74.2% FCC: 25.7%
Π_{10K_3}	SEC: 70.8% FCC: 29.0%
Π_{50K_1}	SEC: 78.6% 20.6%
Π_{50K_2}	SEC: 75.1% FCC: 24.8%

Table 6 shows a detailed analysis of the benchmark results. The meaning of data presented under each column follows the same convention as in Table 4—please confer 2.5.1 for a detailed description of what is listed under each column.

Table 6: Detailed analysis of benchmark in the "shelf" environment.

Upper-body	Lower-body	Success rate (%)		Candidates		Durations (ms)		
		Stage 1	Request	Total	Rejected	UB filter	IK	Request
Υ_{2M_c}	Π_{10K_2}	64.1	57.2	16.6 ± 65.7	0.37 ± 0.93	95 ± 7	17.8 ± 1.5	195 ± 98
	Π_{50K_2}	77.5	74.1	105.0 ± 547.4	0.63 ± 1.28	94 ± 7	17.4 ± 1.4	299 ± 427
Υ_{4M_c}	Π_{10K_2}	68.9	63.0	22.3 ± 123.3	0.41 ± 1.72	166 ± 15	18.1 ± 1.9	318 ± 168
	Π_{50K_2}	84.9	**81.8**	104.9 ± 564.3	0.73 ± 2.21	152 ± 12	17.4 ± 1.7	382 ± 445

By inspecting Table 6 we can observe that increasing the number of samples in the lower-body dataset (e.g., using Π_{50K_2} instead of Π_{10K_2}) significantly increases the total number of candidates per request. This is ideal, since a wider set of available whole-body configurations translates into more variability, which in turn increases the chances of successfully finding a solution when dealing with cluttered environments—cf. the percentage increase in the "Request, Success rate" column. The downside of having more options available is the time increase required to process and choose the best solution. This downside can be prevented by employing a good cost function to sort candidate poses. The "Rejected Candidates" column shows that, on average, fewer than one candidate had to be re-tested beyond the initial candidate until a feasible solution was found. In other words, for most of the end-pose planning requests, the candidate on the top of the sorted candidates list was indeed a valid (and the chosen) solution. Thus, providing enough evidence to support that the employed cost function is reliable and adequate.

2.5.3 *Obstacle-Free vs. Shelf Benchmark Remarks*

Table 7 presents a comparison between the results of the obstacle-free and shelf benchmarks for upper- and lower-body datasets of different sizes. Results show that dataset size has a greater impact on success rates for cluttered environments rather than obstacle-free environments. Note that the planning request durations tend to be much longer for the obstacle-free environment when compared to the shelf environment. This is due to fewer configurations being invalidated by the configuration-to-workspace-occupancy encoding and thus more candidates having to be scored and ranked.

Table 7: Benchmark request *success rate* and *duration* analysis for varying dataset sizes. Each cell contains two lines: (1) results concerning the "obstacle-free" environment, and (2) results obtained for the "shelf" environment.

Lower-Body	Upper-body			
	Υ_{500K_c} (500K)	Υ_{1M_c} (1M)	Υ_{2M_c} (2M)	Υ_{4M_c} (4M)
Π_{1K} (1K)	43.5% / 78 ± 54 ms 7.3% / 71 ± 33 ms	55.7% / 133 ± 77 ms 11.8% / 94 ± 41 ms	72.5% / 191 ± 77 ms 24.8% / 153 ± 60 ms	80.6% / 355 ± 138 ms 25.7% / 222 ± 71 ms
Π_{10K_2} (10K)	81.7% / 184 ± 121 ms 34.3% / 104 ± 74 ms	90.3% / 271 ± 214 ms 38.2% / 122 ± 65 ms	93.8% / 457 ± 369 ms 57.2% / 195 ± 98 ms	97.7% / 733 ± 507 ms 63.0% / 318 ± 168 ms
Π_{50K_2} (50K)	92.9% / 428 ± 655 ms 52.3% / 138 ± 121 ms	96.0% / 787 ± 756 ms 47.2% / 159 ± 142 ms	98.9% / 1244 ± 1528 ms 74.1% / 299 ± 427 ms	99.5% / 2294 ± 2446 ms 81.8% / 382 ± 445 ms

2.6 DISCUSSION

Splitting the kinematic structure of non-homogeneous legged robots leads to a greater coverage of the $\mathcal{C}$-space thanks to the the combinatorial nature of the recombination step. However, this also means there is a greater number of whole-body candidate poses (which grows exponentially with dataset size) that need to be checked and ranked, which might potentially slow down the planning process.

In this work, we addressed this challenge through an informed and effective cost function, leaving further speed-ups through parallelization to future work. An analysis of the different failure stages during an extensive benchmark shows the main bottleneck is due to adjusted samples no longer meeting the static equilibrium criteria. A possible step to addressing this issue is to explicitly include the robustness measure in the formulation of the IK optimisation problem (*Stage 2*). However, finding a differentiable proxy metric is non-trivial and an interesting avenue for future work.

Finally, it would be desirable to plan footsteps in a continuous fashion, akin to [23], in order to overcome the uncertainty of accumulated state estimation drift during locomotion, and to actively re-plan walking trajectories to avoid dynamic obstacles.

2.7 CONCLUSION

In this chapter, we presented a method for whole-body end-pose planning on inclined supports in complex environments, taking contact properties such as slope and friction into account. We analysed the impact of including static equilibrium robustness as part of the sampling heuristics for the offline preprocessing stage to improve dataset quality, leading to reduced planning times and increased algorithm success rates while using smaller datasets and covering larger state spaces. In particular, offloading the computation of static stability properties to the preprocessing stage allows the algorithm to propose candidate poses with higher quality and chance of success. We validated our approach with the 38-DoF NASA Valkyrie humanoid in full physics simulation and in real-world experiments, whereby we showed that the proposed method results in physically-achievable robust configurations on inclined surfaces.

In the next chapter, we are going to continue our investigations on robustness metrics for robotics applications. More specifically, we are going to propose a new robustness metric based on first principles, and we are going to include that metric in a trajectory optimisation framework, which will allow us to optimise robot motion rather than just a single whole-body configuration.

3

ROBUST TRAJECTORIES FOR FIXED-BASE ROBOTS

In this chapter, we propose the *residual force polytope*: a representation for the set of forces a robot can counteract while considering the full-order system dynamics. Given the nominal torques required by a dynamic motion, this representation models the forces which can be sustained without interfering with that motion. The residual force polytope can be used to analyse and compare the set of admissible forces of different trajectories, but it can also be used to define metrics for solving optimisation problems, such as in trajectory optimisation or system design. We demonstrate how such a metric can be applied to trajectory optimisation and compare it against other objective functions typically used. Our results show that the trajectories computed by optimising objectives defined as functions of the residual force polytope are more robust to unknown external disturbances. The computational cost of these metrics is relatively high and not compatible with the short planning times required by online methods, but they are acceptable for offline motion planning.

3.1 INTRODUCTION

Robots have well-defined actuation limits and, usually, a clear definition of the task to be completed, but the conditions of the environment in which they operate may be a source of uncertainty. Besides environmental uncertainty, robots can also be affected by sensor noise, signal delay, and model mismatches, and these sources of error are often addressed with a feedback controller. However, controllers have their own limitations, and their ability to execute a motion depends not only on the complexity of the trajectory but also on the control authority available to track the motion plan and counteract any external disturbances at the same time. In general, there are two ways to improve robustness:

- During *control* ([19, 58, 79]), by increasing robustness when executing a nominal motion plan.
- During *planning* ([48, 55]), by considering uncertainty and robot capabilities to find trajectories with larger feasibility regions that can be exploited by controllers.

Being robust at the control stage does not necessarily result in a robust execution overall if the commanded motion is not robust itself. In fact, a bad motion plan will inherently compromise the robustness strategy of a controller. Despite the importance of robust controllers, we believe that ensuring robustness at an earlier stage is paramount for reliable deployment of robotic systems and, for that reason, this chapter tackles the problem of increasing robustness during planning. While predicting and modelling uncertainty at the planning stage is difficult, we can exploit well-known capabilities and limitations of a system to optimise highly-robust trajectories. We argue that, by explicitly taking into account robot-specific capabilities and computing the set of admissible forces in task-space, we can define a metric as a function of that set to find trajectories that are more capable of resisting unexpected forces. To that end, we first propose a representation of admissible task-space forces taking into account the dynamics of the system (i.e., not limited to quasi-static scenarios). Then, we test our hypothesis by defining an objective function based on our proposed representation, and compare it against other established objectives. We use a direct method to formulate the optimal control problems where those objectives are employed. This allows for straightforward definition of mathematical constraints (in the form of equalities and inequalities) on either state or control variables, as well as the computation of the force/torque capabilities of the robot as a polytope, for any of the trajectory points discretized.

The main contributions of this work are:

1. Proposal of a representation of all the realizable forces given a configuration, a vector of forces/torques, and the system dynamics: the *residual force polytope*.
2. Elucidation of two models for representing force uncertainty and their combination with the residual force polytope for optimising robust trajectories.
3. Comparison of several objective functions from related work with an objective function based on the residual force polytope for dynamic trajectory optimisation.

3.2 RELATED WORK

In previous work [26], we exploited the kinematic redundancy of robots with many degrees of freedom in order to select configurations more robust to torque-tracking errors. Our method indexes a previously-sampled database efficiently, but it does not optimise the robot's ability to resist unknown external disturbances and is limited to choosing single configurations. In contrast, this chapter focus on robustness against unexpected forces and demonstrates how the states of the system can be optimised for entire trajectories to achieve more robust motions.

Other researchers have also exploited kinematic redundancy to improve robot capabilities. For example, Yoshikawa [83] proposed the *force manipulability ellipsoid* to take into account the ability to apply and resist forces based on the robot geometry. Building on top of this concept, Jaquier *et al.* [40] proposed a control scheme which tracks desired profiles of manipulability ellipsoids, either as the main task or as a secondary objective. Haviland and Corke [35] presented a resolved-rate motion control also making use of manipulability ellipsoids: their real-time controller tracks the Cartesian velocity of the end-effector while maximizing the manipulability of the system. Both [40] and [35] employ manipulability metrics during the control stage, but such metrics can also be employed for motion planning. An example of this is Chu *et al.*'s [17] path planning algorithm for multi-arm robots: their approach uses Yoshikawa's measure of manipulability to avoid kinematic singularities while planning complex and collision-free manoeuvres. Despite the widespread use of manipulability ellipsoids in robotics applications, the real manipulability of actuated systems is a convex polytope which cannot be represented accurately using an ellipsoid, i.e., the ellipsoid is only an approximation. Additionally, manipulability ellipsoids make it difficult to capture and incorporate descriptions of other system constraints. In contrast, the polytope of admissible forces that we propose in this chapter is not an approximation, and allows for easy integration of extra constraints through polytope manipulation.

The ability to manipulate and intersect polytopes can be very useful. For example, it allows the aggregation of multiple constraints into a single description of necessary conditions for feasibility of a system, provided that each individual constraint can be modelled in the form of a polytope. For example, Audren and Kheddar [1] extended 2D stability regions to 3D by accounting for possible centre of mass (CoM) accelerations in order to achieve robust multi-contact stability in whole-body posture generation. Orsolino *et al.* [55] proposed the actuation wrench polytope and intersected it with the contact wrench cone [36] to create the feasible wrench polytope. The actuation wrench polytope is a representation of all the wrenches a robot can generate given its actuation limits. However, it is limited to quasi-static scenarios. In this chapter, we propose a new representation that accounts for the dynamics of the system and the torques required by a nominal motion, hence, providing a description of the admissible forces for dynamic scenarios. In [55], the feasibility polytope was used to optimise the CoM position of a quadruped's static crawl gait. However, due to the required computational cost, they calculated the polytope once at the beginning of the optimisation and used that as a constant approximation thereafter. As such, computing the exact polytope at every point of the trajectory during optimisation and the impact of this approach on performance are two important aspects that have not been studied before, and which we address in this work. It is also worth noting that the optimisation problem in [55] optimises four variables in time (the positions and velocities of the centre of mass in the xy-plane), while our problem optimises 21 variables in time (the state and control inputs of a 7-DoF robot arm) and is therefore significantly more complex.

The idea of improving the robustness of robot motions using trajectory optimisation has been explored before: Manchester and Kuindersma [48] presented an algorithm that incorporates linear feedback, bounded disturbances, and a penalty for closed-loop deviations from a nominal trajectory. A key advantage of their method is that the resulting control trajectories avoid *bang-bang* control, and leave margins of stability for LQR feedback control around the nominal trajectory. Our approach also retains these advantages as a result of the polytope-based objective functions and, additionally, the new representation we propose allows determining the exact margins remaining before torque saturation occurs.

The more general idea of increasing robustness of an optimisation model is also important in fields outside of robotics. For instance, Ben-Tal and Nemirovsky [4] applied the more general idea of increasing robustness of an optimisation model to truss topology design (TTD). They cast TTD problems as semidefinite programs to minimize worst-case compliance of trusses under external loads, and used additional constraints to increase their robustness. For that, they considered not only primary loads (specified by the user), but also secondary loads from different directions and with reasonable magnitude.

3.3 PRELIMINARIES

3.3.1 *Polytopes and the Double Description Method*

A convex polytope [86] can be defined in one of two ways:

- Vertex representation ($\mathcal{V}$-rep): a finite set of points;
- Half-space representation ($\mathcal{H}$-rep): a bounded intersection of a finite set of half-spaces.

For some mathematical operations, one representation has some inherent advantages over the other. For example, the intersection of two or more polytopes is easier to perform in $\mathcal{H}$-rep than in $\mathcal{V}$-rep, and a Minkowski sum is easier to carry out in $\mathcal{V}$-rep than in $\mathcal{H}$-rep.

It may happen that a $\mathcal{V}$-rep is required when only an $\mathcal{H}$-rep is available, or vice versa—this is known as the representation conversion problem. It is possible to convert from one representation to the other using the double-description method [32]. Nonetheless, switching between representations can be computationally very expensive and should be avoided.

3.3.2 *Robot Model Formulation*

Consider a fully-actuated robot manipulator with n degrees of freedom, a fixed base, and with an end-effector operating in an m-dimensional task-space. Such a system can be parameterized with a generalized coordinates vector $\boldsymbol{q} \in \mathbb{R}^n$ and a generalized velocities vector $\boldsymbol{v} \in \mathbb{R}^n$. The dynamics of the system are given by the equations of motion:

$$\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{v}} + \boldsymbol{h}(\boldsymbol{q}, \boldsymbol{v}) = \boldsymbol{\tau} + \boldsymbol{J}_e^\top(\boldsymbol{q})\boldsymbol{f}_{\text{tip}}, \tag{4}$$

where $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$ is a symmetric positive-definite mass matrix, $\boldsymbol{h}(\boldsymbol{q}, \boldsymbol{v}) \in \mathbb{R}^n$ is the vector of Coriolis, centrifugal, and gravity terms, $\boldsymbol{\tau} \in \mathbb{R}^n$ is the vector of joint forces and torques, $\boldsymbol{J}_e \in \mathbb{R}^{m\times n}$ is the Jacobian matrix that maps joint velocities to the linear velocity of the end-effector, and $\boldsymbol{f}_{\text{tip}} \in \mathbb{R}^m$ is a force applied to the end-effector. The transpose of $\boldsymbol{J}_e$ maps a linear force applied at the end-effector to a vector of torques experienced at the joints of the mechanism—in the following referred to as $\boldsymbol{\tau}_{\boldsymbol{f}_{\text{tip}}}$. Conversely, we can determine an end-effector force generated from a vector of input torques with

$$\boldsymbol{f}_{\text{tip}} = \boldsymbol{J}_e^{-\top}\boldsymbol{\tau}_{\boldsymbol{f}_{\text{tip}}}. \tag{5}$$

In some cases, it may not be possible to invert $\boldsymbol{J}_e^\top$ because it is singular. Likewise, kinematically redundant systems have more joints than the dimension of their task space ($n > m$) and therefore $\boldsymbol{J}_e^\top$ is not square and cannot be inverted. However, for such cases, we can still solve equation (5) by using the Moore-Penrose pseudoinverse to invert $\boldsymbol{J}_e^\top$.

The mapping in equation (5) is instrumental for computing force polytopes, which we explain next.

3.3.3 *Joint Force Polytope and Force Polytope*

The *joint force polytope* [16] is an n-dimensional region bounded by the upper and lower actuation limits of the system. It is described by the $2n$ bounding inequalities

$$|\tau_i| \leq \tau_{i,\mathrm{lim}} \qquad i = 1, \cdots, n, \tag{6}$$

where $\tau_{i,\mathrm{lim}}$ is the bound on the i-th joint force.

The *force polytope* is the convex set of all the realizable forces by the end-effector for quasi-static scenarios, given the actuation limits of the system. A force polytope $P_{\boldsymbol{f}}$ results from transforming a joint force polytope $P_{\boldsymbol{\tau}}$ with $P_{\boldsymbol{f}} = \boldsymbol{J}_e^{-\top} P_{\boldsymbol{\tau}}$, analogous to how equation (5) converts a vector of joint-space forces and torques into a task-space force. Because of this nonlinear relationship, different robot configurations result in force polytopes with different shapes. Figure 6 illustrates this trait: two redundant configurations, $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$, reach the same end-effector target, but their respective force polytopes, P_1 and P_2, have distinct shapes.

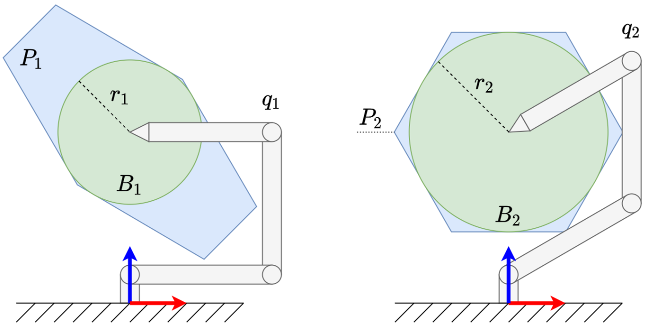


Figure 6: Two valid configurations for reaching the same end-effector target. The blue polygons P_1 and P_2 are the force polytopes of configurations $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$, respectively. The green circles B_1 and B_2 are the largest balls centred at the end-effector that can be inscribed inside those polytopes. The radius of B_1 and B_2 are denoted by r_1 and r_2, and here $r_2 > r_1$.

3.4 RESIDUAL FORCE POLYTOPE

In Section 3.3.3 we have reviewed what a force polytope is and how it results from the mapping of the actuation limits of a robot into the task-space. The force polytope is limited to quasi-static scenarios and, besides the kinematic configuration of the robot, it does not take into account any information about the task being performed.

We propose a new representation called the *residual force polytope*, which takes the dynamics of the robot into account, as well as the nominal forces and torques required by a task. We define the residual forces and torques of a robot state as the difference between the absolute actuation limits and a given vector of joint forces and torques. Residual forces and torques are important to deal with disturbances, as they represent the control authority left in a system after accounting for the task at hand. The residual force polytope is the result of transforming those residual forces and torques with $\boldsymbol{J}_e^{-\top}$, similarly to equation (5). In summary, the residual force polytope is a subset of its force polytope counterpart. It represents exclusively the forces that the robot is capable of resisting (as a secondary task) while tracking a nominal trajectory as its primary task.

Figure 7 shows the relationship between forces/torques in actuation-space and forces in task-space. For convenience of illustration, it displays a planar manipulator with three degrees of freedom in joint-space and two-dimensional task-space forces. The figure highlights how the residual force polytope P_3 is obtained for a given configuration $\boldsymbol{q}_1$.

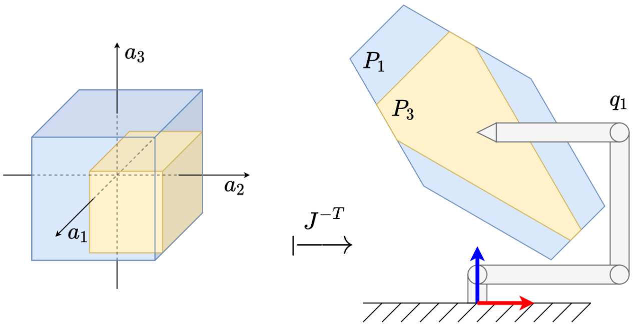


Figure 7: Equation (5) transforms actuation-space representations (on the left) into task-space representations (on the right). The blue polyhedron on the left is the *joint force polytope*, and by taking into account a given vector of torques it is reduced along some dimensions into the yellow polyhedron. The yellow polygon P_3 (on the right) is the *residual force polytope*.

3.5 MODELLING FORCE UNCERTAINTY

As we have seen so far, polytopes are useful to represent and model regions of interest in space. But in addition to this, we may want to extract a single metric that quantifies one of those regions. For example, given a configuration $\boldsymbol{q}_1$ and its corresponding force polytope P_1, we may want to know how robust that configuration is with respect to forces applied at the end-effector of the robot.

3.5.1 *Largest Ball Inscribed in a Polytope*

One way to tackle this problem is to consider the worst-case scenario, i.e., the situation with most uncertainty, where a force could originate from any given direction. In order to represent this uncertainty, we can use a ball to model a set of forces with any given direction and with a magnitude ranging from 0 N to the radius of the ball. Then, if we constrain the ball to be centred at the end-effector, and maximize the size of the ball without exceeding the boundaries of the force polytope, we obtain the set of all forces that the robot is able to deliver without saturating its torque limits. Consequently, the radius of this ball denotes the magnitude of the greatest force that the robot can counteract, and it can be used as a metric for isotropic robustness of a configuration.[1] For example, both $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ shown in Figure 6 solve the same reaching task, but $\boldsymbol{q}_2$ is more robust than $\boldsymbol{q}_1$ because $r_2 > r_1$.

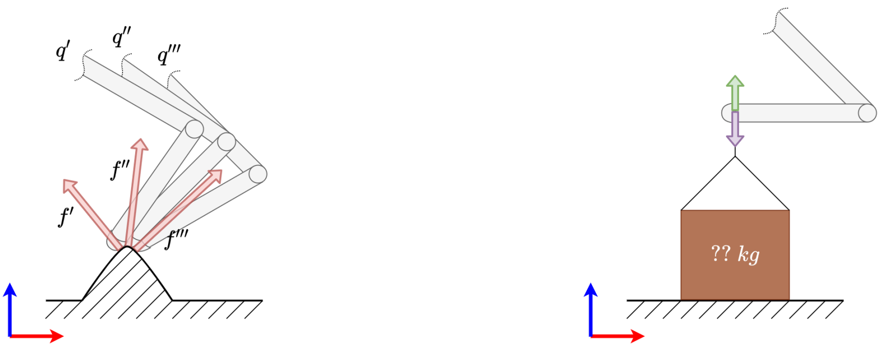
---

[1] In light of directional uncertainty, an isotropic robustness metric is more useful than other general quantities like the overall volume of a polytope.

The centre of the largest ball B inscribed in a bounded set of non-empty interior is known as the *Chebyshev centre* [7]. We can find the Chebyshev centre of a polytope P by solving a linear programming (LP) problem where the centre of the ball B and its radius r are the decision variables, and the goal is to maximize r subject to the constraint $B \subseteq P$. In our work, we are interested in a similar problem but where the centre of the ball lies at the origin of the end-effector frame. This is because we only care about the forces that can be applied specifically to the end-effector. Therefore, we formulate an LP problem which maximizes r subject to the constraint $B \subseteq P$, but where the only decision variable is the radius r (since the centre of B is known and given by the forward kinematics function of the robot's current configuration).

3.5.2 *Largest Intersection with a Polytope*

The previous subsection demonstrated how to calculate the robustness of a robot to completely unknown external disturbances. However, there are cases where the interaction between the robot and its environment is not fully uncertain. As an example, consider a task where the robot needs to open or close a door of unknown mass: the robot may not know *a priori* how much force is needed to solve the task, but the door can only open or close in a specific way—see Figure 8 for a further example.



(a) For legged robots locomoting on complex terrains, the direction of the terrain normals can change greatly with small variations in the contact location, leading to very different contact forces applied to the feet.

(b) In the context of a manipulation task, the direction of action/reaction forces for lifting a box attached to a rope is well-known *a priori*, and regardless of the weight of the box.

Figure 8: Illustration of two different levels of uncertainty concerning the direction of interaction forces for two real-world scenarios. On the left, factors such as controller tracking errors or noisy state estimation can ultimately lead to inaccurate foot placement, which in turn, and depending on the terrain, can induce forces applied in unexpected directions (high direction uncertainty). In contrast, on the right, the forces are expected to be close-to-vertical due to the nature of the task (low direction uncertainty).

The direction of the interaction can therefore be exploited to our advantage. We can use a cone to model the set of forces originating from some expected direction and applied to the end-effector: the cone axis is aligned with the expected direction, the cone apex is fixed at the end-effector frame, and the aperture of the cone represents the prediction uncertainty of the force direction. Then, if we intersect the cone with a force polytope, we obtain a subset of the forces in the cone which the robot can cancel out within its actuation limits. Consequently, the volume of the resulting intersection is proportional to how much the robot is capable of resisting forces modelled by the cone, and it can be used as a surrogate metric of robustness to expected forces. An example of modelling expected forces using this approach is illustrated in Figure 9, where the intersection of a cone C_1 with a residual force polytope P_3 results in the purple polygon P_4, i.e., $P_4 = P_3 \cap C_1$.

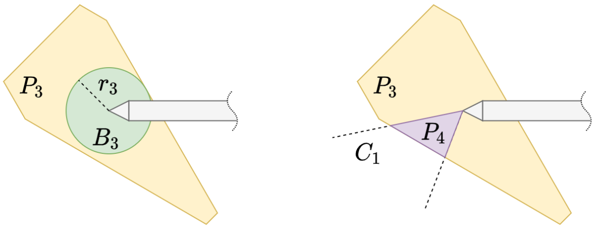


Figure 9: Two distinct models for representing force disturbances: the ball B_3 models unexpected forces, whereas the cone C_1 models the direction of an expected force. The aperture of the cone is proportional to the uncertainty of the force direction. The purple polygon P_4 results from the intersection of the *residual force polytope* with the cone, i.e., $P_4 = P_3 \cap C_1$.

In this section, we showed that redundant configurations result in different capabilities to counteract external forces applied to the end-effector. We proposed a representation for modelling those capabilities, and discussed two robustness metrics that can be extracted from it. In the next section, we will demonstrate how to formulate a trajectory optimisation problem with objective functions that employ those metrics in order to plan dynamic motions more robust to unexpected forces through exploitation of kinematic redundancy.

3.6 OPTIMISATION OF ROBUST TRAJECTORIES

Trajectory optimisation is a process that allows to compute control trajectories as functions of time that drive a system from an initial state towards a final state while satisfying a given set of constraints [5]. In robotics, the problem is a second-order dynamical system governed by the equations of motion (4).

Direct transcription [65] is a popular approach within trajectory optimisation and works by transcribing a continuous problem into a constrained nonlinear optimisation problem by means of explicit discretization of the state and control trajectories. The result of this transcription is the formulation of a large and sparse nonlinear problem which can be solved using a large-scale nonlinear programming solver [5].

We have chosen direct transcription to demonstrate how the residual force polytope can be used to plan robust and dynamic trajectories. Thanks to the discretization of states and controls, the configuration of the robot and the commanded torques are represented as decision variables for every discrete point of the trajectory. This means that all the "ingredients" required to compute the polytope representations (discussed in previous sections) are readily available as decision variables. Similarly, it also means that it is easy to define equality and inequality constraints using those decision variables, which general off-the-shelf nonlinear programming (NLP) solvers can then handle during problem resolution. In contrast, the most popular alternative, differential dynamic programming (DDP) [50], does not allow for easy definition of constraints (neither equalities nor inequalities). There are variations and extensions to classical DDP which attempt to mitigate this inconvenience (e.g., [33, 49, 67]), but this topic is still a subject of ongoing research and those variations are not yet mature enough.

In summary, we chose direct transcription because:

- Discretization of both states and controls is particularly convenient for computing polytope representations;
- Defining general state and path constraints using direct transcription is more straightforward than alternatives;
- Its simplicity of formulation and implementation.

3.6.1 *Problem Formulation*

We divide the trajectory into N equally spaced segments

$$t_I = t_1 < t_2 < \cdots < t_M = t_F, \tag{7}$$

where t_I and t_F are the start and final instants, respectively. Thus, the number of discretized *mesh points* is $M = N + 1$. Let $x_k \equiv x(t_k)$ and $u_k \equiv u(t_k)$ be the values of the state and control variables at the k-th mesh point, respectively. We treat $x_k \triangleq \{\boldsymbol{q}_k, \boldsymbol{v}_k\}$ and $u_k \triangleq \{\boldsymbol{\tau}_k\}$ as a set of nonlinear programming variables, and formulate the trajectory optimisation problem as:

$$\begin{aligned} \underset{\boldsymbol{\xi}}{\text{argmin}} \quad & \sum_{k=1}^{M} g(x_k, u_k) \\ \text{subject to} \quad & \dot{x} = f(x, u) \\ & x_k \in \mathcal{X} \\ & u_k \in \mathcal{U} \end{aligned} \tag{8}$$

where $\boldsymbol{\xi}$ is the vector of decision variables, $g(\cdot, \cdot)$ is a cost function, $\dot{x} = f(x, u)$ gives the nonlinear dynamics of the system, and $\mathcal{X}$ and $\mathcal{U}$ are sets of feasible states and control inputs enforced by a set of equality and inequality constraints. The vector of decision variables $\boldsymbol{\xi}$ results from aggregating the generalized coordinates, generalized velocities, and control inputs of every mesh point:

$$\boldsymbol{\xi} \triangleq \{\boldsymbol{q}_1, \boldsymbol{v}_1, \boldsymbol{\tau}_1, \cdots, \boldsymbol{q}_N, \boldsymbol{v}_N, \boldsymbol{\tau}_N, \boldsymbol{q}_M, \boldsymbol{v}_M\}.^2 \tag{9}$$

3.6.2 *Constraints*

We want to optimise trajectories that are consistent with the full dynamics of the robot, do not exceed the kinematic and actuation limits of the robot, and use the end-effector for a given task. We formulate all these requirements as equality and inequality constraints which the solver must respect.

---

[2] The control inputs at the final state $\boldsymbol{\tau}_M$ need not be discretized.

3.6.2.1 *End-effector task*

The exemplar task we use for this evaluation is to move the end-effector of a multi-degrees of freedom (DoF) robot arm from an initial point $\boldsymbol{p}_I$ to a final point $\boldsymbol{p}_F$:

$$f_{\text{FK}}(\boldsymbol{q}_1) = \boldsymbol{p}_I \quad \text{and} \quad f_{\text{FK}}(\boldsymbol{q}_M) = \boldsymbol{p}_F \tag{10}$$

where $f_{\text{FK}}(\cdot)$ is the forward kinematics function. In addition, the end-effector must always lie on a rectangular surface R positioned in its workspace:

$$f_{\text{dist}}(R, f_{\text{FK}}(\boldsymbol{q}_k)) = 0 \qquad \forall k = 1 : M \tag{11}$$

where $f_{\text{dist}}(\cdot)$ is the distance between a surface and a point. This task is analogous to drawing a line on a whiteboard using a marker attached to the end-effector, where the initial and final points are given and the path taken by the end-effector does not matter as long as it does not lift the tip of the marker off from the surface of the whiteboard.

3.6.2.2 *System dynamics*

We enforce the nonlinear dynamics of the system with a finite set of *defect constraints*. In summary, defect constraints are nonlinear equality constraints that ensure consistency between two consecutive mesh points.[3] They make sure that the robot state at the next time step (x_{k+1}) matches the propagation of the previous robot state (x_k) given its control inputs (u_k). In our formulation, we define these constraints as

$$x_{k+1} - \big(x_k + h \cdot f(x_k, u_k)\big) = 0. \tag{12}$$

For simplicity of exposition, we integrate the differential equations of the system dynamics using the explicit Euler method, where $h = (t_F - t_I)/N$ is the integration time step.

3.6.2.3 *Initial and final joint velocities*

We enforce the initial and final velocities of every joint to be zero with

$$\boldsymbol{v}_1 = \boldsymbol{v}_M = \boldsymbol{0}. \tag{13}$$

3.6.2.4 *Bounds of the decision variables*

We constrain the joint positions, velocities, and torques to be within their corresponding lower and upper bounds:

$$\boldsymbol{q}_{\text{lb}} \leq \boldsymbol{q}_k \leq \boldsymbol{q}_{\text{ub}} \qquad \forall k = 1 : M \tag{14}$$

$$\boldsymbol{v}_{\text{lb}} \leq \boldsymbol{v}_k \leq \boldsymbol{v}_{\text{ub}} \qquad \forall k = 1 : M \tag{15}$$

$$\boldsymbol{\tau}_{\text{lb}} \leq \boldsymbol{\tau}_k \leq \boldsymbol{\tau}_{\text{ub}} \qquad \forall k = 1 : M - 1 \tag{16}$$

---

[3] See Chapter 3.4 of Betts [5] for further detail regarding defect constraints.

3.6.3 *Objectives*

There are many objective functions which could be used to achieve different optimal results under the same problem constraints. We will now list some well-known objectives as well as our own. Later, in our experiments, we will compare the obtained trajectories against each other in terms of their robustness, torque expenditure, and computation time.

It is typical in optimal control to use energy as a cost, and this is usually formulated as a minimization of torques:

$$g_A : \quad \min_{\boldsymbol{\xi}} \quad \sum_{k=1}^{M} \boldsymbol{\tau}_k^\top \boldsymbol{\tau}_k \tag{17}$$

In order to avoid torque saturation, we can define a simple objective function to maximize residual actuator torques:

$$g_B : \quad \max_{\boldsymbol{\xi}} \quad \sum_{k=1}^{M} \left(\boldsymbol{\tau}_{\text{lim}} - \boldsymbol{\tau}\right)^\top \left(\boldsymbol{\tau}_{\text{lim}} - \boldsymbol{\tau}\right) \tag{18}$$

Yoshikawa [83] defined a quantitative measure of manipulability as $w = \sqrt{\det\left(\boldsymbol{J}_e \boldsymbol{J}_e^\top\right)}$. Later, Chiacchio *et al.* [16] proposed a more accurate definition by scaling the joint forces with $\boldsymbol{W} = \operatorname{diag}\left(1/\tau_{1,\text{lim}}, \cdots, 1/\tau_{n,\text{lim}}\right)$, which allowed to define a scaled Jacobian $\boldsymbol{J}_e'^\top = \boldsymbol{W}\boldsymbol{J}_e^\top$ and a more accurate measure of manipulability $w' = \sqrt{\det\left(\boldsymbol{J}_e' \boldsymbol{J}_e'^\top\right)}$. For our formulation, we can maximize the manipulability of every configuration in a discretized trajectory with the following objective:

$$g_C : \quad \max_{\boldsymbol{\xi}} \quad \sum_{k=1}^{M} w_t' \tag{19}$$

We can also define objectives with metrics extracted from polytopes. Let us denote the force polytope of a configuration as $P_k \equiv P(\boldsymbol{q}_k)$. Similarly to [55], and assuming static equilibrium, we can maximize the robustness to external forces from any given direction with:

$$g_D : \quad \max_{\boldsymbol{\xi}} \quad \sum_{k=1}^{M} B_r(P_k) \tag{20}$$

where $B_r(\cdot)$ denotes the radius of the largest ball centred at the end-effector and inscribed in the given polytope.

For the dynamic scenario, let us consider the residual force polytope as $P_k' \equiv P'(\boldsymbol{q}_k, \boldsymbol{\tau}_k)$, which is the novel representation we propose in this chapter. Analogous to (20), we can maximize the largest ball centred at the end-effector and inscribed in P_k' for every mesh point with:

$$g_E : \quad \max_{\boldsymbol{\xi}} \quad \sum_{k=1}^{M} B_r(P_k') \tag{21}$$

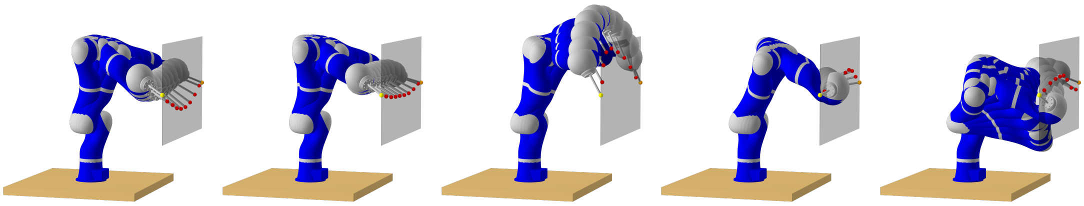
The last objective function we consider in this work is the intersection of the residual force polytope with a cone that models an expected force but with some level of uncertainty—we proposed this in Section 3.5.2. An objective function that maximizes the robustness in this scenario is:

$$g_F : \quad \max_{\xi} \quad \sum_{k=1}^{M} P_{\text{vol}}(P'_k \cap C_k) \tag{22}$$

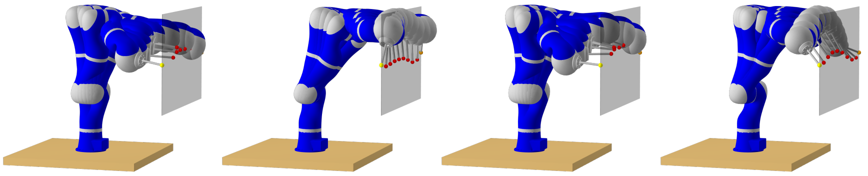
where $P_{\text{vol}}(\cdot)$ denotes the volume of a given polytope, and $C_k \equiv C(t_k)$ is a cone modelling a disturbance at instant t_k.

3.7 EXPERIMENTAL RESULTS

Using a *KUKA LWR* robot arm with 7-DoF, we solved the optimisation problem formulated in the previous section for each of the objective functions g_A–g_F without changing the problem constraints. We considered 1 s trajectories divided into 10 equally spaced segments (11 mesh points). Figure 10 shows the motion trace of the resulting trajectories. We used Julia [6] to implement our trajectory optimisation framework, and the library Knitro [11] to solve the nonlinear optimisation problems.



(a) Objective g_A (b) Objective g_B (c) Objective g_C (d) Objective g_D (e) Objective g_E



(f) Objective g_F, 0° (g) Objective g_F, 90° (h) Objective g_F, 180° (i) Objective g_F, 270°

Figure 10: Visualization of the trajectories obtained using the interior-point method without a payload. The configuration samples are equally spaced in time. The orange and yellow spheres denote the start and final targets for the end-effector. Trajectories generated with g_F depend on a specific direction; here we show four examples: 0°, 90°, 180°, and 270°. These angles correspond to being robust to forces originating from the front, left, back, and right sides of the robot.

In this section, we first compare the performance of two state-of-the-art optimisation methods for solving the problem we formulated. Afterwards, we compare the obtained trajectories against each other in terms of their robustness, simulated torque expenditure, and computation time. All evaluations were carried out in a single-threaded process on an Intel i7-6700K CPU at 4.0 GHz and with 32 GB 2133 MHz memory.

3.7.1 *Interior-Point vs. Active-Set Methods*

We want to compare the objective functions in our formulation using different classes of optimisation algorithms. There are two broad classes of methods for solving constrained nonlinear optimisation problems categorized based on how they handle constraints: *interior-point* (IP) methods incorporate the constraints into the objective (e.g., via a barrier function or an augmented Lagrangian), while *active-set* methods formulate a tractable model (e.g., by linearizing part of the constraints and penalizing them as well in the objective, as done with sequential quadratic programming (SQP) algorithms). For highly nonlinear problems, SQP methods are known to suffer from excessive pivoting, requiring expensive gradient evaluations of the constraints to update the active-set. As such, they are said to scale poorly to systems with many constraints. As a result, in robotics literature, IP methods are commonly used for direct transcription and collocation [64, 76], while some rely on SQP-based solvers [59]. However, few related work compare IP and SQP methods for solving equivalent problems, with the notable exception of [77]. In this subsection, we compare the performance of state-of-the-art, commercial large-scale sparse IP and SQP methods on the equivalent direct transcription problem (8) for all objective functions g_A–g_F. This emphasizes the differences between classical IP and SQP for direct transcription applications.

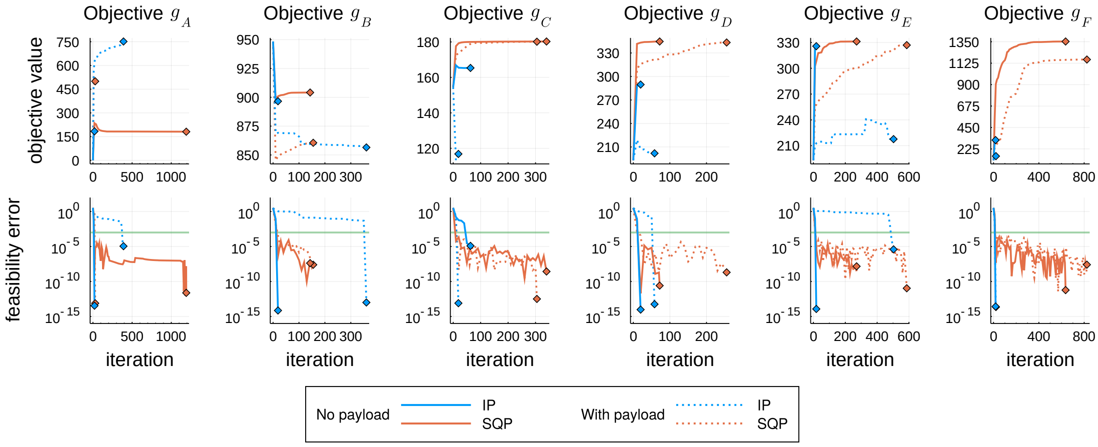


Figure 11: These plots show, for each function g_A–g_F, the evolution of the objective value and the feasibility error along the solver iterations. In the feasibility plots, the faint-green line at $y = 10^{-3}$ denotes the absolute tolerance under which a problem is considered feasible. We can see that all metrics were able to handle the payload. We can also see the back-and-forth progression of feasibility error for the SQP method due to excessive pivoting.

We used the SQP and IP method provided by [11]. For all comparisons and either method, we used automatic differentiation to obtain the Jacobian of the constraints, finite-differencing for the gradients of the objectives, and L-BFGS[4] for Hessian approximation (with 10 limited memory pairs). The results are presented in Figure 11, Table 8, and Table 9. In Figure 11, we can see that the interior-point method required very few iterations to converge when compared with the active-set method. As shown in Table 8, the total amount of time taken to find a locally optimal solution by the interior-point method was significantly less than the active-set method. In Table 9, we can see that the active-set method required significant more function and gradient evaluations than the interior-point method for the majority of the objective functions, which is expected and related to SQP's excessive pivoting (clearly observable in the feasibility error plots of SQP in Figure 11).

---

[4] L-BFGS stands for Limited-memory quasi-Newton BFGS.

Table 8: Convergence times (in seconds).

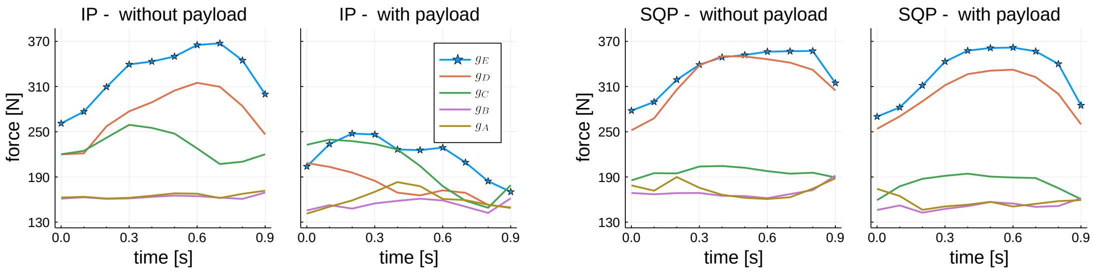
Objective function	No payload		With payload	
	Interior Point	Active Set	Interior Point	Active Set
g_A	0.07 ± 0.01	64.50 ± 0.85	0.82 ± 0.02	14.44 ± 0.32
g_B	0.08 ± 0.01	22.51 ± 0.31	0.24 ± 0.01	25.87 ± 0.47
g_C	0.11 ± 0.01	18.21 ± 0.21	0.11 ± 0.01	20.99 ± 0.54
g_D	150.46 ± 0.28	590.44 ± 5.27	1529.20 ± 2.97	2177.93 ± 40.91
g_E	261.66 ± 3.42	3698.82 ± 9.99	1120.90 ± 1.71	8195.97 ± 184.67
g_F	384.28 ± 0.61	14 354.95 ± 88.38	512.91 ± 1.10	19 902.78 ± 369.43

Table 9: Number of function evaluations and gradient evaluations of the problem constraints.

	Objective function	No payload		With payload	
		Interior Point	Active Set	Interior Point	Active Set
# function evals.	g_A	509	34629	9399	787
	g_B	484	4084	8686	4829
	g_C	1734	11066	542	9992
	g_D	571	2285	1594	8470
	g_E	952	15041	25217	32655
	g_F	953	34909	1104	45864
# gradient evals.	g_A	21	1192	391	30
	g_B	20	144	361	156
	g_C	64	341	20	306
	g_D	21	73	59	256
	g_E	19	272	504	587
	g_F	19	637	22	825

3.7.2 *Robustness to External Disturbances*

We want to evaluate each trajectory's ability to counteract external forces while executing its planned motion. As such, we first consider the torques required by the planned motion, and then calculate the set of all admissible forces from the remaining torques available. We define our evaluation metric as the magnitude of the maximum admissible force, considering all possible force directions. Therefore, each trajectory is evaluated as follows: for each point, (i) compute the residual force polytope, then (ii) find the largest ball centred at the end-effector inscribed in that polytope, and (iii) take the radius of the ball as the robustness metric. This is how we computed the forces shown in Figure 12.



(a) Interior-Point method.

(b) Sequential Quadratic Programming method.

Figure 12: These plots show the maximum admissible force magnitudes over time of trajectories computed using objective functions g_A–g_E. We can see that the objective function g_E, which uses the *residual force polytope*, resulted in greater admissible magnitudes than any other objective function.

3.7.2.1 *Overview of all objective functions*

Figure 12 shows the evaluation results considering all objectives g_A–g_E. In the plot, greater values correspond to greater robustness against unpredicted forces. The trajectory computed with the *residual force polytope* resulted in greater robustness than any other objective function considered.[5]

3.7.2.2 *Force Polytope vs. Residual Force Polytope*

Figure 13 shows the evaluation results for a scenario without a payload and for a scenario with a 2 kg cylindrical payload. The results in the plot correspond to trajectories obtained using g_D and g_E. We can see that the objective using the *residual force polytope* provided a significant improvement over the traditional force polytope; more specifically, for the 1-second-long trajectories we computed, an improvement of 53.2 ± 6.52 N without the payload, and 40.55 ± 21.37 N with the 2 kg payload.

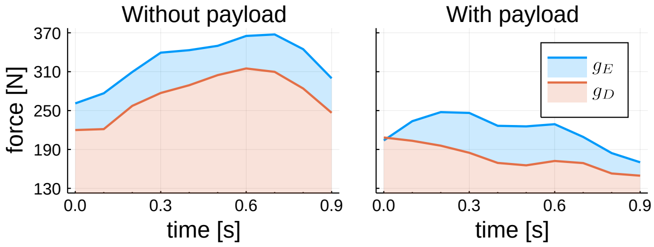


Figure 13: These plots show the magnitude of forces applied to the end-effector from any given direction and which the robot is able to cancel out given its actuation limits. Solid lines represent the maximum admissible magnitude over time, and shaded areas represent the magnitudes in between zero (no disturbance at all) and the maximum admissible magnitude. We can see that using the *residual force polytope* (shown in blue) provided a significant improvement over the classical force polytope (shown in red).

---

[5] The trajectory computed with g_C for the scenario with the payload and using the interior-point method resulted in an initial configuration with greater robustness than the other objective functions. However, we are interested in the robustness overall during the trajectory (area under the curve) and, for that, the objective function g_E defined as a function of the *residual force polytope* performed best.

3.7.3 *Unexpected Forces vs. Expected Forces*

In this experiment, we want to compare the torque required by the trajectories optimised using objectives g_E and g_F, which optimise a motion for resisting forces from any given direction and from a specific direction, respectively. More specifically, we want to determine how much torque the robot would need to complete a planned motion while, at the same time, resisting an external force applied to its end-effector. In order to do that, we apply an impulse to the robot and, for each point of the trajectory, we compute the extra torques required to oppose the external force with equation (5). The magnitude of the force applied to the robot at each instant is given by $\boldsymbol{f}(t) = \boldsymbol{f}_{\text{peak}} \cdot \exp(-(t-0.5)^2/0.02)$, where $\boldsymbol{f}_{\text{peak}}$ defines the magnitude at the peak of the impulse. The profile of this test force is shown in Figure 14.

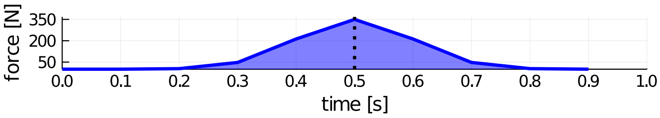


Figure 14: Profile of the test force applied to the end-effector. The impulse of this force is 87.73 N s and the peak magnitude is 350 N (at $t = 0.5$ s).

In order to compare optimal resistance to forces from any given direction (g_E) against optimal resistance to forces from a specific direction (g_F), we compute the torque required by the optimised trajectories for a test impulse that matches the direction estimation used during optimisation of the specialized trajectory with g_F. Afterwards, we invert the direction of the impulse and repeat the test to compute the required torques again. The results are shown in Figure 15.

3.7.4 *Summary of Computational Runtime*

Table 10 shows the average time required to evaluate each of the objective functions per solver iteration. The average was calculated from 10 samples. It is clear that the objectives defined as functions of polytopes take significantly longer to evaluate than the other objective functions tested.

Table 11 shows the average time required to compute: a force polytope, a residual force polytope, the largest ball inscribed in a polytope, the intersection of two polytopes, and the volume of a polytope. These methods are considerably expensive and are the reason why objectives g_D–g_F take so much time to evaluate.

3.8 DISCUSSION

Our initial hypothesis was that optimising trajectories with an objective defined as a function of admissible forces in task-space—after accounting for the torques required by the motion itself—would result in motion plans more robust to external disturbances. We defined an objective function based on the *residual force polytope* to optimise a trajectory robust to forces from any given direction, and compared it against other objective functions commonly used in trajectory optimisation, such as torque minimization, and manipulability maximization. The results we obtained support our initial hypothesis: as shown in Figure 12, for both the interior-point and active-set

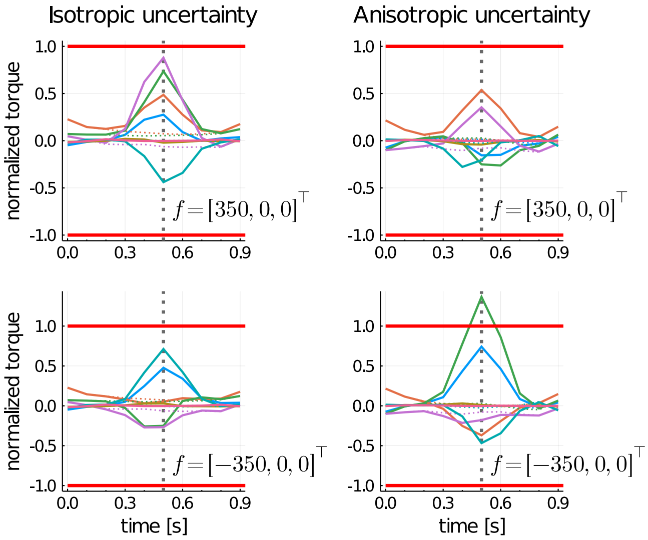


Figure 15: Joint torques required to complete the planned task and resist the disturbance. The torques have been normalized to $[-1, 1]$ according to actuation limits (solid red horizontal lines). The nominal torques are shown as dotted lines. The left and right columns correspond to the trajectories computed with g_E and g_F, respectively. On the left, we can see that the limits were not exceeded. On the top right, the trajectory resisted the impulse with less torque than g_E (this impulse was applied in the same direction as the estimation during optimisation). On the bottom right, when we applied the impulse in the opposite direction to what the specialized trajectory expected, the torques required exceeded the actuation limits of the robot.

methods tested, the objective function g_E we propose leads to optimal trajectories that are able to counteract forces from any direction with greater magnitude than any other objective function we explored. Moreover, the objective function g_F, which optimises trajectories specialized in specific directions, leads to even more robust motion plans than g_E if the disturbance is applied approximately in the same direction as the one considered for the specialization. However, specialized trajectories are less robust if the direction taken into account during optimisation does not match the actual disturbance accurately (case shown in the bottom right plot of Figure 15). Therefore, in terms of robustness, if a disturbance originating from a completely unexpected direction is not out of question, the objective considering any given direction (g_E) should be preferred over the optimisation of a specific direction (g_F). On the other hand, any accurate bias about disturbance directions that may arise out of known environmental constraints (e.g., axis of fixation of articulated objects being manipulated) should be incorporated into g_E to allow more dynamic range of motion.

Despite the promising results in terms of robustness, the objective functions we proposed are very demanding computationally: even though we used a coarse problem discretization, all the objectives defined as functions of polytopes took at least 3 orders-of-magnitude longer to converge than the simpler objective functions g_A–g_C. This significant difference is due to the double description method required to convert across polytope representations as discussed in Section 3.3.1, and due to the other mathematical operations involving polytopes (benchmarked in Table 11). Nonetheless, the objectives g_E and g_F utilizing the residual force polytope representation did not

Table 10: Time required to evaluate each objective function once.

Objective	Average time (ms)
g_A	0.007 ± 0.029
g_B	0.012 ± 0.002
g_C	0.023 ± 0.096
g_D	85.004 ± 10.342
g_E	72.565 ± 7.209
g_F	102.036 ± 9.069

Table 11: Time benchmark of computational geometry methods.

Operation	Time (µs)
Force polytope	22 ± 124
Residual force polytope	24 ± 146
Largest inscribed ball	7354 ± 2405
Polytope intersection	7081 ± 2049
Polytope volume	6312 ± 1923

incur significant convergence time differences compared to objective g_D using the traditional force polytope.

3.8.1 *On the Scalability of Our Metric*

We did not carry out experiments using different robot arms. While the absolute values shown in our results will vary across different manipulators, we speculate that the relative differences observed should generalize to manipulators of different sizes and with more or less joints.

Regarding the scalability of our approach to floating-base robots—such as quadrupeds or bipeds—there is a distinction to be made: whether the metric is to be used as an evaluation metric for existing trajectories, or if it is to be used as an objective function in a trajectory optimisation setting.

Robustness as an evaluation metric. Given an existing dynamic trajectory, computing the residual force polytope for each point in time is straightforward. A possible application for this is to evaluate the robustness of different trajectories, and to compare them against each other. In fact, this is exactly what we did in Section 3.7.2 in order to evaluate the robustness of the motions obtained from the optimisation of different objective functions. For this use-case, our metric should be scalable to different platforms, but it will become more computationally demanding—and therefore slower—as the degrees of freedom of the system increase: the number of vertices of the polytope grows with the number of degrees of freedom of the system, and the complexity of converting representations (from $\mathcal{V}$-rep to $\mathcal{H}$-rep, or vice-versa) grows with the number of vertices.

Robustness as an objective function. In the context of trajectory optimisation, using our metric as an objective function for floating-base systems with many degrees of freedom is not straightforward and presents significant scalability issues. The reason for this is related (but not limited) to the point mentioned above: computing the residual force polytope becomes more demanding and slower as the number of degrees of freedom of the robot increases. For purposes of evaluating a trajectory, the polytope only needs to be computed once for each mesh point. On the other hand, in trajectory optimisation, the solver takes several iterations (in our case, hundreds of iterations) while converging to a locally optimal solution, and for each of those iterations it may need to perform more than one function or gradient evaluation, which requires computing the residual force polytope again and again. As a consequence, optimising trajectories for high-DoF robots in a reasonable amount of time is not possible, and could take multiple days to complete. We would like to emphasize that this is not a limitation of the residual force polytope we propose, but a limitation of using any polytope. Since this is a well-known issue, other authors have tried to use approximations to work around it. Next, we list a few options for mitigating this drawback.

3.8.2 *Mitigating the Computational Cost*

Less frequent polytope evaluations. One way to decrease computational cost is by evaluating the polytope less frequently. For online planning and control, this would mean computing the polytope at regular time intervals, using it to adapt the motion of the robot every now and then. This approach was used by Orsolino *et al.* [55] for optimising the centre-of-mass position of a quadruped's static crawl gait. In that work, the feasibility polytope was calculated once at the beginning of the optimisation and used as a constant approximation thereafter.

Approximation of polytope geometries. Another way to decrease computational cost is to approximate polytope geometry with morphing techniques or with surrogate models. Bratta *et al.* [8] used polytope morphing for computing the polytope of each leg of a quadruped robot. The authors computed an exact polytope representation for two key configurations, and then approximated the polytope for intermediate configurations by interpolating its shape. Another option (not yet explored) is to use a surrogate model. Surrogate models approximately mimic the behaviour of functions that are computationally expensive to evaluate. They can be constructed offline by exploring the states of the system, and then evaluated online quickly.

Specialized solvers. In this work, we used an off-the-shelf optimisation library, Knitro [11], which implements state-of-the-art algorithms for solving numerical problems. Instead of using generic solvers, one could attempt to take advantage of problem-specific features to customize the solver's internal implementation (e.g., with heuristics, linearized relaxations, cutting planes), trading off generality for performance. However, developing such custom solvers is very time-consuming and requires expert knowledge in numerical optimisation.

3.9 CONCLUSION

In this chapter, we proposed an exact representation for task-space forces which the robot can counteract: the *residual force polytope*. The representation takes into account the whole-body dynamics of the robot, and considers only the torques remaining after accounting for the controls of a nominal trajectory (or the controls of a trajectory being optimised). Our proposition contrasts with approximate representations (e.g., in ellipsoidal forms) from previous related work, which do not account for the nominal control trajectory and therefore overestimate the true capabilities of a system. We defined two functions based on the residual force polytope, for two different levels of disturbance uncertainty, and used them as objectives in trajectory optimisation to plan motions more robust to external disturbances.

Despite the qualitative benefits of the trajectories obtained using our method, its computational cost does not allow deploying it as a real-time planning method. However, our approach can be used for offline motion planning (where time consumption is not as critical), as well is in other areas besides trajectory optimisation, such as system analysis and co-design.

Another drawback of our current approach is that it does not scale to robots with a floating base (such as quadrupeds or humanoids) due to the immense amount of vertices in exact polytopes that high-dimensional systems would generate. However, instead of using explicit polytope descriptions, we can approximate them[6] [85] in exchange for scalability and decreased computational cost. For that, choosing the right level of approximation becomes an important decision [78], and the trade-off between speed and accuracy has to be considered carefully.

In the next chapter, we are going to tackle that very problem. We will work out how to compute our robustness metric without exact polytope descriptions, and we will reformulate the trajectory optimisation problem in a way that enables our framework to scale to complex underactuated robots.

---

[6] Not to be confused with approximations to dynamic quantities of the controlled system, which our proposed representation is trying to avoid.

4

ROBUST TRAJECTORIES FOR FLOATING-BASE ROBOTS

This chapter focuses on robustness to external forces and uncertain payloads. We present a novel formulation to optimise the robustness of dynamic trajectories. A straightforward transcription of this formulation into a nonlinear programming problem is not tractable for state-of-the-art solvers, but it is possible to overcome this complication by exploiting the structure induced by the kinematics of the robot. The non-trivial transcription that we propose allows trajectory optimisation frameworks to converge to highly-robust and dynamic solutions. We demonstrate the results of our approach using a quadruped robot equipped with a manipulator.

4.1 INTRODUCTION

When an external force is applied to a legged robot with a manipulator, it may cause the robot to slip, or to fail to track a path with its end-effector. Similarly, the performance degrades when the robot poorly estimates how slippery the ground is or how heavy is its payload. In either case the motion fails because completing the task while compensating for the external force requires the robot to either command more torque to its actuators than they are capable of delivering, to produce unrealistic contact forces, or both. These limitations impose constraints that the robot motion has to satisfy. Therefore, one way to look at robustness is to define it as some metric of distance to these constraints, for instance, as the force the robot can compensate for before violating the motion constraints. This kind of robustness could be optimised over by the robot controller, however, considering robustness during motion planning would allow us to avoid difficult-to-execute motions altogether.

We tackle the problem of robustness against external perturbations and unmodelled payloads for complex legged robots with manipulation capabilities. We focus on increasing robustness at the planning stage to provide any tracking controller, including robust control schemes, with greater margins of control authority. Previous work [78] used the smallest unrejectable force (SUF) applied at some link of the robot as a robustness metric for improving single configurations via convex conic optimisation. In this work, we propose a novel formulation to make the computation more tractable and versatile, allowing us to consider the optimisation of entire trajectories with nonlinear dynamics. Our new computational framework enables us to combine trajectory optimisation (TO) with the SUF metric to produce highly robust and dynamic trajectories.

4.2 RELATED WORK

Bellicoso et al. [3] presented a motion planning and control framework for a platform similar to ours (see Figure 16). The authors demonstrated successful execution of tasks such as opening a door and carrying a box alongside a human. The authors addressed robustness to external disturbances with an inverse dynamics-based whole-

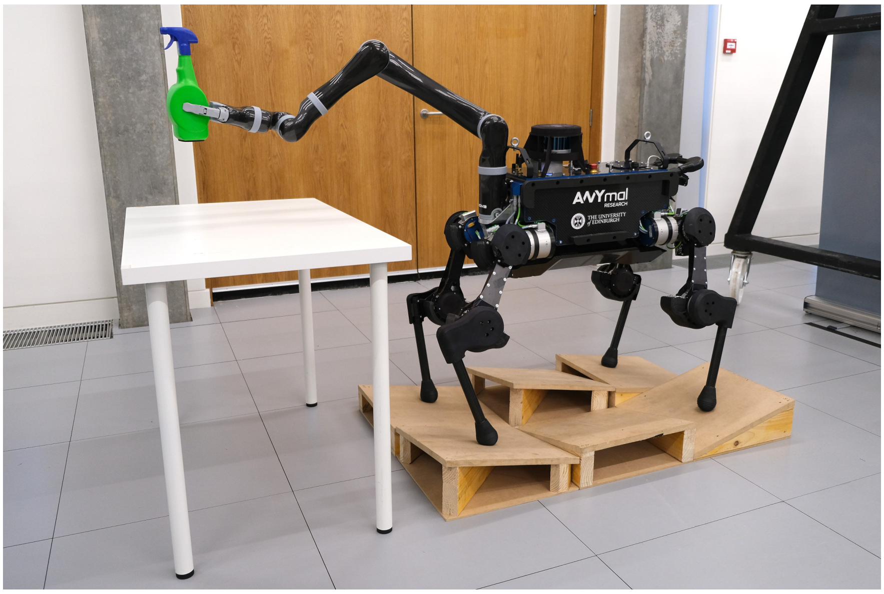


Figure 16: A legged loco-manipulation system: ANYmal [37] is a fully torque-controlled quadruped robot. We equipped it with a Kinova Jaco [12] robot arm. An accompanying video is available at `https://youtu.be/vDesP7IpThw`.

body controller and by re-planning locomotion continuously in a receding-horizon fashion. However, contrary to our approach, they did not take into account robustness explicitly at the planning-level.

Del Prete and Mansard [19] proposed a solution to improve the robustness to joint-torque tracking errors at the control stage. The authors modelled deterministic and stochastic uncertainties in joint torques within their control framework optimisation. Their idea is similar to what we present in this chapter, but we maximize the upper-bound force magnitude the system can withstand from any possible direction—and we do this during planning.

The authors of Xin et al. [79] included external forces estimation directly into their hierarchical controller. Their objective was to minimize actuator torques while enforcing constraints on the contact forces. However, contrary to our work, they did not enforce actuator limitations explicitly.

Modelling the capabilities of the system explicitly using polytopes has recently become more popular than using simplified metrics for robustness. In Caron, Pham and Nakamura [13], the authors derived the equations of a so-called gravito-inertial wrench cone (GIWC). It is a feasible region used as a general stability criterion. This representation is very efficient for testing robust static equilibrium of a legged robot, but it fails to take into account any actuation limits. Orsolino et al. [55] proposed to incorporate the properties of [13] with system torque limits. They use the resulting polytopes to optimise the centre of mass (CoM) trajectory in the xy-plane for the base-transfer motion of a quadruped. Despite the reduced size of this problem, the technique used to compute polytopes was prohibitively expensive, and as a workaround they computed polytopes once at the beginning and used them as an approximation for the remaining motion.

We have followed this line of research in our previous work [30] and we proposed a force polytope representation considering system dynamics: the *residual force polytope*.

The polytope is computed from the forces and torques remaining after accounting for Coriolis, centrifugal, and gravity terms, as well as nominal motion feed-forward torques.

All the polytope calculations proposed in the literature [30, 55, 83] require a significant amount of computation time. In general, deriving an explicit description of a projected polytope is NP-hard [70]. As a result, prior work using polytopes in trajectory optimisation, e.g., Orsolino et al. [55], resorted to the approximation of fixing the polytope for an entire trajectory.

Zhen and Hertog [85] recently formulated a computationally tractable approach for finding maximally sized convex bodies inscribed in a projected polytope. Their scheme does not require an explicit description of the projection and works by combining Fourier-Motzkin elimination with techniques from adjustable robust optimisation. The scheme was adapted for robustness computations in robotics in [78], where the SUF were estimated for static configurations. However, despite an improvement over exact computation, due to the computational complexity of their formulation it was not previously possible to consider trajectory optimisation of full system dynamics and maximization of robustness based on dynamic polytopes at the same time. We further adapt the technique of Zhen and Hertog [85] to reformulate the problem of computing the SUF. The resulting reformulation allows considering trajectory optimisation and robustness maximization in a bilevel optimisation setting.

Bilevel optimisations are mathematical programs that include the solution to other programs in their constraints or objectives. They are common in robustness settings, and have been used for robust control of robots. [46] optimised trajectories with full dynamics for robustness as a bilevel problem; it is particularly related to our work, but with some key differences: they considered robustness to noise and dealt with a fixed base manipulator—both differences allowed for simplifications in their optimisation problem.

We present a TO framework capable of planning robust and dynamic manipulation tasks for legged robots, such as the one shown in Figure 16. Our main contributions are:

1. Proposal of a novel solution to a bilevel optimisation problem that marries dynamic trajectory optimisation with maximization of robustness against disturbances.
2. Explanation of the non-trivial transcription and reformulation required to make this problem tractable for a nonlinear programming (NLP) solver.
3. Comparison of our method's results against a traditional optimisation objective across different scenarios.
4. Validation of the planned motions using both full-physics simulation and real-life hardware experiments.

4.3 TRAJECTORY OPTIMISATION

Trajectory Optimisation (TO) is a well-known and powerful framework for planning locally-optimal trajectories of dynamic systems such as legged robots subject to constraints. TO falls under the broader category of optimal control problems. In general, TO aims to design a finite-time control trajectory as a function of time, $u(t)$, which drives the system from an initial state $x(t_I)$ towards a final state $x(t_F)$, and given the system dynamics $\dot{x} = f(x, u)$ which must be satisfied over the entire interval $t_I \leq t \leq t_F$. Optimal control problems can be solved using dynamic programming or by means of transcription (see Betts [5]).

In this work, we employ a technique called direct transcription because it readily handles strict constraints on states and controls. Such constraints take a key role in computing the SUF. The main alternative, differential dynamic programming (DDP), offers faster computation and provides a linear controller next to the optimised trajectory. However, handling constraints with DDP is a challenging subject of research [33, 49]. This currently makes DDP less applicable to our case. A second alternative, shooting methods, have been reported to result in slower computations and higher susceptibility to local optima [5]. Using direct transcription, we formulate the continuous optimisation problem by explicitly discretizing the system state and control trajectories. This method results in the formulation of a large and sparse NLP problem [5]. The resulting constrained nonlinear optimisation problem can then be solved using a sparse, large-scale nonlinear programming solver such as Knitro [11].

4.4 MODEL FORMULATION

The model of a legged robot can be formulated as a free-floating base B to which limbs are attached. For the specific case of the robot shown in Figure 16, the kinematic tree stemming from the base branches into four legs and one robotic manipulator with six degrees of freedom (DoF). The motion of the system can be described with respect to (w.r.t.) a fixed inertial frame I. Let us express the position of the base w.r.t. the inertial frame, expressed in the inertial frame, as ${}_I\boldsymbol{r}_{IB} \in \mathbb{R}^3$. Let $\boldsymbol{q}_{IB} \in \mathbb{H}$ be a Hamiltonian unit quaternion defining the orientation of the free-floating base w.r.t. the inertial frame, and let $\boldsymbol{\psi}_{IB} \in \overline{\mathbb{R}}^3$ be the modified rodrigues parameters (MRP) [34, 69] of the unit quaternion $\boldsymbol{q}_{IB}$.[1] We use $\boldsymbol{\psi}_{IB}$ to parameterize the orientation of the free-floating base.[2] The joint angles describing the configuration of the 6-DoF arm and the four 3-DoF legs are stacked in a vector $\boldsymbol{q}_j \in \mathbb{R}^{n_j}$, where $n_j = 18$. The generalized coordinates vector $\boldsymbol{q}$ and the generalized velocities vector $\boldsymbol{v}$ of this floating-base system may therefore be written as

$$\boldsymbol{q} = \begin{bmatrix} {}_I\boldsymbol{r}_{IB} \\ \boldsymbol{\psi}_{IB} \\ \boldsymbol{q}_j \end{bmatrix} \in \mathbb{R}^3 \times \overline{\mathbb{R}}^3 \times \mathbb{R}^{n_j}, \quad \boldsymbol{v} = \begin{bmatrix} \boldsymbol{\nu}_B \\ \dot{\boldsymbol{q}}_j \end{bmatrix} \in \mathbb{R}^{n_v}, \tag{23}$$

where $n_v = 6 + n_j$ and the *twist* $\boldsymbol{\nu}_B = [{}_I\boldsymbol{v}_B \quad {}_B\boldsymbol{\omega}_{IB}] \in \mathbb{R}^6$ encodes the linear and angular velocities of the base B w.r.t. the inertial frame expressed in the I and B

---

[1] $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ is the *affinely extended set of real numbers*. We use the same notation as Terzakis, Lourakis and Ait-Boudaoud [69].

[2] The MRP encode a 3D rotation with the stereographic projection of a Hamiltonian unit quaternion. The derivatives of the rotation matrix w.r.t. the MRP parameters are rational functions, making this representation a particularly good choice for purposes of differentiation and optimisation.

frames, respectively. The equations of motion of a floating base system that interacts with the environment are written as

$$\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{v}} + \boldsymbol{h}(\boldsymbol{q},\boldsymbol{v}) = \boldsymbol{S}^\top\boldsymbol{\tau} + \boldsymbol{J}_s^\top(\boldsymbol{q})\boldsymbol{\lambda} + \boldsymbol{J}_e^\top(\boldsymbol{q})\boldsymbol{f}, \tag{24}$$

where $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n_v \times n_v}$ is the mass matrix and $\boldsymbol{h}(\boldsymbol{q},\boldsymbol{v}) \in \mathbb{R}^{n_v}$ is the vector of Coriolis, centrifugal, and gravity terms. The selection matrix $\boldsymbol{S} = [\boldsymbol{0}_{n_\tau \times (n_v - n_\tau)} \quad \mathbb{I}_{n_\tau \times n_\tau}]$ selects which DoF are actuated. Here, $n_\tau = n_j$ as all limb joints are actuated. The vector of ground-feet contact forces and contact torques $\boldsymbol{\lambda}$ is mapped to joint-space torques through the support Jacobian $J_s \in \mathbb{R}^{n_s \times n_v}$, which is obtained by stacking the Jacobians which relate generalized velocities to limb end-effector motion as $\boldsymbol{J}_s = [\boldsymbol{J}_{C_1}^\top \quad \cdots \quad \boldsymbol{J}_{C_{n_c}}^\top]^\top$, with n_c being the number of limbs in contact and n_s the total dimensionality of all contact wrenches. We assume ANYmal has point-feet, and thus we only model linear contact forces at the feet. Finally, $\boldsymbol{f}$ represents any external force applied to the end-effector. This force may be the result of a push or some unpredicted disturbance. In a nominal scenario, this force is zero, i.e., $\boldsymbol{f} = \boldsymbol{0}$. The Jacobian $J_e \in \mathbb{R}^{3 \times n_v}$ is used to map a linear force $\boldsymbol{f}$ applied at the end-effector to joint-space torques.

4.5 PROBLEM FORMULATION

We transcribe the continuous optimisation problem by explicitly discretizing the system state and the control trajectory using a *direct transcription* technique. We divide the trajectory into N equally spaced segments or intervals

$$t_I = t_1 < t_2 < \cdots < t_M = t_F, \tag{25}$$

where the points are referred to as *mesh points*.[3] The number of mesh points is given by $M = N + 1$. Henceforth, we use $x_k \equiv x(t_k)$ and $u_k \equiv u(t_k)$ to indicate the value of the state and control variables, respectively, at mesh point k. We treat the values of x_k and u_k as a set of NLP variables, and we finally formulate the general TO problem as:

$$\begin{aligned} \underset{\boldsymbol{\xi}}{\text{argmin}} \quad & g_M(x_M) + \sum_{k=1}^{M-1} g(x_k, u_k) \\ \text{subject to} \quad & x_{k+1} = x_k + h\, f(x_k, u_k) \\ & x_k \in \mathcal{X} \\ & u_k \in \mathcal{U} \end{aligned} \tag{26}$$

where $g(\cdot,\cdot)$ and $g_M(\cdot)$ form an optional cost function, $h = (t_F - t_I)/N$ is a fixed *integration step size*, and $\mathcal{X}$ and $\mathcal{U}$ are the sets of feasible states and inputs, respectively. We use the explicit Euler method to integrate the differential equations of the system dynamics, but other K-stage Runge-Kutta schemes could be used, e.g., the Trapezoidal method (implicit, $K = 2$) or the Hermite-Simpson method (implicit, $K = 3$).

4.5.1 *Parameterization*

Similarly to Posa, Cantu and Tedrake [59], we directly optimise over the space of feasible states, control inputs, and constraint forces, i.e., for each discretized mesh point k, the vectors of generalized coordinates $\boldsymbol{q}_k$, generalized velocities $\boldsymbol{v}_k$, control inputs $\boldsymbol{\tau}_k$, and contact forces $\boldsymbol{\lambda}_k$ form the vector of decision variables $\boldsymbol{\xi}_k$. The entire vector of NLP decision variables is:

---

[3] Some authors also refer to these mesh points as *nodes*, *knots*, *way points*, or *grid points*.

$$\boldsymbol{\xi} \triangleq \{\boldsymbol{q}_1, \boldsymbol{v}_1, \boldsymbol{\tau}_1, \boldsymbol{\lambda}_1, \cdots, \boldsymbol{q}_N, \boldsymbol{v}_N, \boldsymbol{\tau}_N, \boldsymbol{\lambda}_N, \boldsymbol{q}_M, \boldsymbol{v}_M\}.^4 \tag{27}$$

Methods that treat contact forces as optimisation variables are referred to as planning "through contact". This approach increases the number of decision variables, but the problem becomes better conditioned [45].

4.5.2 *Objectives*

We consider three different optimisation objectives $\mathcal{G}_1$–$\mathcal{G}_3$. The first objective corresponds to the *feasibility problem*, i.e., a problem with constraints but without any cost to minimize.

The second objective $\mathcal{G}_2$ achieves the minimization of actuator torques and is defined as

$$\mathcal{G}_2 : \quad \underset{\boldsymbol{\xi}}{\operatorname{argmin}} \quad \sum_{k=1}^{M-1} \boldsymbol{\tau}_k^\top \boldsymbol{\tau}_k. \tag{28}$$

Finally, objective $\mathcal{G}_3$ corresponds to the maximization of the SUF at the end-effector. $\mathcal{G}_3$ involves a problem reformulation which is explained in Section 4.6.1—the main contribution presented in this chapter.

4.5.3 *Constraints*

We now analyse the constraints formulated in the NLP in detail. Table 12 shows a summary of these constraints.

4.5.3.1 *Bounds on decision variables*

We constrain the joint positions, velocities, and torques to be within their respective lower and upper bounds with (29)–(31).

$$\boldsymbol{q}_L \leq \boldsymbol{q}_k \leq \boldsymbol{q}_U \qquad \forall k = 1 : M \tag{29}$$

$$\boldsymbol{v}_L \leq \boldsymbol{v}_k \leq \boldsymbol{v}_U \qquad \forall k = 2 : M-1 \tag{30}$$

$$\boldsymbol{\tau}_L \leq \boldsymbol{\tau}_k \leq \boldsymbol{\tau}_U \qquad \forall k = 1 : M-1 \tag{31}$$

We further fix the initial and final velocities to zero:

$$\boldsymbol{v}_1 = \boldsymbol{v}_M = \boldsymbol{0}. \tag{32}$$

---

[4] Notice that $\boldsymbol{\tau}_M$ and $\boldsymbol{\lambda}_M$ (i.e., the control and contact forces at the final state) are not required, and thus not part of $\boldsymbol{\xi}$.

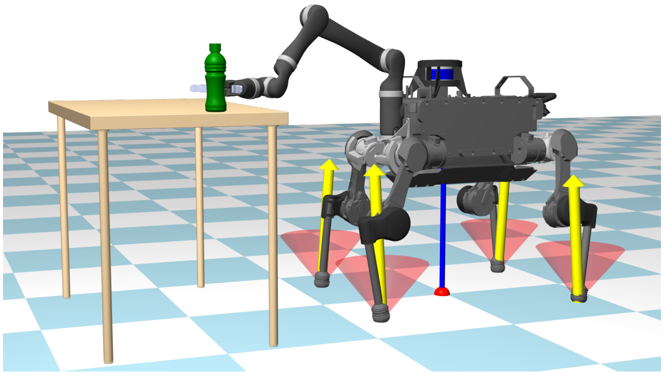


Figure 17: The figure shows the robot at the beginning of a pick-and-place task. The ground-feet contact forces are shown in yellow and the friction cones are shown in red. The blue line represents the CoM projection.

4.5.3.2 *Friction cones*

Similarly to Caron, Pham and Nakamura [13], we model friction at the contact points using an *inner linear approximation* with a four-sided friction pyramid. Consider the set of points $\{C_i\}$ where the robot is in contact with its environment. Let $\boldsymbol{n}_i$ and μ_i be the unit normal and the friction coefficient of the support region at each contact, respectively. A point contact remains fixed as long as its contact force $\boldsymbol{f}_i^c$ lies inside the linearized friction cone directed by $\boldsymbol{n}_i$:

$$|\boldsymbol{f}_i^c \cdot \boldsymbol{t}_i| \leq (\mu_i/\sqrt{2})(\boldsymbol{f}_i^c \cdot \boldsymbol{n}_i), \tag{33}$$

$$|\boldsymbol{f}_i^c \cdot \boldsymbol{b}_i| \leq (\mu_i/\sqrt{2})(\boldsymbol{f}_i^c \cdot \boldsymbol{n}_i), \tag{34}$$

$$\boldsymbol{f}_i^c \cdot \boldsymbol{n}_i > 0, \tag{35}$$

where $(\boldsymbol{t}_i, \boldsymbol{b}_i)$ form the basis of the tangential contact plane, such that $(\boldsymbol{t}_i, \boldsymbol{b}_i, \boldsymbol{n}_i)$ is a direct frame.

4.5.3.3 *System dynamics*

Using explicit Euler integration, we enforce the nonlinear system dynamics (f) with a finite set of *defect* (or *gap*) constraints in our NLP formulation:

$$\begin{bmatrix} \boldsymbol{q}_{k+1} \\ \boldsymbol{v}_{k+1} \end{bmatrix} - \begin{bmatrix} \boldsymbol{q}_k \\ \boldsymbol{v}_k \end{bmatrix} - h\, f\left(\begin{bmatrix} \boldsymbol{q}_k \\ \boldsymbol{v}_k \end{bmatrix}, \begin{bmatrix} \boldsymbol{\tau}_k \\ \boldsymbol{\lambda}_k \end{bmatrix} \right) = \boldsymbol{0}. \tag{36}$$

4.5.3.4 *Stationary feet*

Let the *forward kinematics* function for a foot-point contact i be given by $f^{\text{fk}}(\boldsymbol{q}, i)$.

$$f^{\text{fk}}(\boldsymbol{q}_k, i) = \boldsymbol{p}_i \qquad \forall i = 1:4,\ k = 1:M \tag{37}$$

4.5.3.5 *Gripper task*

The gripper is constrained at the initial and final instants of the trajectory ($k = 1$ and $k = M$). For both of these mesh points, there exist five constraints: three to constrain the placement of the end-effector, and two for constraining the *pitch* and *roll* describing the orientation of the end-effector. This enforces a specific location for the pick and placing of a bottle (e.g., see Figure 17), as well as the correct orientation of the fingers to embrace it, while leaving the grasp *yaw* as a degree of freedom to the solver.

Table 12: Summary of the formulated NLP constraints.

Constraint	Structure	Relation
Bounds on ξ	Linear	Mixed
Friction Cones	Linear	Inequality
System Dynamics	Nonlinear	Equality
Stationary Feet	Nonlinear	Equality
Gripper Task	Nonlinear	Equality

Building on the formulation above, we now want to implement additional terms to model the external disturbances. We are interested in maximizing the forces applied at the end-effector that the robot can compensate for while still satisfying all the NLP constraints from Table 12.

4.6 ROBUSTNESS TO DISTURBANCES

External forces applied to the robot can cause the robot to slip, lose contact between a foot and the environment, or to fail to track the desired end-effector path. In each of these cases, the motion fails because the external force causes a violation of one of the motion constrains. We therefore define robustness as some metric of distance to the constraints. More specifically, we consider the friction cone constraints on the contact forces and the actuator torque bounds limiting the control commands that can be used to compensate for external forces. As pointed out by [55], when transformed into a common reference frame, these constraints form a convex polytope bounding the volume of all admissible external wrenches applied to the robot. In [30] we proposed a geometric method to inscribe a ball into the polytope and use its radius as a metric of robustness. This approximation is especially useful since the radius of the maximum volume inscribed ball gives a bound on the magnitude of forces from any direction that the system can compensate for without violating the constraints.

4.6.1 *Maximum-Volume Ball Inscribed in a Polytopic Projection*

In order to reject a disturbance force, additional motor torques and ground reaction forces are needed. Our robustness metric, the SUF, is the smallest force for which no reaction forces/torques exist that also satisfy friction-cone constraints and motor limitations. In previous work [30], the SUF was computed via a linear programming (LP) problem. The trajectory optimisation would have this LP problem inside its objective, but that is not desirable if the underlying solver for the LP is not differentiable itself. Hence, we propose a new way to compute the SUF. The key idea in this robustness analysis is to approximate the relationship between the disturbance force and reaction forces/torques as affine. Adapting the results from [85], we find a practically efficient way to simultaneously optimise the robustness metric and the affine relationship prescribing it. These results go beyond earlier adaptations in robotics by [78] because those, like our work relying on LP, were not suitable for use in a trajectory optimisation setting.[5]

Let us define the *extended* torques and ground-feet contact forces as $\boldsymbol{\tau}^+$ and $\boldsymbol{\lambda}^+$, respectively:

$$\boldsymbol{\tau}^+ = \boldsymbol{\tau} + \boldsymbol{K}_{\boldsymbol{\tau}} \boldsymbol{f} \tag{38}$$

$$\boldsymbol{\lambda}^+ = \boldsymbol{\lambda} + \boldsymbol{K}_{\boldsymbol{\lambda}} \boldsymbol{f}, \tag{39}$$

where $\boldsymbol{\tau}$ and $\boldsymbol{\lambda}$ are the *nominal* torques and ground-feet contact forces, $\boldsymbol{K}_{\boldsymbol{\tau}}$ and $\boldsymbol{K}_{\boldsymbol{\lambda}}$ are some (instantaneous) gain matrices which map a force expressed in end-effector space to joint-torque space and ground-feet contact space, respectively, and $\boldsymbol{f}$ is a potential external force applied at the end-effector. In a nominal situation, there are no disturbance forces and thus $\boldsymbol{f} = \boldsymbol{0}$, $\boldsymbol{\tau}^+ = \boldsymbol{\tau}$, and $\boldsymbol{\lambda}^+ = \boldsymbol{\lambda}$. Assuming no variation in accelerations, replacing $\boldsymbol{\tau}$ and $\boldsymbol{\lambda}$ in the equations of motion (24) with the right-hand side of (38) and (39) gives:

$$\boldsymbol{0} = \left(\boldsymbol{S}^\top \boldsymbol{K}_{\boldsymbol{\tau}} + \boldsymbol{J}_s^\top \boldsymbol{K}_{\boldsymbol{\lambda}} + \boldsymbol{J}_e^\top \right) \boldsymbol{f} \tag{40}$$

Alternatively to constraints (31) and (33)–(34), we can represent the actuator torque bounds and friction cones constraints using $\boldsymbol{\tau}^+$ and $\boldsymbol{\lambda}^+$ as:

$$\boldsymbol{A}_{\boldsymbol{\tau}} \boldsymbol{\tau}^+ \leq \boldsymbol{b}_{\boldsymbol{\tau}} \tag{41}$$

$$\boldsymbol{A}_{\boldsymbol{\lambda}} \boldsymbol{\lambda}^+ \leq \boldsymbol{b}_{\boldsymbol{\lambda}}. \tag{42}$$

We then substitute (38)–(39) into (41)–(42) and for each row $\boldsymbol{a}_{\boldsymbol{\tau}}$ of matrix $\boldsymbol{A}_{\boldsymbol{\tau}}$ we write the constraint as:

$$\boldsymbol{a}_{\boldsymbol{\tau}}^\top \left(\boldsymbol{\tau} + \boldsymbol{K}_{\boldsymbol{\tau}} \boldsymbol{f} \right) \leq \boldsymbol{b}_{\boldsymbol{\tau}} \qquad \forall \, |\boldsymbol{f}| \leq \rho, \tag{43}$$

where ρ is the radius of the maximum volume inscribed ball of a polytopic projection, and it represents the magnitude of the smallest potential disturbance that cannot be directly rejected. We then define $\boldsymbol{f} = \rho \chi$, where $\chi \in \mathbb{R}^3$ is a vector with unit length, which allows us to find the greatest ρ with:

$$\left(\max_{\chi} \quad \boldsymbol{a}_{\boldsymbol{\tau}}^\top \left(\boldsymbol{\tau} + \boldsymbol{K}_{\boldsymbol{\tau}} \rho \chi \right) \right) \leq \boldsymbol{b}_{\boldsymbol{\tau}}. \tag{44}$$

The objective function of the left-hand side of equation (44) can be seen as a scalar product of the vectors $\boldsymbol{a}_{\boldsymbol{\tau}}^\top \boldsymbol{K}_{\boldsymbol{\tau}} \rho$ and χ, which is greatest when these vectors are collinear:

$$\underset{\chi}{\operatorname{argmax}} \quad \boldsymbol{a}_{\boldsymbol{\tau}}^\top \boldsymbol{K}_{\boldsymbol{\tau}} \rho \chi \quad \equiv \quad \frac{\boldsymbol{K}_{\boldsymbol{\tau}}^\top \boldsymbol{a}_{\boldsymbol{\tau}}}{\|\boldsymbol{a}_{\boldsymbol{\tau}}^\top \boldsymbol{K}_{\boldsymbol{\tau}}\|}. \tag{45}$$

Simplifying (44) with the right-hand side of (45) leads to:

$$\boldsymbol{a}_{\boldsymbol{\tau}}^\top \boldsymbol{\tau} + \left\| \boldsymbol{a}_{\boldsymbol{\tau}}^\top \boldsymbol{K}_{\boldsymbol{\tau}} \right\| \rho \leq \boldsymbol{b}_{\boldsymbol{\tau}}. \tag{46}$$

---

[5] This is due to the fact that computing derivatives of an LP would require a differentiable solver. Solving an LP inside an optimisation problem can also lead to higher computational time.

Equations (43)–(46) address the constraints on actuation limits. We repeat the same process for the ground-feet contact forces to obtain:

$$\boldsymbol{a}_{\boldsymbol{\lambda}}^{\top}\boldsymbol{\lambda} + \left\|\boldsymbol{a}_{\boldsymbol{\lambda}}^{\top}\boldsymbol{K}_{\boldsymbol{\lambda}}\right\|\rho \leq \boldsymbol{b}_{\boldsymbol{\lambda}}. \tag{47}$$

Next, we substitute $\overline{\boldsymbol{K}}_{\boldsymbol{\tau}} = \boldsymbol{K}_{\boldsymbol{\tau}}\rho$ and $\overline{\boldsymbol{K}}_{\boldsymbol{\lambda}} = \boldsymbol{K}_{\boldsymbol{\lambda}}\rho$ into (40), (46) and (47) and write:

$$\boldsymbol{S}^{\top}\overline{\boldsymbol{K}}_{\boldsymbol{\tau}} + \boldsymbol{J}_s^{\top}\overline{\boldsymbol{K}}_{\boldsymbol{\lambda}} + \boldsymbol{J}_e^{\top}\rho = \boldsymbol{0}, \tag{48}$$

$$\boldsymbol{a}_{\boldsymbol{\tau}}^{\top}\boldsymbol{\tau} + \left\|\boldsymbol{a}_{\boldsymbol{\tau}}^{\top}\overline{\boldsymbol{K}}_{\boldsymbol{\tau}}\right\| \leq \boldsymbol{b}_{\boldsymbol{\tau}}, \tag{49}$$

$$\boldsymbol{a}_{\boldsymbol{\lambda}}^{\top}\boldsymbol{\lambda} + \left\|\boldsymbol{a}_{\boldsymbol{\lambda}}^{\top}\overline{\boldsymbol{K}}_{\boldsymbol{\lambda}}\right\| \leq \boldsymbol{b}_{\boldsymbol{\lambda}}. \tag{50}$$

This substitution removes the bilinear products between $\boldsymbol{K}_{\boldsymbol{\tau}}$, $\boldsymbol{K}_{\boldsymbol{\lambda}}$ and ρ while keeping the equality and inequalities valid.

4.6.2 *Constraints' Structure Exploitation*

We now extend the problem formulation and transcribe the constraints (48)–(50) directly into NLP constraints, and we extend the vector of decision variables with $\overline{\boldsymbol{K}}_{\boldsymbol{\tau}}$, $\overline{\boldsymbol{K}}_{\boldsymbol{\lambda}}$, and ρ. However, by trying this, one will soon realize we face an NP-hard problem. Additionally, there is a significant increase in the amount of decision variables, and the dependency of both $\boldsymbol{J}_s$ and $\boldsymbol{J}_e$ on joint positions means that constraint (48) is nonlinear and non-convex. Ultimately, this quickly renders any efforts of a naïve transcription ineffective, as the solver would be unable to digest it.

In order to solve this issue, we have to analyse the inherent structure of the problem and its constraints. Let $\overline{\boldsymbol{K}}_{\boldsymbol{\tau}}$ be the unknown in constraint (48). Splitting the structure of the constraint as

$$\begin{bmatrix} \boldsymbol{0} \\ \mathbb{I} \end{bmatrix} \overline{\boldsymbol{K}}_{\boldsymbol{\tau}} = - \begin{bmatrix} \boldsymbol{J}_s^{\top\,\text{base}} \\ \boldsymbol{J}_s^{\top\,\text{limbs}} \end{bmatrix} \overline{\boldsymbol{K}}_{\boldsymbol{\lambda}} - \begin{bmatrix} \boldsymbol{J}_e^{\top\,\text{base}} \\ \boldsymbol{J}_e^{\top\,\text{limbs}} \end{bmatrix} \rho \tag{51}$$

highlights that $\overline{\boldsymbol{K}}_{\boldsymbol{\tau}}$ can be obtained as a function of $\overline{\boldsymbol{K}}_{\boldsymbol{\lambda}}$ and ρ without performing any inversions. Doing this satisfies the bottom equality implicitly. The top part of the equality affecting the floating base still needs to be enforced.

This key-insight into the structure of the constraints allows us to transcribe the problem so that the solver will converge successfully.

4.6.3 *NLP Reformulation*

4.6.3.1 *Parameterization*

We extend the previous definition of $\boldsymbol{\xi}$ to accommodate for the extra decision variables required. Recall that $\overline{\boldsymbol{K}}_{\boldsymbol{\tau}k}$ need not be discretized.

$$\boldsymbol{\xi}^{+} \triangleq \boldsymbol{\xi} \cup \{\rho_1, \overline{\boldsymbol{K}}_{\boldsymbol{\lambda}1}, \cdots, \rho_N, \overline{\boldsymbol{K}}_{\boldsymbol{\lambda}N}\}. \tag{52}$$

4.6.3.2 *Objective*

$\mathcal{G}_3$ is the sum of all the ρ_k in $\boldsymbol{\xi}^+$:

$$\mathcal{G}_3 : \quad \underset{\boldsymbol{\xi}^+}{\operatorname{argmax}} \quad \sum_{k=1}^{M-1} \rho_k \tag{53}$$

4.6.3.3 *Constraints*

We bound all the ρ_k in $\boldsymbol{\xi}^+$ to $\mathbb{R}^+$ with a linear one-sided inequality:

$$\rho_k \geq 0 \quad \forall k = 1 : M-1. \tag{54}$$

We enforce the top part of constraint (51) explicitly:

$$\boldsymbol{J}_s^{\top\text{base}} \overline{\boldsymbol{K}}_{\boldsymbol{\lambda}} + \boldsymbol{J}_e^{\top\text{base}} \rho = \boldsymbol{0} \tag{55}$$

Finally, (49) is rewritten as:

$$\boldsymbol{a}_{\boldsymbol{\tau}}^\top \boldsymbol{\tau} + \left\| \boldsymbol{a}_{\boldsymbol{\tau}}^\top \left(-\boldsymbol{J}_s^{\top\text{limbs}} \overline{\boldsymbol{K}}_{\boldsymbol{\lambda}} - \boldsymbol{J}_e^{\top\text{limbs}} \rho \right) \right\| \leq \boldsymbol{b}_{\boldsymbol{\tau}}, \tag{56}$$

A summary of the constraints added to the NLP with the reformulation is shown in Table 13.

Table 13: Summary of the reformulated NLP constraints.

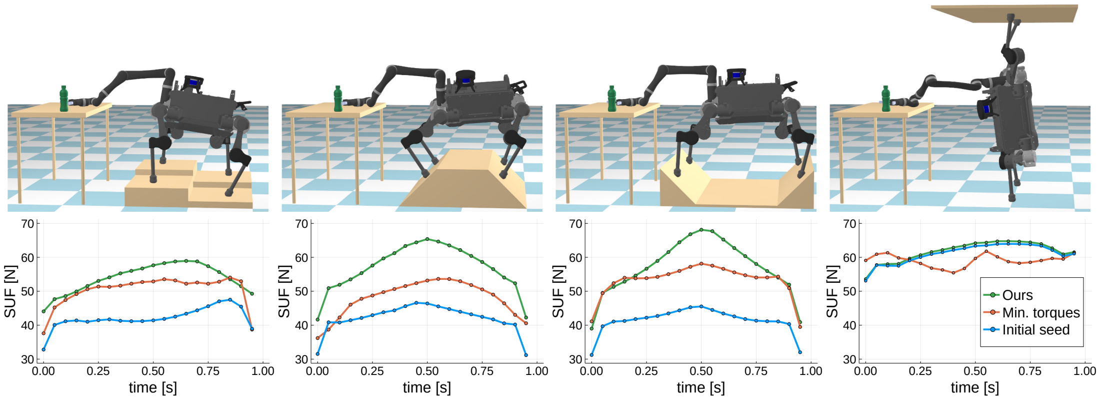
Constraint	Structure	Relation
Bounds on ρ	Linear	Inequality
Equation (55)	Nonlinear	Equality
Equation (56)	Nonlinear	Inequality
Equation (50)	Conic	Inequality

4.7 PERFORMANCE EVALUATION

In order to evaluate our work, we compared the robustness of the three objective functions proposed in Section 4.5.2: feasibility ($\mathcal{G}_1$), minimum torques ($\mathcal{G}_2$), and maximum SUF ($\mathcal{G}_3$). We ran the comparison across different scenarios for a pick-and-place task. Furthermore, we benchmarked the times taken to evaluate all NLP constraints and the convergence times for problems of different sizes.

Figure 18 shows four different settings for a pick-and-place task of a bottle on a table. We set up scenarios with challenging terrain, where the robot stands on steps with different heights or inclined slabs. The trajectories optimised with our method ($\mathcal{G}_3$) demonstrated greater robustness, as shown in plots (a)–(c) in Figure 18. The initial guess used for $\mathcal{G}_2$ and $\mathcal{G}_3$ was the result of the feasibility problem $\mathcal{G}_1$.

We also verified the robustness of trajectories for different inclines. For this, we varied the grade of the slopes in the "ramp" scenario (Figure 18b) from 0° to 60°. The



(a) "Steps" terrain. (b) "Ramp" terrain. (c) "Valley" terrain. (d) "Handstand" scenario.

Figure 18: We set up a multitude of terrains for testing a pick-and-place task: flat ground, slabs at different heights (a), and inclined supports (b)–(c). An extreme scenario where the robot performs a "handstand" is shown in (d). The plots underneath each scene show the SUF (in newtons) for the trajectories computed using different optimisation objectives.

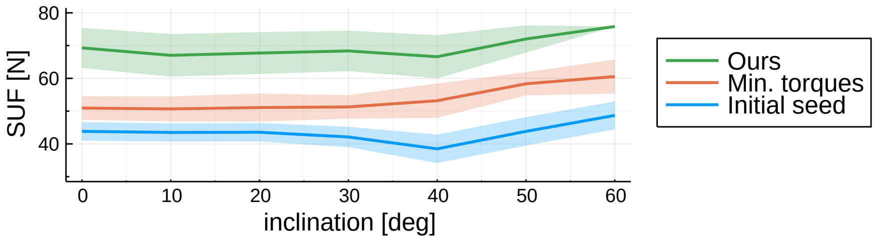


Figure 19: Mean and standard deviation of the SUF at the end-effector for varying inclinations on the "ramp" scenario.

trajectories computed with our metric consistently showed a greater SUF for all the tested slopes (see Figure 19).

As an extreme example, we created a scenario where the robot has to perform a "handstand", i.e., support its own weight on two of its legs (see Figure 18d) while using the remaining two for keeping its balance. In this scenario, it is especially important for the robot to press downwards against the floor and upwards against the ceiling to maintain stability. Because of this, torque minimization ($\mathcal{G}_2$) is not an appropriate objective for this scenario, as confirmed by the degraded SUF when compared to the initial seed in plot (d). On the other hand, our method is able to increase the robustness of the initial seed by a small amount, because it allows to trade off torque expenditure for more stable ground/ceiling-feet contact forces. We would like to emphasize that the motions in all the scenarios are within the actual physical capabilities of the robot, even the "handstand" scenario.

Table 14 shows the computation times for function and Jacobian evaluations of the NLP problem constraints. The longest time is spent computing the Jacobian of the system dynamics. Evaluating the Jacobian of the SUF—which is involved when optimising $\mathcal{G}_3$—takes the second-longest time.

Table 15 shows the total time it takes for the solver to converge for problems of different size. We benchmarked problems with 11, 21, and 41 mesh points (for a

Table 14: Times taken to evaluate the NLP constraints.

Constraint	Function (μs)	Jacobian (μs)
Gripper Task	9.15 ± 25.18	14.93 ± 2.47
Stationary Feet	18.29 ± 2.07	57.67 ± 225.53
System Dynamics	55.07 ± 169.64	3801.85 ± 1353.08
SUF Constraints	73.76 ± 239.97	1396.90 ± 989.31

Table 15: Convergence times of objectives $\mathcal{G}_1$–$\mathcal{G}_3$ for problems with different size: 11, 21, and 41 mesh points.

M	$\mathcal{G}_1$ (s)	$\mathcal{G}_2$ (s)	$\mathcal{G}_3$ (s)
11	0.46 ± 0.007	115.45 ± 0.27	229.34 ± 0.39
21	0.74 ± 0.009	143.48 ± 5.56	608.09 ± 8.04
41	1.21 ± 0.005	835.81 ± 15.59	1775.23 ± 12.85

1-second trajectory, this is the equivalent of a discretization at 10, 20, and 40 Hz). Each average and standard deviation are taken from five samples. $\mathcal{G}_1$ is the fastest to solve, as it is a feasibility problem and does not consider any optimality function. In our tests, the overall best robustness of $\mathcal{G}_3$ (shown in Figure 18) also comes with the trade-off of the longest times required until convergence.

All evaluations in this section were carried out in a single-threaded process on an Intel i7-6700K CPU with 4.0 GHz and 32 GB 2133 MHz memory. The proposed optimisation framework has been implemented using Julia [6] and the optimisation library Knitro [11]. The chosen solving algorithm was the interior-point method[6] presented by Waltz et al. [75].

4.8 EXPERIMENTS

We conducted hardware experiments on an ANYmal [37] quadruped equipped with a Kinova Jaco [12] robot arm. The motion planning is performed a priori, and the optimised trajectories are then sent to the controller for playback.

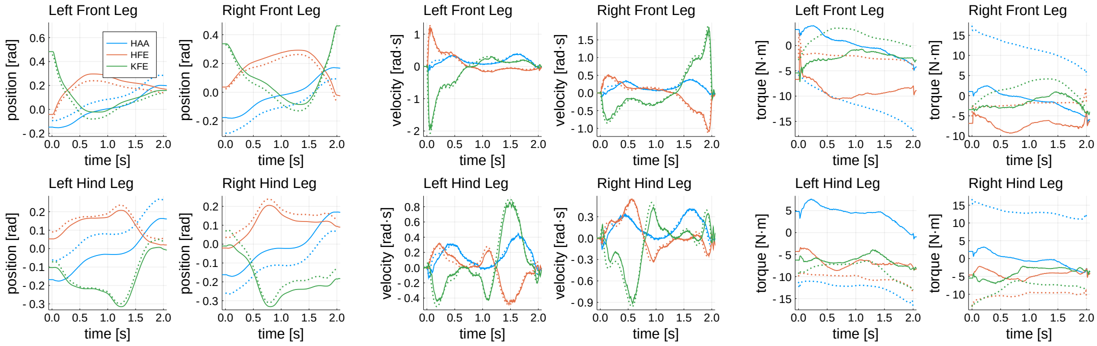
4.8.1 *Robot Control*

To execute our whole-body motions, we tracked the joint position with feed-forward velocity and torque. We updated the references for joint position, joint velocity, and joint torque at 400 Hz. The decentralized motor controller at every joint closes the loop, compensating for friction effects. During our experiments, we used $k_p = [150, 150, 100]$ as proportional and $k_d = [0.5, 0.5, 0.45]$ as derivative joint space gains for each leg, respectively. The arm used Kinova's driver for joint trajectory control. We synchronized the execution of both controllers.

---

[6] Interior-point methods (also known as barrier methods) replace the NLP problem by a series of barrier subproblems controlled by a barrier parameter. They are generally preferable for large-scale problems.

In order to evaluate how well the real robot tracks motions using our controller, we compared the planned joint states over time with the state estimation data from the robot. We computed a 2-seconds long trajectory using our framework for a pick-and-place task sampled at 400 Hz and commanded the robot at the same frequency. Figure 20 shows the plots of the planned trajectory against the data collected during our experiments. The plots show that joint positions are within acceptable tolerances and joint velocities are tracked well, but joint efforts are significantly different. This validates that the motions generated by our trajectory optimisation are dynamically consistent. The mismatch in joint efforts is expected due to differences between the real robot and our model, and also due to signal delays. Additionally, as we executed our trajectory open-loop without re-planning, the errors accumulated. Nonetheless, the tracking controller can execute the dynamic motions we planned.



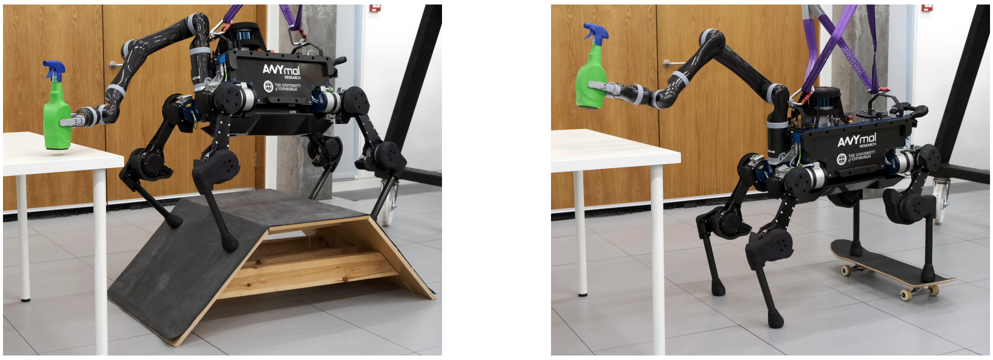
(a) Position

(b) Velocity

(c) Effort

Figure 20: Joint positions, velocities, and torques of ANYmal for a 2-seconds long trajectory on flat ground. The dotted lines correspond to the planned trajectory. The solid lines show the data collected by the state estimation on the real robot.

4.8.2 *Description of the Experiments*



(a) "Ramp" terrain

(b) "Skateboard" scenario

Figure 21: Snapshots of the real robot executing the planned motions on a ramp (see Figure 18b) and on a skateboard.

We executed the pick-and-place of a bottle on a table for different terrain: on flat ground and on a "ramp" (Figure 21a). The object being grasped was not modelled, and it is therefore an external disturbance. We also tested the trajectories for ground-feet friction coefficient mismatches by placing a skateboard underneath the feet of the robot (Figure 21b). A video is available at `https://youtu.be/vDesP7IpThw`.

For the motions shown in the video, we optimised the trajectories at 100 Hz and then linearly interpolated them to 400 Hz. It was the interpolation result that was then tracked by the controller. We did this because computing optimal trajectories with $\mathcal{G}_3$ gets more computationally expensive as the problem discretization increases (see Table 15).

To select the frequency of the trajectory before interpolation, we computed the root-mean-square error (RMSE) of the SUF over time for the same trajectory using different discretization resolutions, with a 400 Hz resolution as a baseline. As shown in Figure 22, for a trajectory discretized at 100 Hz its SUF RMSE ≈ 0.5 N, which is acceptable for our purposes.

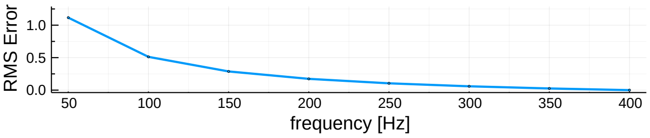


Figure 22: Root-mean-square error (RMSE) in newtons of the SUF given different discretisations, for a 400 Hz baseline.

4.9 CONCLUSION

In this chapter, we looked at how we can formulate trajectory optimisation problems that maximise robustness against external disturbances for legged robots with a floating base (underactuated systems). However, the current approach is only valid for standing balance behaviours.

In the next chapter, we are going to extend our formulation such that we can consider contact switches, i.e., the making and breaking of contacts between the feet of the robot and its environment.

5

ROBUST LOCO-MANIPULATION WITH APPLICATIONS TO INDUSTRY

Deployment of robotic systems in the real world requires a certain level of robustness in order to deal with uncertainty factors, such as mismatches in the dynamics model, noise in sensor readings, and communication delays. Some approaches tackle these issues *reactively* at the control stage. However, regardless of the controller, online motion execution can only be as robust as the system capabilities allow at any given state. This is why it is important to have good motion plans to begin with, where robustness is considered *proactively*. To this end, we propose a metric (derived from first principles) for representing robustness against external disturbances. We then use this metric within our trajectory optimisation framework for solving complex loco-manipulation tasks. Through our experiments, we show that trajectories generated using our approach can resist a greater range of forces originating from any possible direction. By using our method, we can compute trajectories that solve tasks as effectively as before, with the added benefit of being able to counteract stronger disturbances in worst-case scenarios.

5.1 INTRODUCTION

In this chapter, we tackle the problem of robust loco-manipulation for quadruped robots equipped with robot arms, such as the one shown in Figure 23. Here, the challenge is not only to generate whole-body trajectories for solving complex tasks requiring simultaneous locomotion and manipulation (commonly referred to as *loco-manipulation*), but also to optimise the robustness of such trajectories against unknown external disturbances. This is an important problem for two reasons: first, loco-manipulation

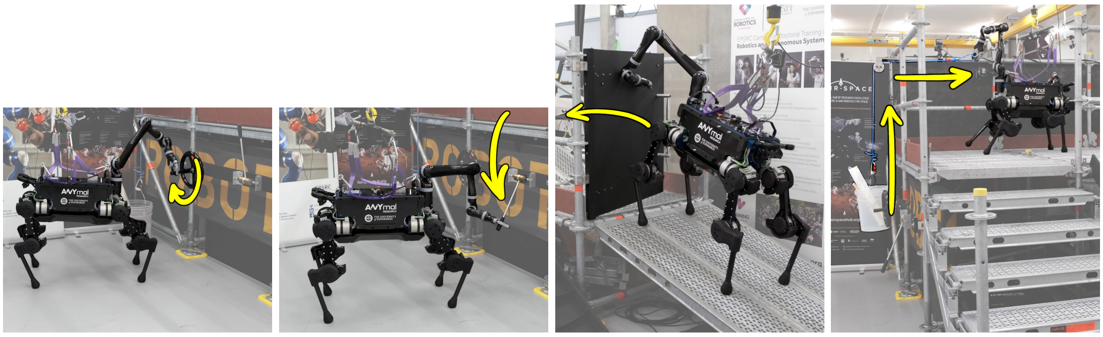

(a) (b) (c) (d)

Figure 23: Snapshots of our robot solving real-world tasks in an industrial setting: (a) turning a hand wheel, (b) pulling a lever, (c) opening a gate whilst standing on a ramp, and (d) lifting a bucket by pulling a rope. The robot and the objects being manipulated have been highlighted for clarity. The overlaid yellow arrows indicate motion. Video footage: `https://youtu.be/3qXNHVCagL8`.

allows us to extend the workspace of an otherwise-fixed-base-manipulator through the mobility of a mobile base, such as a legged robot; and second, increasing the robustness of the overall motion against disturbances leads to systems that are more reliable, and that can therefore be deployed in the real world with greater confidence.

Enabling loco-manipulation is a very challenging problem because it involves repeatedly breaking and making contacts between the feet and the environment in order to move around, while maintaining balance and avoiding kinematic/actuation limits. Moreover, robust motion planning is a complex subject on its own, as it requires the derivation of good metrics that are able to quantify how robust trajectories are. Consequently, combining loco-manipulation and robustness is not trivial, as it brings together the challenges from both problems.

Most of the previous research done on loco-manipulation, [47, 52, 87], has tackled the problem by splitting the arm from the base, planning the manipulation separately from the locomotion, and then considering arm movement as a disturbance that the base should compensate for. Furthermore, most of the existing research [19, 47, 61, 79] has focused on reactive robustness at the control stage, rather than taking into account robustness proactively during planning.

The key components of our approach are (i) a trajectory optimisation framework which is able to solve complex real-world tasks and which takes into account the full system dynamics, and (ii) a robustness metric derived from first principles. This allows us to calculate the largest force magnitude that the robot can counteract from any given direction, while considering ground-feet contact stability and the actuation limits of the system. Our results show that, given a contact sequence, our framework is able to plan whole-body trajectories that are significantly more robust to external disturbances compared to other approaches.

This chapter is a direct follow-up of our previous work. In the previous chapter, our framework was able to optimise whole-body trajectories for standing balance behaviours only, i.e., not *actual* loco-manipulation. In contrast, this chapter proposes an improved formulation which is able to optimise trajectories involving making, breaking and switching contacts. We are also able to better enforce the nonlinear dynamics of rigid-body systems, as we have incorporated our findings from [28]. Furthermore, we show that our formulation can handle cases where contact positions are not enforced explicitly, which actually allows it to further maximize robustness through the adoption of more suitable feet contact positions. Finally, the significant amount of systems integration work that we have done in comparison to [29] allowed us to deploy the robot in a realistic scenario mimicking an industrial offshore platform, where we showed the robot operating uninterrupted and repeatedly solving the complex sequence of tasks highlighted in Figure 23. This has resulted in a robust loco-manipulation system which is capable of online motion planning for deployment in realistic scenarios.

5.2 RELATED WORK

We now summarize previous research related to motion planning for quadruped robots equipped with arms, as well as existing research on the topic of robustness.

5.2.1 *Planning and Control for Quadrupeds with Arms*

Murphy *et al.* [52] were one of the first to investigate the use of a legged robot base to improve the capabilities of a robotic arm. They used trajectory optimisation (TO) with a simplified dynamics model to generate open-loop behaviours for Boston Dynamics' robot BigDog. As a result of the coordinated motion between the robot arm and the robot base, they were able to increase the performance of lifting and throwing tasks. In their hardware experiments, they showed the robot dynamically tossing cinder blocks as heavy as 16.5 kg and as far as 4.2 m. However, they only considered standing balance behaviours (which do not change support contacts/move the feet). In contrast, we consider the full dynamics model of the robot during TO, and we also consider behaviours where the robot's feet can make and break contact with the environment.

A few years later, Zimmermann *et al.* [87] equipped Boston Dynamics' flagship quadruped robot Spot with a Kinova arm in order to perform dynamic grasping manoeuvres. Direct control of Spot's actuated joints is not possible because of restricted access to its low-level controller. As Spot's whole-body controller for locomotion compensates for the wrench induced by the manipulator, the tracking and executed motion can differ from planned motion, resulting in poor performance. To achieve precise loco-manipulation, Zimmermann *et al.* treated Spot's overall behaviour as a black box, and built a simplified model of the combined platform from experimental data. While their approach successfully grasped the target most of the time, it failed when the estimated position of the ball was inaccurate or when the robot started executing the planned trajectories from slight offset poses. There were also cases when the robot failed to grasp the target due to Spot's complex internal behaviour not being fully captured by their simplified model. For example, when the disturbance induced by the arm to the base was sufficiently large, it could cause a delay that brought the individual trajectory components out of sync. In contrast, our approach does not suffer from this drawback because the robot we use in our experiments grants us full control over its joints, and because we optimise trajectories for the robot considering the dynamics of its whole body.

Ma *et al.* [47] combined manipulation using model predictive control (MPC) with a locomotion policy obtained from reinforcement learning (RL). First, they modelled the wrenches (arising from the motion of the arm) applied to the base of the robot as external disturbances that can be predicted. Then, they trained the base control policy to counteract those disturbances while (i) trying to keep a horizontal base orientation and (ii) tracking velocity commands from the MPC controller of the arm. In other words, their base policy uses wrench predictions from the arm's motion to compensate for the disturbances applied to the base of the robot.

All the previous work mentioned thus far [47, 52, 87] have one thing in common: they all see the arm as a disturbance to be compensated for. In contrast, Bellicoso *et al.* [3] approached the problem differently, and took into account the dynamics of the whole system. The authors used a whole-body controller based on inverse dynamics, re-planned locomotion continuously in a receding-horizon fashion, and explicitly provided end-effector forces for the controller to track. They equipped ANYbotics' quadruped robot ANYmal B with a Kinova arm—a combination which results in a fully torque-controlled mobile manipulator, and one which users have full control over. Bellicoso *et al.* demonstrated that the resulting system is able to perform dynamic locomotion while executing manipulation tasks, such as opening doors, delivering payloads, and human-robot collaboration.

In [52], the manipulation task is planned offline and separately from the locomotion planner; and in [3], the task for opening the door uses a controller that tracks gripper forces, which need to be explicitly specified. Sleiman *et al.* [61] tackled both these weaknesses when they proposed a unified MPC framework for whole-body loco-manipulation. Their approach augments the dynamics of the object being manipulated to the centroidal dynamics and full kinematics of the robot. This allows the solver to exploit the base-limb coupling and, e.g., to use the arm as a balancing "tail". Despite using an MPC approach, their planner is not adaptive with respect to the dynamic properties of the objects being manipulated.

In our previous work [29], we proposed a motion planning framework for legged robots equipped with manipulators. In that work—and in this chapter—we used the same robot arm and quadruped shown in [3], with a difference only in the number of fingers on the gripper. Our motion planning approach was similar to [61] in the sense that we formulated the planning problem for the whole body of the robot in a unified manner, i.e., for the quadrupedal base and the robot manipulator simultaneously. There are a couple of differences in the problem formulation between our previous work [29] and [3, 61]; e.g., we use a full model of the robot's articulated rigid-body dynamics[1] instead of a simplified version, which allows us to plan trajectories more faithfully to the real hardware. However, the most important difference and our main contribution is that we focus on planning motions that are not only physically feasible, but that also maximize robustness against unknown external disturbances. This aspect is something that none of the previous work mentioned [3, 47, 52, 61, 87] have considered.

5.2.2 *Motion Robustness Against Disturbances*

Del Prete *et al.* [19] proposed a solution to improve the robustness to joint-torque tracking errors at the control stage. The authors modelled deterministic and stochastic uncertainties in joint torques within their control framework optimisation. In our case, we maximize the upper-bound force magnitude the system can withstand from any possible direction, and we do this during the planning stage. Xin *et al.* [79] proposed a hierarchical controller in which external forces are estimated directly. Their goal was to minimize actuator torques while enforcing constraints for the contact forces. However, in contrast to our work, they did not explicitly enforce actuator limits; and since their main focus is on control, they do not have a planner for computing elaborate whole-body behaviours.

The robot experiments shown in [47] and in [61] demonstrate some capability of resistance against external forces; but this robustness is *reactive*, in the sense that it is either the learned locomotion policy or the MPC that compensate for the disturbances online in a reactive fashion. In contrast, research on *proactive* robustness [13, 30, 55] attempts to take uncertainty into account at the planning stage, i.e., ahead (or just in time) of online execution. The latter approach allows planning frameworks to increase the system's ability of counteracting disturbances by exploiting kinematic redundancy.

---

[1] Henceforth, we shall refer to this as the 'full dynamics model' of the robot.

In [13], Caron *et al.* proposed a feasible region (a *polytope*), called gravito-inertial wrench cone (GIWC), which can be used as a general stability criterion. This representation is very efficient for testing the robust static equilibrium of a legged robot, but it neglects the system's actuation limits. Subsequently, Orsolino *et al.* [55] proposed to extend the properties of the GIWC by incorporating the torque limits of the system. They demonstrated how the resulting polytopes can be used to e.g. optimise the centre of mass (CoM) trajectory in the xy-plane for the base-transfer motion of quadrupeds. Orsolino *et al.* formulated a reduced version of the problem, but even then the technique used to compute polytopes was prohibitively expensive. As a workaround, they computed the polytope only once for the first point of the trajectory, and used that as an approximation for the rest of the motion. We have followed this line of research in our previous work [30], where we proposed a force polytope representation, called *residual force polytope*, which considers not only the torque limits but also the dynamics of the system during trajectory execution. The polytope is computed from the forces and torques remaining after accounting for Coriolis, centrifugal, and gravity terms, as well as from the nominal feed-forward torques of the motion.

The polytope calculations in [55] and [30] require significant computation time and, in general, deriving explicit descriptions of a projected polytope is NP-hard [70]. Zhen *et al.* [85] formulated a computationally-tractable approach for finding maximally-sized convex bodies inscribed in projected polytopes. Later, Wolfslag *et al.* [78] adapted that approach for computing the robustness of static robot configurations. Their work was an improvement over exact computations; however, it was still too complex for being considered in trajectory optimisation. In our previous work [29], we adapted the technique from [85] to reformulate the problem of computing the smallest unrejectable force (SUF), which allowed us to formulate bilevel trajectory optimisation problems for maximizing the robustness of the generated trajectories. In this chapter, our work extends those ideas further to motion with contact changes.

5.3 ROBUST TRAJECTORY OPTIMISATION

5.3.1 *Robot Model Formulation*

We formulate the model of a legged robot in the same way as in our previous work [29], i.e., as a free-floating base B to which the limbs are attached to. For example, the robot we used for our experiments (seen in Figure 23), has four legs and one arm attached to its base; each leg has three motors and the arm has six.[2] We describe the motion of the system with respect to (w.r.t.) a fixed inertial frame I. We represent the position of the free-floating base w.r.t. the inertial frame, and expressed in the inertial frame, as ${}_I\boldsymbol{r}_{IB} \in \mathbb{R}^3$. As for the orientation of the base, we represent it using modified rodrigues parameters (MRP) [34, 69] as $\boldsymbol{\psi}_{IB} \in \overline{\mathbb{R}}^3$. The joint angles describing the configuration of the 6-degrees of freedom (DoF) arm and the four 3-DoF legs are stacked in a vector

---

[2] Please note that our formulation is not tied to the specific robot shown in Figure 23. In fact, it is general enough such that it can be applied to any legged robot, biped or quadruped, with or without arms.

$\boldsymbol{q}_j \in \mathbb{R}^{n_j}$, where $n_j = 18$. Finally, we write the generalized coordinates vector $\boldsymbol{q}$ and the generalized velocities vector $\boldsymbol{v}$ as

$$\boldsymbol{q} = \begin{bmatrix} {}_I\boldsymbol{r}_{IB} \\ \boldsymbol{\psi}_{IB} \\ \boldsymbol{q}_j \end{bmatrix} \in \mathbb{R}^3 \times \overline{\mathbb{R}}^3 \times \mathbb{R}^{n_j}, \quad \boldsymbol{v} = \begin{bmatrix} \boldsymbol{\nu}_B \\ \dot{\boldsymbol{q}}_j \end{bmatrix} \in \mathbb{R}^{n_v}, \tag{57}$$

where the twist $\boldsymbol{\nu}_B = \begin{bmatrix} {}_I\boldsymbol{v}_B & {}_B\boldsymbol{\omega}_{IB} \end{bmatrix}^\top \in \mathbb{R}^6$ encodes the linear and angular velocities of the base B w.r.t. the inertial frame expressed in the I and B frames, and $n_v = 6 + n_j$.

The equations of motion of a floating-base rigid-body system that interacts with the environment are written as

$$\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{v}} + \boldsymbol{h}(\boldsymbol{q}, \boldsymbol{v}) = \boldsymbol{S}^\top \boldsymbol{\tau} + \boldsymbol{J}_s^\top(\boldsymbol{q})\boldsymbol{\lambda} + \boldsymbol{J}_e^\top(\boldsymbol{q})\boldsymbol{f}, \tag{58}$$

where $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n_v \times n_v}$ is the mass matrix, and $\boldsymbol{h}(\boldsymbol{q}, \boldsymbol{v}) \in \mathbb{R}^{n_v}$ is the vector of Coriolis, centrifugal, and gravity terms. On the right-hand side of the equation, $\boldsymbol{\tau} \in \mathbb{R}^{n_\tau}$ is the vector of joint torques commanded to the system, and the selection matrix $\boldsymbol{S} = \begin{bmatrix} \boldsymbol{0}_{n_\tau \times (n_v - n_\tau)} & \mathbb{I}_{n_\tau \times n_\tau} \end{bmatrix}$ selects which DoF are actuated. We consider that all limb joints are actuated, thus $n_\tau = n_j$. The vector $\boldsymbol{\lambda} \in \mathbb{R}^{n_s}$ denotes the forces and torques experienced at the contact points, with n_s being the total dimensionality of all contact wrenches. The support Jacobian $\boldsymbol{J}_s \in \mathbb{R}^{n_s \times n_v}$ maps the contact wrenches $\boldsymbol{\lambda}$ to joint-space torques, and it is obtained by stacking the Jacobians which relate generalized velocities to limb end-effector motion as $\boldsymbol{J}_s = \begin{bmatrix} \boldsymbol{J}_{C_1}^\top & \cdots & \boldsymbol{J}_{C_{n_c}}^\top \end{bmatrix}^\top$, with n_c being the number of limbs in contact. Finally, $\boldsymbol{f}$ represents any external force applied to the end-effector. This force may be the result of a push or of some unpredicted disturbance. Under nominal circumstances, this force is zero, i.e., $\boldsymbol{f} = \boldsymbol{0}$. The Jacobian $J_e \in \mathbb{R}^{3 \times n_v}$ is used to map a linear force $\boldsymbol{f}$ applied at the end-effector to joint-space torques.

5.3.2 *Problem Discretization*

In order to plan motions for complex robot systems, we use an approach called *direct transcription*, which is a powerful technique for TO.

We start by converting the original motion planning problem (which is *continuous* in time) into a numerical optimisation problem that is *discrete* in time. We divide the trajectory into N equally spaced segments, $t_I = t_1 < \cdots < t_M = t_F$, where t_I and t_F are the start and final instants, respectively. This division results in $M = N + 1$ discrete *mesh points*, for each of which we explicitly discretize the states of the system, as well as the control inputs. Let $x_k \equiv x(t_k)$ and $u_k \equiv u(t_k)$ be the values of the state and control variables at the k-th mesh point. We treat $x_k \triangleq \{\boldsymbol{q}_k, \boldsymbol{v}_k\}$ and $u_k \triangleq \{\boldsymbol{\tau}_k, \boldsymbol{\lambda}_k\}$ as a set of nonlinear programming (NLP) variables, and formulate the basis of our trajectory optimisation problem as

$$\text{find } \boldsymbol{\xi} \quad \text{s.t. } x_{k+1} = f(x_k, u_k), \;\; x_k \in \mathcal{X}, \;\; u_k \in \mathcal{U}, \tag{59}$$

where $\boldsymbol{\xi}$ is the vector of decision variables, $x_{k+1} = f(x_k, u_k)$ is the state transition function incorporating the nonlinear dynamics of the system, and $\mathcal{X}$ and $\mathcal{U}$ are sets of feasible states and control inputs enforced by a set of equality and inequality constraints. The decision variables vector $\boldsymbol{\xi}$ results from aggregating the generalized coordinates $\boldsymbol{q}_{1:M}$, generalized velocities $\boldsymbol{v}_{1:M}$, joint torques $\boldsymbol{\tau}_{1:N}$, and contact forces $\boldsymbol{\lambda}_{1:N}$, i.e.,

$$\boldsymbol{\xi} \triangleq \{\boldsymbol{q}_1, \boldsymbol{v}_1, \boldsymbol{\tau}_1, \boldsymbol{\lambda}_1, \cdots, \boldsymbol{q}_N, \boldsymbol{v}_N, \boldsymbol{\tau}_N, \boldsymbol{\lambda}_N, \boldsymbol{q}_M, \boldsymbol{v}_M\}. \tag{60}$$

5.3.3 *System Constraints*

After having discretized the states and controls of the system over time as decision variables, we need to define a set of rules that restrict the motion represented by those variables. We do this by specifying a set of mathematical equalities and inequalities, so that the solver "knows" how to compute trajectories that are not only physically feasible but that also complete the tasks we want the robot to solve.

5.3.3.1 *Domain of decision variables*

The most straightforward constraints we need to write are the lower and upper bounds of each decision variable in $\boldsymbol{\xi}$. We constrain the joint positions, velocities, and torques to be within their corresponding lower and upper bounds.

$$\boldsymbol{q}_L \leq \boldsymbol{q}_k \leq \boldsymbol{q}_U \qquad \forall k = 1:M \tag{61}$$

$$\boldsymbol{v}_L \leq \boldsymbol{v}_k \leq \boldsymbol{v}_U \qquad \forall k = 1:M \tag{62}$$

$$\boldsymbol{\tau}_L \leq \boldsymbol{\tau}_k \leq \boldsymbol{\tau}_U \qquad \forall k = 1:M-1 \tag{63}$$

5.3.3.2 *Initial and final velocities*

We enforce the initial and final velocities of every joint to be zero, i.e., $\boldsymbol{v}_1 = \boldsymbol{v}_M = \mathbf{0}$. Note, however, that this is not a strict requirement of our framework, but is chosen to ensure static start and end configurations.

5.3.3.3 *End-effector poses*

We enforce end-effector poses with

$$f^{\text{fk}}(\boldsymbol{q}_k, i) = \boldsymbol{p}_i, \tag{64}$$

where $f^{\text{fk}}(\cdot)$ is the forward kinematics function, $\boldsymbol{q}_k$ are the joint coordinates at the k-th mesh point, i refers to the i-th end-effector of the robot, and $\boldsymbol{p}_i \in SE(3)$ is the desired pose. We use these constraints for defining the position and orientation of the robot's hand at specific mesh points, as well as to define the point contacts for the robot's feet during stance phases.[3] We pre-specify the contact sequence for the feet, which can be computed e.g. using contact planners such as [73].

5.3.3.4 *Contact forces*

For mesh points where the robot is *not* in contact with the environment (according to the pre-specified contact sequences), we enforce the contact forces at the respective contact points to be zero, i.e., $\boldsymbol{\lambda}_k = \mathbf{0}$.

---

[3] We do not constrain the robot's feet positions during leg swing phases.

5.3.3.5 *Friction cone constraints*

Similarly to [13], we model friction at the contact points using an *inner linear approximation* with a four-sided friction pyramid. Consider the set of points $\{C_i\}$ where the robot is in contact with its environment. Let $\boldsymbol{n}_i$ and μ_i be the unit normal and the friction coefficient of the support region at each contact, respectively. A point contact remains fixed as long as its contact force $\boldsymbol{f}_i^c$ lies inside the linearized friction cone directed by $\boldsymbol{n}_i$:

$$
\begin{aligned}
|\boldsymbol{f}_i^c \cdot \boldsymbol{t}_i| &\leq (\mu_i/\sqrt{2})(\boldsymbol{f}_i^c \cdot \boldsymbol{n}_i), && (65)\\
|\boldsymbol{f}_i^c \cdot \boldsymbol{b}_i| &\leq (\mu_i/\sqrt{2})(\boldsymbol{f}_i^c \cdot \boldsymbol{n}_i), && (66)\\
\boldsymbol{f}_i^c \cdot \boldsymbol{n}_i &> 0, && (67)
\end{aligned}
$$

where $(\boldsymbol{t}_i, \boldsymbol{b}_i)$ form the basis of the tangential contact plane, such that $(\boldsymbol{t}_i, \boldsymbol{b}_i, \boldsymbol{n}_i)$ is a direct frame.

5.3.3.6 *System dynamics*

In order to enforce the equations of motion (Equation 58), we use inverse dynamics rather than forward dynamics. This is because problems formulated using inverse dynamics are faster, more robust to coarser problem discretization, and converge in fewer iterations [28]. (We will discuss this in the next chapter.[4])

The inverse dynamics problem computes the joint torques and forces required to meet desired joint accelerations at a given state, i.e.,

$$
\boldsymbol{\tau}_k^* = f^{\text{id}}(\boldsymbol{q}_k, \boldsymbol{v}_k, \dot{\boldsymbol{v}}_k^*, \boldsymbol{\lambda}_k), \tag{68}
$$

where $f^{\text{id}}(\cdot)$ is the function that solves the inverse dynamics problem, and the desired joint accelerations can be calculated implicitly with $\dot{\boldsymbol{v}}_k^* = (\boldsymbol{v}_{k+1} - \boldsymbol{v}_k)/h$. We compute $\dot{\boldsymbol{q}}_{k+1}^*$ from $\boldsymbol{v}_{k+1}$, and integrate it (using semi-implicit Euler integration) to compute the next generalized coordinates $\boldsymbol{q}_{k+1}^*$. Finally, we define the dynamics defect constraints as

$$
\boldsymbol{q}_{k+1}^* - \boldsymbol{q}_{k+1} = \boldsymbol{0} \quad \text{and} \quad \boldsymbol{\tau}_k^* - \boldsymbol{\tau}_k = \boldsymbol{0}. \tag{69}
$$

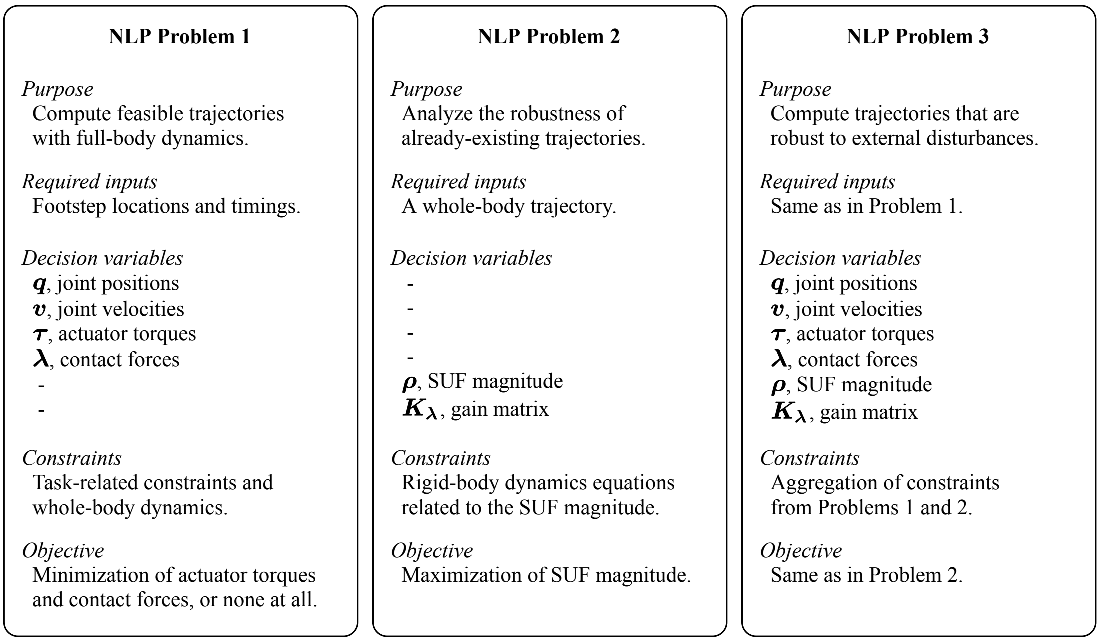
5.3.4 *Robustness Against Disturbances*

5.3.4.1 *NLP Problem 1*

Thus far, we have modelled the robot and its full body dynamics, discretized the motion planning problem, and defined a set of rules in the form of mathematical constraints. At this stage, we have all the "ingredients" required for planning feasible trajectories that can be executed on the robot. Henceforth, we will refer to this version of the formulation as NLP Problem 1—a summary for this version of the formulation is shown on the left block in Figure 24.

---

[4] Our work [28] on inverse dynamics vs. forward dynamics for computing dynamics defects in direct transcription problems was carried out (time-wise) between Chapter 4 and Chapter 5. However, in order to not break the flow regarding the research thread on robustness against external disturbances, we have decided to present that work as the last chapter of this thesis (i.e., just before the concluding chapter).



NLP Problem 1

Purpose
Compute feasible trajectories with full-body dynamics.

Required inputs
Footstep locations and timings.

Decision variables
$\boldsymbol{q}$, joint positions
$\boldsymbol{v}$, joint velocities
$\boldsymbol{\tau}$, actuator torques
$\boldsymbol{\lambda}$, contact forces
-
-

Constraints
Task-related constraints and whole-body dynamics.

Objective
Minimization of actuator torques and contact forces, or none at all.

NLP Problem 2

Purpose
Analyze the robustness of already-existing trajectories.

Required inputs
A whole-body trajectory.

Decision variables
-
-
-
-
$\boldsymbol{\rho}$, SUF magnitude
$\boldsymbol{K_{\lambda}}$, gain matrix

Constraints
Rigid-body dynamics equations related to the SUF magnitude.

Objective
Maximization of SUF magnitude.

NLP Problem 3

Purpose
Compute trajectories that are robust to external disturbances.

Required inputs
Same as in Problem 1.

Decision variables
$\boldsymbol{q}$, joint positions
$\boldsymbol{v}$, joint velocities
$\boldsymbol{\tau}$, actuator torques
$\boldsymbol{\lambda}$, contact forces
$\boldsymbol{\rho}$, SUF magnitude
$\boldsymbol{K_{\lambda}}$, gain matrix

Constraints
Aggregation of constraints from Problems 1 and 2.

Objective
Same as in Problem 2.

Figure 24: Block summaries of NLP Problems 1, 2, and 3. Inside each block, the summary states the purpose for using that formulation, the required inputs, the decision variables and constraints involved, and the objective function employed. The empty lines with a '-' under 'Decision variables' emphasize that NLP Problem 3 is a combination of Problems 1 and 2.

Next, we are going to build upon our previous work [29] to present two different problem formulations, NLP Problem 2 and NLP Problem 3, which can be used to analyse the robustness of known trajectories and to maximize the robustness of trajectories being computed, respectively.

5.3.4.2 *NLP Problem 2*

When we compute a robot trajectory using NLP Problem 1, we may be interested in understanding how robust those trajectories are against forces applied to e.g. the end-effector. Thus, one way of determining the robustness of that trajectory is by studying the set of forces that the end-effector is able to resist, both in terms of force magnitude and direction. The metric proposed in our previous work [29], the SUF, represents the smallest force magnitude (applied from any possible direction) that the robot is not able to counteract. In other words, it gives the magnitude of the largest force that the robot can counteract in a worst-case scenario. Next, we explain how to compute it.

Given a discretized robot trajectory (e.g., the output of NLP Problem 1), we can compute the SUF magnitude throughout that motion by re-formulating the NLP problem. The decision variables for such a problem are

$$\boldsymbol{\xi}_{\text{NLP Problem 2}} \triangleq \{\rho_1, \overline{\boldsymbol{K}}_{\boldsymbol{\lambda}1}, \cdots, \rho_N, \overline{\boldsymbol{K}}_{\boldsymbol{\lambda}N}\}, \tag{70}$$

where, for each k-th mesh point, ρ_k is the magnitude of the SUF and $\overline{\boldsymbol{K}}_{\boldsymbol{\lambda}k}$ is the instantaneous gain matrix mapping a force expressed in end-effector space to ground-feet contact space. Each and every ρ_k is bound to $\mathbb{R}^+$, i.e.,

$$\rho_k \geq 0 \quad \forall k = 1 : N. \tag{71}$$

$\overline{\boldsymbol{K}}_{\boldsymbol{\lambda}k}$ have no explicit bounds; but they are constrained by

$$\boldsymbol{a}_{\boldsymbol{\lambda}}^\top \boldsymbol{\lambda} + \left\| \boldsymbol{a}_{\boldsymbol{\lambda}}^\top \overline{\boldsymbol{K}}_{\boldsymbol{\lambda}} \right\| \leq \boldsymbol{b}_{\boldsymbol{\lambda}}, \tag{72}$$

with $\boldsymbol{a}_{\boldsymbol{\lambda}}$ and $\boldsymbol{b}_{\boldsymbol{\lambda}}$ pertaining to the alternative form of writing the friction cone constraints, i.e., $\boldsymbol{A}_{\boldsymbol{\lambda}}\boldsymbol{\lambda} \leq \boldsymbol{b}_{\boldsymbol{\lambda}}$.[5] The relationship between the SUF and the robot capabilities at every mesh point is given by

$$\begin{bmatrix} \mathbf{0} \\ \mathbb{I} \end{bmatrix} \overline{\boldsymbol{K}}_{\boldsymbol{\tau}} = - \begin{bmatrix} \boldsymbol{J}_s^{\top \text{base}} \\ \boldsymbol{J}_s^{\top \text{limbs}} \end{bmatrix} \overline{\boldsymbol{K}}_{\boldsymbol{\lambda}} - \begin{bmatrix} \boldsymbol{J}_e^{\top \text{base}} \\ \boldsymbol{J}_e^{\top \text{limbs}} \end{bmatrix} \rho. \tag{73}$$

We enforce this relationship by splitting it into two parts. For the top part of the equation, concerning the floating base, we write the following nonlinear equality:

$$\boldsymbol{J}_s^{\top \text{base}} \overline{\boldsymbol{K}}_{\boldsymbol{\lambda}} + \boldsymbol{J}_e^{\top \text{base}} \rho = \mathbf{0}. \tag{74}$$

As for the bottom part, concerning the limbs of the robot, we write the following nonlinear inequality:

$$\boldsymbol{a}_{\boldsymbol{\tau}}^\top \boldsymbol{\tau} + \left\| \boldsymbol{a}_{\boldsymbol{\tau}}^\top \left(-\boldsymbol{J}_s^{\top \text{limbs}} \overline{\boldsymbol{K}}_{\boldsymbol{\lambda}} - \boldsymbol{J}_e^{\top \text{limbs}} \rho \right) \right\| \leq \boldsymbol{b}_{\boldsymbol{\tau}}. \tag{75}$$

Once we have defined the above constraints and decision variables, we can maximize the following objective function with any off-the-shelf nonlinear solver:

$$\underset{\xi_{\text{NLP Problem 2}}}{\operatorname{argmax}} \quad \sum_{k=1}^{N} \rho_k. \tag{76}$$

In summary, for a given (constant) trajectory, the outcome of this nonlinear optimisation problem will be the magnitude of the SUF over time (i.e., the ρ_k for every mesh point of the discretized trajectory) and $\overline{\boldsymbol{K}}_{\boldsymbol{\lambda}k}$. $\overline{\boldsymbol{K}}_{\boldsymbol{\tau}}$ are also an output, since they can be computed as a function of $\overline{\boldsymbol{K}}_{\boldsymbol{\lambda}}$ and ρ without performing any inversion—as hinted by Equation 73. The outputs $\overline{\boldsymbol{K}}_{\boldsymbol{\lambda}}$ and $\overline{\boldsymbol{K}}_{\boldsymbol{\tau}}$ can be used to understand and explain the specific constraint that determines the upper bound of the SUF (i.e., friction cone or torque, on which foot or motor), although that is something we do not investigate in this chapter.

5.3.4.3 *NLP Problem 3*

NLP Problems 1 and 2 allow us to compute a feasible whole-body trajectory and to calculate the robustness of said trajectories, respectively. By combining NLP Problem 1 and NLP Problem 2, we obtain a single (albeit more complex) problem formulation which is able to compute whole-body trajectories that are not only feasible but also more robust against external disturbances. We call this single formulation *NLP Problem 3*, and it is summarized on the right block in Figure 24. The decision variables of NLP Problem 3 are

---

[5] We refer readers to [29] for a full explanation and derivation of the terms.

$$\begin{aligned}\boldsymbol{\xi}_{\text{NLP Problem 3}} \triangleq \{&\boldsymbol{q}_1, \boldsymbol{v}_1, \boldsymbol{\tau}_1, \boldsymbol{\lambda}_1, \rho_1, \overline{\boldsymbol{K}}_{\boldsymbol{\lambda}1},\\ &\dots,\\ &\boldsymbol{q}_N, \boldsymbol{v}_N, \boldsymbol{\tau}_N, \boldsymbol{\lambda}_N, \rho_N, \overline{\boldsymbol{K}}_{\boldsymbol{\lambda}N},\\ &\boldsymbol{q}_M, \boldsymbol{v}_M\}.\end{aligned} \tag{77}$$

The constraints are the combined constraints of NLP Problems 1 and 2. The objective function is the same as NLP Problem 2, i.e., the maximization of every mesh point's SUF added up:

$$\underset{\boldsymbol{\xi}_{\text{NLP Problem 3}}}{\operatorname{argmax}} \quad \sum_{k=1}^{N} \rho_k. \tag{78}$$

5.3.5 *Contact Switching*

In contrast to our previous work [29], the NLP formulations in Figure 24 consider the making and breaking of contacts between the feet of the robot and its environment. This is one of the main contributions of this chapter, and we will now explain how we have enabled this.

As we have previously explained in 'System Constraints' (Section 5.3.3), during problem discretization, we handle feet that are in contact with the ground (*stance phase*) differently from feet that are moving through free space (*swing phase*). In short, for each mesh point of the trajectory, and for each foot of the robot: if that foot is in stance phase, we enforce it to remain still, and we also enforce the contact force to lie within friction cone boundaries; otherwise, we enforce only a zero contact force constraint. Since this is done during the problem transcription process, it requires knowing the timings for contact switches *a priori*, as well as feet positions[6] for each stance phase. The upside is that the problem complexity does not increase much, especially compared with other approaches that consider contact switching by formulating complementarity constraints (e.g. Posa *et al.* [59]).

The approach explained in the last paragraph is enough for enabling contact switching in NLP Problem 1. However, additional changes are needed for Problems 2 and 3, since those problem formulations involve extra constraints and decision variables regarding the maximization of the SUF. In essence, the decision variables $\overline{\boldsymbol{K}}_{\boldsymbol{\lambda}k}$ pertaining to the feet in swing phase must be set to zero, since no contact forces exist in that context. Fortunately, we need not worry about $\overline{\boldsymbol{K}}_{\boldsymbol{\tau}_k}$, since it is defined as a function of $\overline{\boldsymbol{K}}_{\boldsymbol{\lambda}k}$ (see Equation 73). Nonetheless, the most challenging aspect of these modifications is not in the writing of the constraints, but rather in the writing of the functions that evaluate the Jacobian of those constraints (together with their sparsity structure). To this end, we use modern automatic differentiation capabilities from Julia [6], but the process of passing the results of the Jacobian and sparsity structure evaluations to specialized NLP solvers remains tricky and requires particular attention by the programmer to the indexing of the decision variables and constraints. We formulate large but sparse NLP problems through our framework and therefore specifying the sparsity pattern is very important to attain shorter computation times, as the Jacobian of the constraints contain mostly zeroes (especially the dynamics Jacobian, which is block diagonal).

---

[6] For now, we will consider the feet positions to be fixed, but we will release this limitation in Section 5.5, where we discuss this subject further.

5.4 EXPERIMENTS

We now present the experiments we carried out for evaluating our work and their respective results. This section is organized as follows:

6.4.1 Describes the system integration work required for combining the quadruped robot and the robot arm, as well as our planning framework with existing controllers;

6.4.2 Presents a repeatability test where we commanded the robot to turn an industrial hand wheel multiple times from different starting positions and orientations;

6.4.3 Compares the robustness of two distinct trajectories for turning a hand wheel and for pulling a lever;

6.4.4 Explains how the SUF can be used as a tool for analysing existing robot trajectories;

6.4.5 Demonstrates the capabilities of our framework to plan robust whole-body motions involving making and breaking of contacts; and finally,

6.4.6 Tests the robustness of a loco-manipulation trajectory for lifting a bucket with incremental weights until failure.

Subsections 6.4.1, 6.4.2, 6.4.3, and 6.4.6 are concrete evaluations of our method on real robot hardware. Whereas the focus of subsections 6.4.4 and 6.4.5 is to demonstrate new features.

In Table 16, we list all the videos supplementing our work, in the same order as they appear in this chapter. A playlist is also available here: `shorturl.at/csDY2`.

Table 16: Supplementary videos.

Description	Link (YouTube)
System demonstration	`youtu.be/3qXNHVCagL8`
Repeatability experiment	`youtu.be/Ok8Pcwn_I0w`
Robustly turning a wheel	`youtu.be/1M32AHuuDhI`
Robustly pulling a lever	`youtu.be/6A9eSdfcj7A`
SUF with contact switching	`youtu.be/H6-g8NLGyYE`
Test with incremental weights	`youtu.be/puy2S90_3CM`
Robust footstep locations	`youtu.be/tUXQUqLneTE`

5.4.1 *System Integration*

In order to demonstrate the capabilities of our planning framework on the real robot, we integrated it with the existing software stacks of the ANYmal quadruped and Kinova arm.

For the quadruped, we used one of ANYbotics' software releases, which comes with multiple locomotion controllers working out-of-the-box. Human operators can control the robot remotely via a joystick to tell the robot where to walk and which gait to use, or they can pre-specify a mission as a set of waypoints for the robot to walk through. Moreover, they provide an interface for specifying a custom payload attached to the robot, but this is assumed to be a static payload, such as a thermal camera, or an imaging sensor.

For controlling the robot as a whole (quadruped + arm), we settled on two operation modes: *teleoperated* and *autonomous*. When the robot is in *teleoperated mode*, we do not move the Kinova arm; it remains still in a "parked" configuration (stowed on top of the quadruped). Doing this allows us to consider the arm a static payload, which we can specify using the interface provided by ANYbotics. In turn, this allows us to take advantage of every existing capability provided by ANYbotics' stack. Finally, whenever we wish to operate the arm, we switch to *autonomous mode*. In this mode, we use a custom controller[7] for the quadruped base and Kinova's velocity controller for the arm. This is the mode we use to execute the whole-body trajectories generated with our planning framework.

In summary, we start the robot in *teleoperated mode* by default. In this mode, we can walk the robot on flat ground and over ramps, but we cannot move the arm. We use the *teleoperated mode* to walk the robot around a facility and towards points of interest. Then, once the robot reaches said points of interest, it awaits an instruction (e.g, "turn the wheel", or "pull the lever"). As soon as the robot is given an instruction, it switches into *autonomous mode*, our planning framework is triggered and computes a whole-body trajectory for the robot. This trajectory is then passed on to the arm and quadruped controllers for synchronous execution. Once the robot finishes executing the task autonomously, it switches back into *teleoperated mode* and the system returns control to the human operator, who can walk the robot remotely towards the next task of the mission.

A demonstration of the entire system is available here: `https://youtu.be/3qXNHVCagL8`. In the video, a human teleoperates the robot to walk around a mock-up scaffolding of an offshore platform (such as an oil rig). The operator approaches different points of interest, and commands the robot to autonomously turn a hand wheel, pull a lever, push a gate whilst standing on a ramp, and pull a rope to lift a 1.1 kg bucket. These tasks are the ones shown in Figure 23. Although we have focused on tasks relevant for industrial inspection, our framework allows us to formulate virtually any loco-manipulation task through the definition of a set of constraints for the robot.

---

[7] We use the same controller as in our previous work [29]. We commanded each joint of the quadruped with feedforward torque and feedback on position and velocity. Position, velocity, and torque references are updated at 400 Hz.

5.4.2 *Repeatability Test*

Our goal for this experiment was to ensure the following characteristics of our system:

- the robot is able to operate continuously for extended periods of time without falling;
- the operator is able to send walking commands to the robot during *teleoperated mode*, but not during *autonomous mode*;
- the Vicon motion capture system calculates the relative transform between the free-floating base of the robot and the manipulation target reliably;
- the planning framework computes a feasible whole-body trajectory for completing the manipulation task;
- the controller is able to reach and grasp targets, and track reference trajectories accurately.

To verify the points above, we carried out a repeatability test for turning an industrial hand wheel. During this test, we operated the robot continuously for 30 min without a safety harness. During this time, a human operator walked the robot to different points near the manipulation target and triggered the "turn wheel" behaviour. The test was completed successfully and our system passed all the points listed above. Figure 25 shows a few snapshots of the robot grasping the wheel during the test. Video footage is also available here: `https://youtu.be/0k8Pcwn_I0w`.

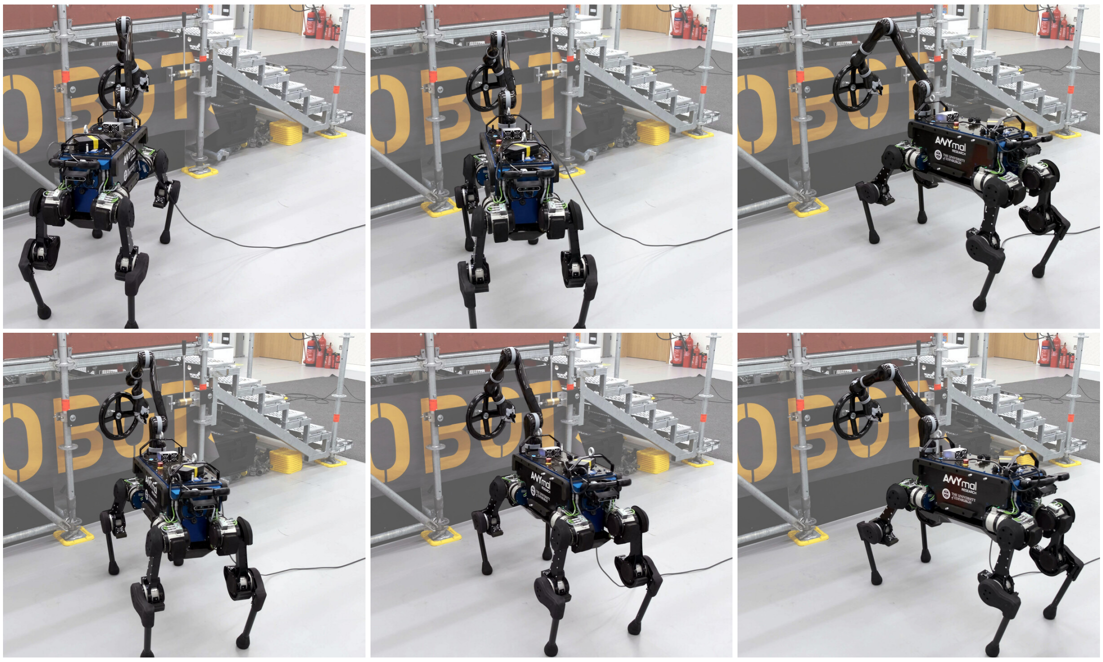
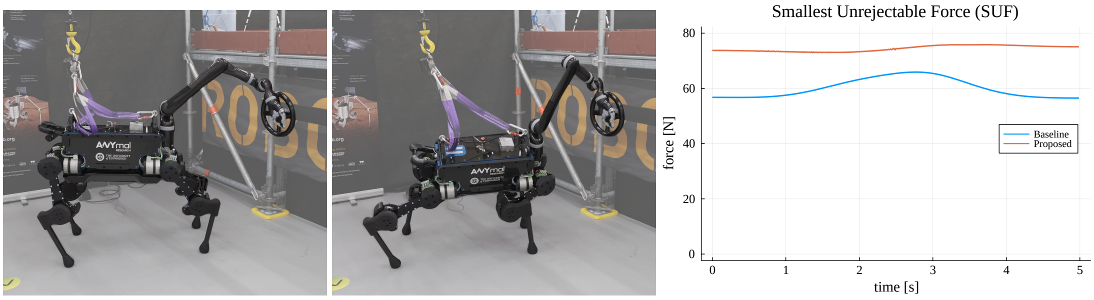


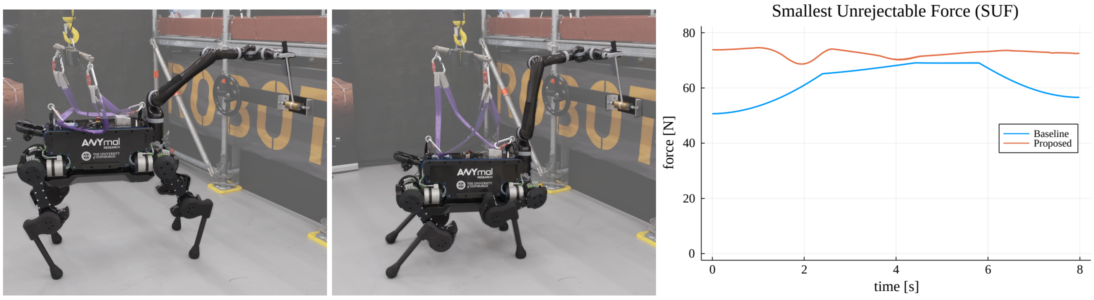
Figure 25: Snapshots of the robot grasping the hand wheel during the repeatability test. Notice how the position and orientation of the robot base relative to the wheel are different on every snapshot. Video: `https://youtu.be/0k8Pcwn_I0w`.

5.4.3 *SUF Optimisation for Turning a Wheel and Pulling a Lever*

In this experiment, our goal is to compare the resulting SUF of two different objective functions: the first trajectory (*baseline*) is optimised considering a cost function that minimizes torques and contact forces, while the second trajectory (*proposed*) maximizes the SUF explicitly. We run this experiment for two tasks: (i) turn a hand wheel clockwise by a full revolution, and (ii) pull down a lever from its resting position. Figure 26 shows an instance of the results of this experiment.



(a) Wheel



(b) Lever

Figure 26: Initial configuration of the *baseline* trajectories (left), initial configuration of the *proposed* trajectories (centre), and plot comparing the SUF of the *baseline* and *proposed* trajectories (right). Top and bottom rows concern the tasks of turning an industrial hand wheel and pulling down a lever, respectively.

5.4.3.1 *Turning the hand wheel*

In Figure 26's left column, we can see that the SUF magnitude of the proposed trajectory (orange line) is within 73 N to 76 N throughout the entire motion; whereas the SUF of the baseline trajectory (blue line) lies within 56 N to 66 N—it starts and ends at $\sim$56 N, increasing slightly in the middle of the trajectory where it peaks at $\sim$66 N. The SUF mean-percentage-increase of the proposed approach versus the baseline is approximately 24 %. Video: `https://youtu.be/1M32AHuuDhI`.

5.4.3.2 *Pulling the lever*

In Figure 26's right column, the SUF magnitude of the proposed trajectory (orange line) remains within 68 N to 75 N throughout the entire motion; on the other hand, the SUF magnitude of the baseline trajectory (blue line) remains within 50 N to 69 N.

The SUF mean-percentage-increase of the proposed approach versus the baseline is approximately 18 %. Video: `https://youtu.be/6A9eSdfcj7A`.

We repeated this experiment multiple times for different robot orientations relative to the wheel and the lever. The results were identical to those shown in Figure 26. Both the baseline and proposed trajectories successfully completed the task. However, trajectories planned using the proposed approach outperformed the baseline version in both tasks in terms of their robustness against external disturbances.

5.4.4 *The SUF as a Tool for Analysing Trajectories*

Earlier in this chapter, when we presented Figure 24, we explained how the SUF allows us to analyse the robustness of existing trajectories, and also how it allows us to compare multiple trajectories with each other. In this experiment, we want to further develop the idea of using the SUF as an analysis tool. Our goal is to show that, thanks to our problem formulation, we can analyse individual trajectories to understand e.g. how important of a role each of the legs play during a motion.

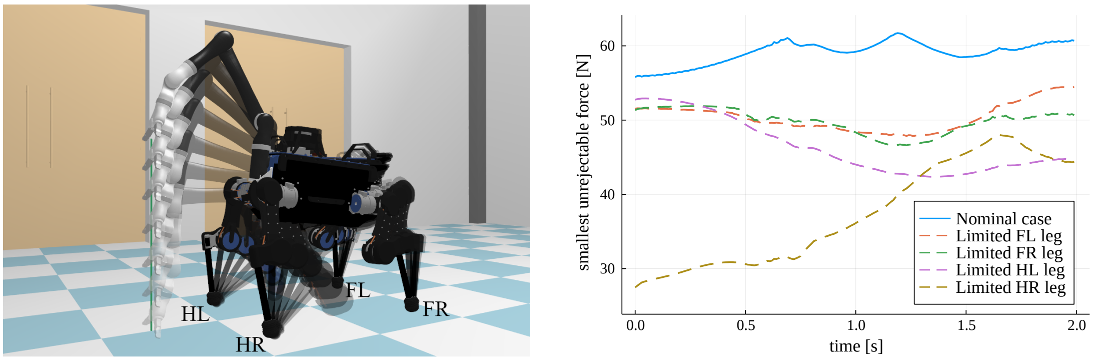
In Figure 27a, we show a robot trajectory planned with our framework for picking up something from the ground; and in Figure 27b, the solid blue line represents the SUF over time for that motion. An important question we may ask is: how important is each leg for a successful execution of the motion? We can ask a similar question from a different perspective: if there is a hardware fault on any of the quadruped legs, how much of the initial motion's robustness remains? Next, we explain how we can answer these questions.

Thanks to the way we formulated the SUF constraints, we can set individual terms in $\overline{\boldsymbol{K}}_{\lambda k}$ to zero in order to "simulate" what would happen if the torques of the motors of a specific leg of the robot were at their torque limits, or what would happen if the contact force at a specific foot was on the boundary of the friction cone—or both. We solved Figure 24's NLP Problem 2 four times (one for each leg) with the goal of analysing the SUF at the end-effector while considering the motor torques to be at their limits and the ground-feet contact forces to be at the boundaries of their respective friction cones. The dashed lines plotted in Figure 27b show the results.

We can see that the magnitude of the SUF decreases, regardless of which leg is affected. We can also see that the dashed lines are slightly different from each other, which is expected since the trajectory is not perfectly symmetric and each leg has a slightly different load from every other leg. Moreover, we can actually identify which leg plays the most important role in this motion by looking at the dashed line with the lowest values (the one for which the SUF magnitude decreased the most)—it was the hind right (HR) leg.

5.4.5 *SUF Optimisation with Making and Breaking Contacts*

In this experiment, we wanted to verify that our planning framework is able to optimise trajectories involving switching contacts. We also evaluated whether the proposed objective function is able to compute a trajectory that is more robust to external disturbances than the baseline cost.



(a) (b)

Figure 27: The image on the top shows a motion plan for picking up an object from the floor. The plot on the bottom shows, for each leg, how the SUF decreases if we were to consider the current torques to be at actuator limits and current contact force at the friction cone boundary.

We defined a task in which the robot starts with all four feet on pre-specified positions and then raises its right hind foot off the ground for a short time. We also constrained the gripper to remain still at a certain location for the duration of the whole motion. We then solved the task in two different ways: with NLP Problem 1 (baseline) and with NLP Problem 3 (optimised). The plot in Figure 28 shows the magnitude of the SUF over time for the resulting trajectories. Because NLP Problem 1 only outputs a trajectory, we passed it to NLP Problem 2 afterwards in order to compute the SUF values—keep in mind that this did not change the baseline trajectory.

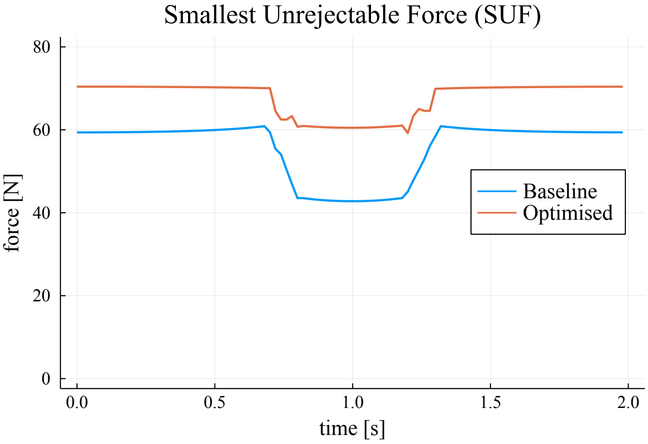
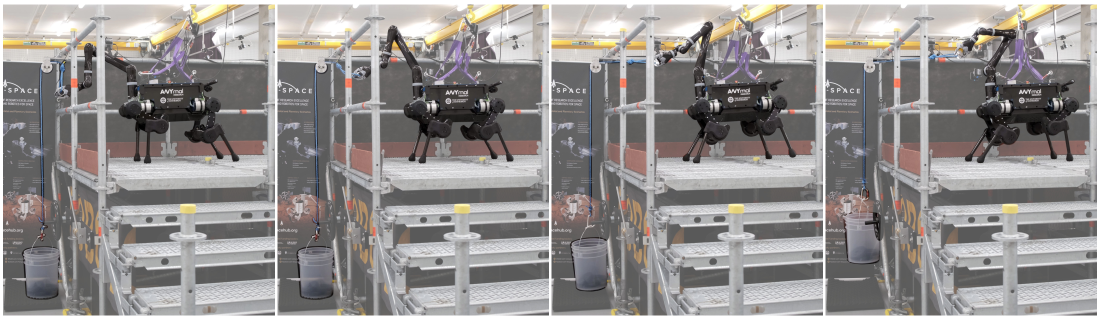


Figure 28: Plot of the SUF magnitude over time for two trajectories solving the same task specification. Blue shows the baseline version (solved with NLP Problem 1), and orange shows the optimised version (solved with NLP Problem 3).

The planner was able to compute a feasible trajectory for solving the task using both approaches. However, as we can see in Figure 28, the optimised trajectory is more robust than the baseline. Moreover, the plot shows that the breaking and making of contacts directly influences the magnitude of the SUF. For both trajectories, the SUF magnitude decreases when the right hind foot breaks contact with the floor; and then increases again as the foot re-establishes contact. Video: `https://youtu.be/H6-g8NLGyYE`.



(a) (b) (c) (d)

Figure 29: Snapshots of the robot lifting a bucket with weight plates, weighing 4.1 kg in total. This task requires the quadruped base to take steps at the same time as the arm moves (loco-manipulation). Despite the dynamics of the heavy bucket not being modelled, the robot is still able to complete the task, thanks to the robustness of the planned motion and to the feedback gains of the controller.

5.4.6 *Robustness Test with Incremental Weights*

In this last (but central) experiment, we plan a robust loco-manipulation trajectory for pulling a rope attached to a bucket and execute it on the real robot. Our goal is to show that the extended version of our framework can handle problems that require simultaneous locomotion and manipulation while maximizing robustness against disturbances. Planning loco-manipulation trajectories where robustness is considered proactively was not possible before, and this is really what we have been trying to tackle with our work in this chapter.

The experimental setup (shown in Figure 29) consisted of a rope threaded through a pulley. On one end, the rope was attached to a bucket on the ground; and on the other end, the rope was attached to a handle on a platform. The task for the robot was to grasp the handle from the top of the scaffolding and then pull the rope to lift the bucket. Importantly, in order to lift the bucket high enough, the robot had to take a few steps back while simultaneously pulling the rope with its arm. The position of the handle w.r.t. the robot was calculated with the Vicon motion capture system (similarly to the other tasks), and the footsteps were pre-specified using a static crawl gait.

To test the robustness of the trajectory planned with our framework, we asked the robot to lift the bucket in multiple trials, wherein the weight of the bucket was incremented 1 kg in each trial (and the bucket was empty on the first trial). The bucket itself weighs approximately 1.1 kg.

The robot completed 4 successful trials, in which it lifted 1.1, 2.1, 3.1, and 4.1 kg. On the 5th trial (5.1 kg) the robot was able to lift the bucket momentarily, but then the handle slipped off the gripper and we counted this as a failure. A video of the experiment is available: `https://youtu.be/puy2S90_3CM`, and Figure 29 shows some snapshots of the last successful trial (i.e., trial 4), where the robot lifted the bucket containing three 1 kg weight plates. Finally, Figure 30 shows the magnitude of the SUF over time for this task, where we can see the dips corresponding to the intervals when the robot was taking steps.

This experiment showed that our robot was capable of reliably executing a loco-manipulation trajectory planned with our framework. Moreover, it showed that the

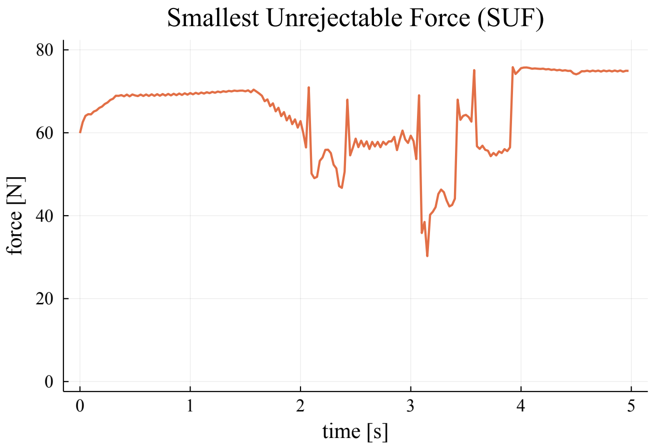


Figure 30: SUF magnitude over time for the bucket task.

robot is indeed capable of dealing with external disturbances while executing a trajectory. It is worth remembering that for this experiment (as well as for the other experiments shown in this chapter) we did not model the payload; in other words, the system does not know about the bucket or how much it weighs a priori (during planning). Instead, the weighted bucket is acting as a disturbance, and the robot's ability to lift the bucket is therefore a direct outcome of our robustness metric and the feedback terms used in our controller.

5.5 OPTIMISATION OF CONTACT LOCATIONS

Thus far, we pre-specified the position for each foot of the robot during the loco-manipulation planning stage. For example, when the human operator commands the robot to turn the hand wheel, our framework calculates the feet positions of the robot at that instant via forward kinematics, and then the mathematical constraints of our optimisation problem ensure the robot's feet remain on those positions while the wheel is being turned. But this raises an important question: when we optimise a trajectory that maximizes robustness, if we constrain the feet to specific positions, will that not limit how robust the resulting trajectory is? In other words, perhaps the resulting trajectory could be more robust if another set of feet locations had been chosen.

In light of this, we set out to investigate an approach for choosing more adequate contact locations within our motion planning framework. We now consider the additional problem of optimising the continuous location of the feet contacts. Our goal was to understand whether our planner is capable of adapting feet positions such that the resultant whole-body trajectory can be more robust.

In order to enable optimisation of feet locations, we extended our NLP formulation. For that, we expand the vector of decision variables of the NLP problem to include the xy-coordinates of each foot for the beginning of each individual stance phase. We added the xy-coordinates of each foot to our problem as decision variables, assuming that the robot would stand on flat ground (i.e., we assume the z-coordinate for each foot is zero). We also modified Equation 64 to consider those decision variables, since previously their right hand-side (i.e., $\boldsymbol{p}_i$) were a pre-specified constant position for each foot. The extended NLP formulation is more complex than the one previously used throughout this work: it has additional decision variables (some of them coupled[8]), and more complex constraints for enforcing the contact positions; however, it provides more flexibility to the solver, which should now be able to compute feasible trajectories that further maximize robustness by adapting foot contact locations.

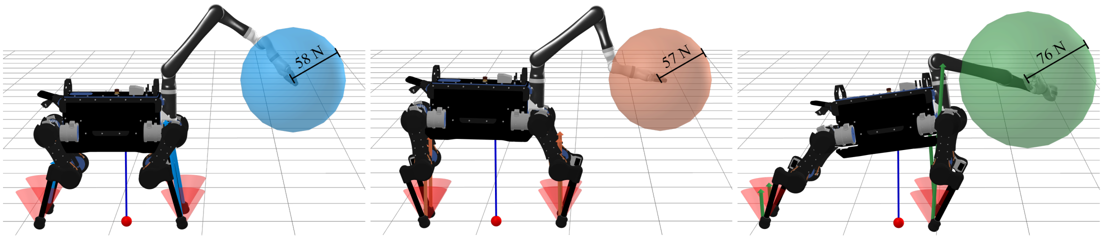
To test this more-flexible NLP formulation, we defined a motion planning task in which the robot must maintain its configuration while reaching a target point in task-space with its end-effector. Then, we solved three slightly different versions of the optimisation problem:

Nominal - In this version of the problem, we fixed the configuration of the robot base to its default joint positions (defined by the manufacturer). We did not constrain the configuration of the Kinova arm. Therefore, the solution to this version of the problem consists of the robot moving only its arm to reach for the target and then holding that configuration. No objective function is provided, so this is a feasibility problem.

Baseline - In this version, the solver is able to change the configuration of the whole-body of the robot. This is similar to the *baseline* formulation used in previous sections, but the feet locations are now decision variables. The torques and contact forces over time are minimized.

Proposed - In this version, the solver is also able to change the configuration of the whole-body of the robot. This is similar to the *proposed* formulation used in previous sections, but the foot locations are now decision variables. The magnitude of the SUF over time is maximized.

Next, we show the results of these slightly different problem versions. The robot configurations of each version are shown in Figure 31. The feet position of each solution are shown in the plot of Figure 32. After solving each version of the problem, we then computed the magnitude of the SUF at the gripper. The computed SUF spheres are also shown in Figure 31, and labelled with their respective magnitude.



(a) Nominal (b) Baseline (c) Proposed

Figure 31: Robot configurations resultant from the extended NLP formulation, which allows the solver to optimise feet locations. The radius of the sphere corresponds to the magnitude of the SUF.

The first thing we can observe from these results is that the solver did take the liberty of optimising the feet locations. This verifies that the solver is able to deal with our

---

[8] Feet positions are represented explicitly by the new *xy*-coordinates, but also implicitly by the forward kinematics of the robot's configuration $\boldsymbol{q}$.

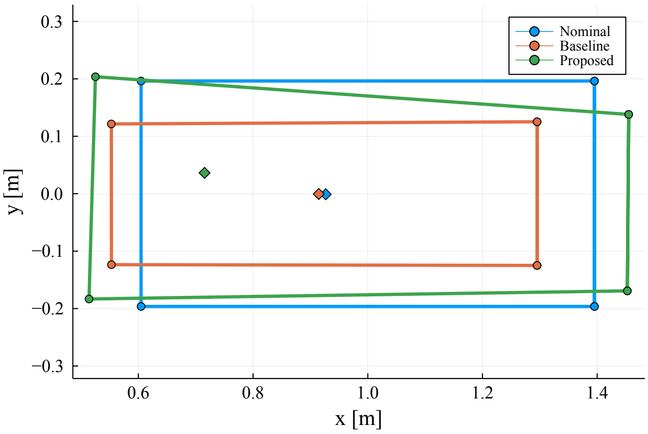


Figure 32: Feet locations, support polygons, and projected centres of mass of the trajectories computed using the *nominal* (blue), *baseline* (orange), and *proposed* (green) versions of the optimisation problem. Feet positions are shown using circles; the line segments connecting the circles form the support polygons; and the diamonds denote the center of mass positions projected onto the support polygons.

more-complex NLP formulation, albeit taking longer than the previous formulation. Secondly, the plot of the feet locations clearly shows that the trajectory optimised with the baseline approach has a smaller support polygon than the trajectory optimised with the proposed approach. This is relevant because we know that the support polygon is a good representation for the region where the CoM projection can lie for achieving static balance. However, the support polygon is only an approximation since it does not take into account robot capabilities (torques and friction at the contacts); e.g., it is not guaranteed that the robot has enough actuation power to maintain a pose whose CoM projection lies on one of the corners of the support polygon. This leads to our next observation: while the baseline approach keeps the CoM position close to the nominal version, the CoM of the trajectory optimised with the proposed approach lies further from the centre of the support polygon. Our metric converges to this configuration because it takes into account the robot capabilities, and is able to find a more stable and robust pose, even though its CoM projection does not lie at the centre of the support polygon. Finally, we can see that the worst-case disturbance scenario that the robot can resist is better for the proposed approach, and worse for the nominal scenario. As labelled in Figure 31, the magnitude of the SUF for the nominal, baseline, and proposed approaches were 58 N, 57 N, and 76 N, respectively.

The above results show that our formulation is able to optimise feet locations, and that our metric is able to guide the solver to solutions that have increased capabilities of resisting external forces—at least in theory, that is. However, when we deploy our method on the robot, there are always practical subtleties that may affect how the robot behaves, such as signal delay or the type of controller being used. With that in mind, we carried out an experiment to assess the actual force-rejection capabilities of the real robot. Next, we explain the experiment setup and then analyse the results.

In this experiment, we executed the trajectory optimised with each approach on the real robot and, for each case, we disturbed the robot by pulling its end-effector from three different angles. A summary of the events that occurred for each angle/trajectory is given below:

Summary of Angle #1

Nominal - First, the end-effector deviated from its set point. Then, the right hind and right front feet started to slide. However, the robot did not fall over.

Baseline - First, the end-effector deviated from its set point. Then, the right front foot started to slide. Next, the right hind foot lost contact. Finally, the robot toppled to its left side.

Proposed - First, the end-effector deviated from its set point. Then, the right front foot started to slide. Next, both left and right hind feet slipped, but only slightly. The robot did not fall over.

Summary of Angle #2

Nominal - The end-effector deviated from its set point and the hind feet started to slip at approximately the same time. The robot would have fallen down, but the harness prevented it from collapsing on the floor.

Baseline - The end-effector deviated from its set point. The feet did not move and the robot did not fall over.

Proposed - First, the end-effector deviated from its set point. Then, the hind feet started to move. If the harness had not supported it, the robot would have fallen down.

Summary of Angle #3

Nominal - The end-effector deviated from its set point. The left hind foot moved. The robot did not fall.

Baseline - The end-effector deviated from its set point. The left front foot moved and the left hind foot lost contact. Finally, the robot toppled over to its right side.

Proposed - The end-effector deviated from its set point. The left front foot moved and the left hind foot lost contact. Finally, the robot toppled over to its right side.

When we carried out the experiment, we used a force gauge between the end-effector of the robot and the source of the disturbance in order to measure—and capture on video—the magnitude of the force being applied to the robot's end-effector.

In Figure 33, we show the state of the experiment at the instant when the force gauge indicated the greatest force magnitude, for each trajectory and for each angle. The label in the bottom right corner of each snapshot indicates the magnitude of the value measured by the force gauge. For each row, i.e., for each trajectory, the label highlighted in red corresponds to the magnitude of the SUF found experimentally for the three distinct angles used for the experiment. The way to interpret these results is to take the smallest SUF found experimentally for each trajectory (highlighted in red and signifying the worst case perturbation), and then compare those magnitudes to find the trajectory with largest worst case SUF (indicating the highest robustness).

As highlighted in Figure 33, the proposed trajectory exhibited the highest SUF (60 N). In percentage terms, the SUF of the proposed trajectory was approximately 33 %

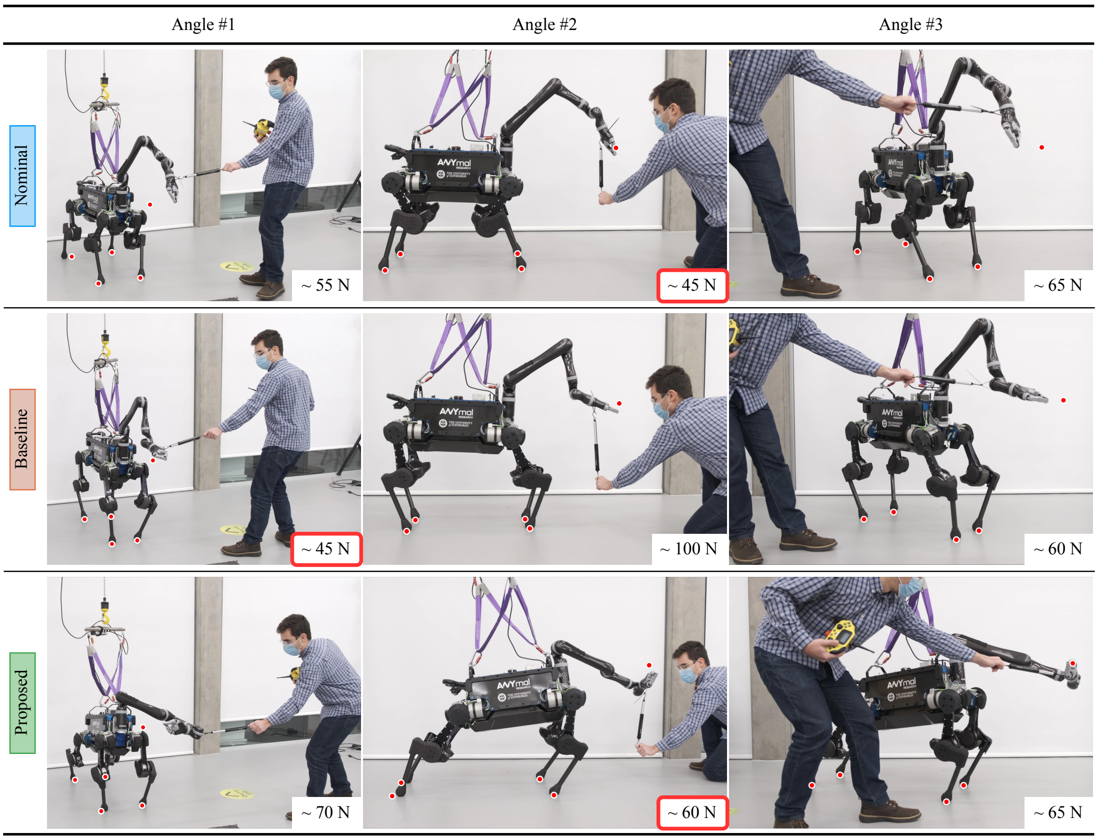


Figure 33: Snapshots of the experiment in which we disturbed the robot by pulling its end-effector. Pictures on the same row correspond to the same trajectory, whereas pictures under the same column share the same disturbance angle. The snapshots were taken from the video footage collected during the experiment, at the time the force gauge indicated the greatest force magnitude—which is shown in the label in the bottom right corner of each picture. Labels highlighted in red are the worst-case scenario for each row (out of the three angles shown). The five red circles on each snapshot represent the initial positions of the four feet and of the end-effector. Video: `https://youtu.be/tUXQUqLneTE`.

better than both nominal and baseline trajectories, which is a significant improvement. Moreover, the SUF magnitudes found experimentally were 21 % to 22 % smaller than the SUF predicted originally (the ones shown in Figure 31). We understand this is due to model mismatch and unaccounted factors when we deploy the trajectory on the real robot. But the fact that the percentage decrease is similar for each trajectory is reassuring, as it tells us that the unaccounted factors of the real robot affected the SUF equally, and the proposed trajectory performed better than the nominal and baseline trajectories as we had predicted in relative terms.

5.6 CONCLUSION

In this chapter, we presented a framework for planning whole-body loco-manipulation trajectories robust to external disturbances. We integrated our framework with existing software stacks to enable easy switching between teleoperated and autonomous modes. We demonstrated the capabilities of that integration by having a human operator remotely control the robot via a joystick in a mock-up rig of an industrial site, and by having the robot autonomously plan complex and rich whole-body motions for real-world tasks within that setting, such as turning a hand wheel, pulling a lever, opening a gate whilst on a ramp, and lifting a heavy bucket by pulling a rope. We also carried out a wide range of experiments to test the reliability of the full system, analyse the SUF of existing trajectories, optimise the robustness of trajectories (including those involving making and breaking of contacts). Finally, we carried out an initial investigation on the possibility of using our framework for the purpose of optimising feet positions.

In the next chapter, we are going to discuss a topic that sits at the core of our framework, which can have (and does have!) a big impact on solver performance: the constraints used for evaluating dynamics defects in direct transcription problems.

6

INVERSE DYNAMICS AS AN ALTERNATIVE TO FORWARD DYNAMICS IN DIRECT TRANSCRIPTION FORMULATIONS

Benchmarks of state-of-the-art rigid-body dynamics libraries report better performance solving the inverse dynamics problem than the forward alternative. Those benchmarks encouraged us to question whether that computational advantage would translate to direct transcription, where calculating rigid-body dynamics and their derivatives accounts for a significant share of computation time.

In this chapter, we implement both approaches for enforcing the system dynamics in our trajectory optimisation framework. We evaluate the performance of each approach for systems of varying complexity, for domains with rigid contacts. Our tests reveal that formulations using inverse dynamics converge faster, require fewer iterations, and are more robust to coarse problem discretisation. These results indicate that inverse dynamics should be preferred to enforce the nonlinear system dynamics in simultaneous methods, such as direct transcription.

6.1 INTRODUCTION

Direct transcription [5] is an effective approach to formulate and solve trajectory optimisation problems. It works by converting the original trajectory optimisation problem (which is *continuous* in time) into a numerical optimisation problem that is *discrete* in time, and which in turn can be solved using an off-the-shelf nonlinear programming (NLP) solver. First, the trajectory is divided into segments and then, at the beginning of each segment, the system state and control inputs are explicitly discretized—these are the decision variables of the optimisation problem. Due to this discretisation approach, direct transcription falls under the class of *simultaneous* methods. Finally, a set of mathematical constraints is defined to enforce boundary and path constraints, e.g., initial and final conditions, or intermediate goals. In dynamic trajectory optimisation, there exists a specific set of constraints dedicated to enforce the equations of motion of the system, the so-called *defect constraints*. This chapter discusses different ways of defining these constraints, as well as their implications.

The dynamics defects are one of the most important constraints in optimisation problems when planning highly dynamic motions for complex systems, such as legged robots. Satisfaction of these constraints ensures that the computed motion is reliable and physically consistent with the nonlinear dynamics of the system. The dynamics defect constraints are usually at the very core of optimal control formulations, and require computing rigid-body dynamics and their derivatives—which account for a significant portion of the optimisation computation time. Therefore, it is of utmost importance to use an algorithm that allows to compute the dynamics of the system reliably, while achieving low computational time.

In the study of the dynamics of open-chain robots, the *forward dynamics* problem determines the joint accelerations resultant from a given set of joint forces and torques applied at a given state. On the other hand, the *inverse dynamics* problem determines the joint torques and forces required to meet some desired joint accelerations at a given state. In trajectory optimisation, most direct formulations use forward dynamics to enforce dynamical consistency [57]. However, benchmarks have shown that most dynamics libraries solve the *inverse dynamics* problem (e.g., with the Recursive Newton-Euler Algorithm) faster than the *forward dynamics* problem (e.g., with the Articulated Body Algorithm) [44, 53]. For example, for the humanoid robot TALOS [62], the library Pinocchio [14] solves the inverse dynamics problem in just 4 µs, while the forward dynamics problem takes 10 µs. These differences in performance motivated us to question whether the computational advantage of inverse dynamics would translate to direct transcription—where the dynamics problem needs to be solved several times while computing the defect constraints. Moreover, there is biological evidence suggesting that inverse dynamics is employed by the nervous system to generate feedforward commands [60], while other studies support the existence of a forward model [51]—which increased our interest in this topic.

In this chapter, we present a trajectory optimisation framework for domains with rigid contacts, using a direct transcription approach. Particularly, our formulation allows defining dynamics defect constraints employing either forward dynamics or inverse dynamics. We defined a set of evaluation tasks across different classes of robot platforms, including fixed- and floating-base systems, with point and surface contacts. Our results showed that inverse dynamics leads to significant improvements in computational performance when compared to forward dynamics—supporting our initial hypothesis.

6.2 RELATED WORK

RigidBodyDynamics.jl (RBD.jl) [43], RBDL [25], Pinocchio [14], and RobCoGen [31] are all state-of-the-art software implementations of key rigid-body dynamics algorithms. Recently, Neuman *et al.* [53] benchmarked these libraries and revealed interesting trends. One such trend is that implementations of inverse dynamics algorithms have faster runtimes than forward dynamics.[1] Koolen and Deits [44] also compared RBD.jl with RBDL, and their results showed that solving inverse dynamics was at least two times faster than solving forward dynamics for the humanoid robot Atlas. Both of these studies only consider computation time of rigid-body dynamics; they do not provide insight into how these algorithms perform when used in trajectory optimisation.

Lee *et al.* [66] have proposed Newton and quasi-Newton algorithms to optimise motions for serial-chain and closed-chain mechanisms using inverse dynamics. However, they used relatively simple mechanisms for which analytic derivatives can be obtained. In our work, we are interested in dynamic motions of complex mechanisms in domains with contact, for which the derivation of analytic derivatives is an error-prone process, involving significant effort.

In the same spirit, Erez and Todorov [22] generated a running gait for a humanoid based on inverse dynamics under external contacts. This method allowed them to formulate an *unconstrained* optimisation where all contact states can be considered equally, contact timings and locations are optimised, and reaction forces are computed using a smooth and invertible contact model [71] with convex optimisation. However, their approach requires "helper forces", as well as tuning of contact smoothness and of the penalty parameters on the helper forces to achieve reasonable-looking behaviour. In contrast, our approach does not require helper forces or any tuning whatsoever; we consider contact forces as decision variables and model contacts rigidly. Another difference is that we formulate a *constrained* optimisation problem and enforce the nonlinear system dynamics with hard constraints, which results in high-fidelity motions. This is especially important for deployment on real hardware, where dynamic consistency and realism are imperative. The main focus of this chapter is not the contact problem, and we assume contact locations and contact times are known *a priori*.

---

[1] RobCoGen is an exception to this observation as it implements a hybrid dynamics solver which has a higher computational cost, and is significantly different from the implementations used by the other libraries.

Finally, to the best of our knowledge, there is no current work directly comparing inverse dynamics against forward dynamics in the context of direct methods. Posa *et al.* [59] also identified that a formal comparison is important, but missing so far. They argued that one of the reasons for this was that the field had not yet agreed upon a set of canonical and hard problems. In this chapter, we tackle this issue, and compare the two approaches on robots of different complexity on a set of dynamic tasks.

The main contributions of this work are:

1. A direct transcription formulation that uses *inverse dynamics* to enforce physical consistency, for constrained trajectory optimisation in domains with rigid contacts.
2. Evaluation of the performance of direct transcription formulations using either forward or inverse dynamics, for different classes of robot platforms: a fixed-base manipulator, a quadruped, and a humanoid.
3. Comparison of performance for different linear solvers, and across strategies to handle the barrier parameter of the interior point optimisation algorithm.

We validated our trajectories in full-physics simulation and with hardware experiments. We also open-sourced a version of our framework for fixed-base robots, TORA.jl [27].

6.3 TRAJECTORY OPTIMISATION

6.3.1 *Robot Model Formulation*

We formulate the model of a legged robot as a free-floating base B to which limbs are attached. The motion of the system can be described with respect to (w.r.t.) a fixed inertial frame I. We represent the position of the free-floating base w.r.t. the inertial frame, and expressed in the inertial frame, as ${}_{I}\boldsymbol{r}_{IB} \in \mathbb{R}^3$; and the orientation of the base as $\boldsymbol{\psi}_{IB} \in \overline{\mathbb{R}}^3$, using modified rodrigues parameters (MRP) [34, 69]. The joint angles describing the configuration of the limbs of the robot (legs or arms) are stacked in a vector $\boldsymbol{q}_j \in \mathbb{R}^{n_j}$, where n_j is the number of actuated joints. The generalized coordinates vector $\boldsymbol{q}$ and the generalized velocities vector $\boldsymbol{v}$ of this floating-base system may therefore be written as

$$\boldsymbol{q} = \begin{bmatrix} {}_I\boldsymbol{r}_{IB} \\ \boldsymbol{\psi}_{IB} \\ \boldsymbol{q}_j \end{bmatrix} \in \mathbb{R}^3 \times \overline{\mathbb{R}}^3 \times \mathbb{R}^{n_j}, \quad \boldsymbol{v} = \begin{bmatrix} \boldsymbol{\nu}_B \\ \dot{\boldsymbol{q}}_j \end{bmatrix} \in \mathbb{R}^{n_v}, \tag{79}$$

where the twist $\boldsymbol{\nu}_B = \begin{bmatrix} {}_I\boldsymbol{v}_B & {}_B\boldsymbol{\omega}_{IB} \end{bmatrix}^\top \in \mathbb{R}^6$ encodes the linear and angular velocities of the base B w.r.t. the inertial frame expressed in the I and B frames, and $n_v = 6 + n_j$.

For fixed-base manipulators, the generalized vectors of coordinates and velocities can be simplified to $\boldsymbol{q} = \boldsymbol{q}_j \in \mathbb{R}^{n_j}$ and $\boldsymbol{v} = \dot{\boldsymbol{q}}_j \in \mathbb{R}^{n_j}$, due to the absence of a free-floating base.

6.3.2 *Problem Formulation*

We tackle the motion planning problem using trajectory optimisation; more specifically, using a *direct transcription* approach. The original problem is *continuous* in time, so we start by converting it into a numerical optimisation problem that is *discrete* in time. For that, we divide the trajectory into N equally spaced segments, $t_I = t_1 < \cdots < t_M = t_F$, where t_I and t_F are the start and final instants, respectively. This division results in $M = N + 1$ discrete *mesh points*, for each of which we explicitly discretize the states of the system, as well as the control inputs. Let $x_k \equiv x(t_k)$ and $u_k \equiv u(t_k)$ be the values of the state and control variables at the k-th mesh point. We treat $x_k \triangleq \{\boldsymbol{q}_k, \boldsymbol{v}_k\}$ and $u_k \triangleq \{\boldsymbol{\tau}_k, \boldsymbol{\lambda}_k\}$ as a set of NLP variables, and formulate the trajectory optimisation problem as

$$\text{find } \boldsymbol{\xi} \quad \text{s.t. } x_{k+1} = f(x_k, u_k), \;\; x_k \in \mathcal{X}, \;\; u_k \in \mathcal{U}, \tag{80}$$

where $\boldsymbol{\xi}$ is the vector of decision variables, $x_{k+1} = f(x_k, u_k)$ is the state transition function incorporating the nonlinear system dynamics, and $\mathcal{X}$ and $\mathcal{U}$ are sets of feasible states and control inputs enforced by a set of equality and inequality constraints. The decision variables vector $\boldsymbol{\xi}$ results from aggregating the generalized coordinates, generalized velocities, joint torques, and contact forces at every[2] mesh point, i.e.,

$$\boldsymbol{\xi} \triangleq \{\boldsymbol{q}_1, \boldsymbol{v}_1, \boldsymbol{\tau}_1, \boldsymbol{\lambda}_1, \cdots, \boldsymbol{q}_N, \boldsymbol{v}_N, \boldsymbol{\tau}_N, \boldsymbol{\lambda}_N, \boldsymbol{q}_M, \boldsymbol{v}_M\}. \tag{81}$$

Similarly to Winkler *et al.* [76] and differently to Erez and Todorov [22], we transcribe the problem by only making use of hard constraints; and satisfaction of those constraints is a necessary requirement for the computed motions to be physically feasible and to complete the task successfully. This design decision is motivated by the fact that considering a cost function requires expert knowledge to carefully tune the weighting parameters that control the trade-off between different objective terms. Optimising an objective function also requires additional iterations and computation time. Nonetheless, for the sake of completion, one of the experiments we present later in this chapter does include and discuss the minimisation of a cost function.

For tasks where the robot makes or breaks contacts with the environment, we assume contact locations and contact timings are known *a priori*. This assumption allows us to enforce zero contact forces for mesh points where the robot is not in contact with the environment, and therefore our formulation does not require any actual complementarity constraints. On the other hand, such assumption depends on pre-determined contact sequences specified either by a human or by a contact planner (such as [63, 73, 76]).

---

[2] The control inputs at the final state need not be discretized.

6.3.3 *Problem Constraints*

6.3.3.1 *Bounds on the decision variables*

We constrain the joint positions, velocities, and torques to be within their corresponding lower and upper bounds.

6.3.3.2 *Initial and final joint velocities*

We enforce the initial and final velocities of every joint to be zero: $\boldsymbol{v}_1 = \boldsymbol{v}_M = \mathbf{0}$.

6.3.3.3 *End-effector pose*

We enforce end-effector poses with $f^{\text{fk}}(\boldsymbol{q}_k, i) = \boldsymbol{p}_i$, where $f^{\text{fk}}(\cdot)$ is the forward kinematics function, i refers to the i-th end-effector of the robot, and $\boldsymbol{p}_i \in SE(3)$ is the desired pose.

6.3.3.4 *Contact forces*

For mesh points where the robot is *not* in contact with the environment, we enforce the contact forces at the respective contact points to be zero: $\boldsymbol{\lambda}_k = \mathbf{0}$.

6.3.3.5 *Friction constraints*

We model friction at the contacts with linearized friction cones, in the same way as [13].

6.3.3.6 *System dynamics*

We enforce nonlinear whole-body dynamics, $\dot{x} = f(x, u)$, with *defect* constraints. The approach used to define these constraints is the main subject of this chapter, and the next section explains this in detail.

6.4 SYSTEM DYNAMICS

The equations of motion for a floating-base robot that interacts with its environment can be written as

$$\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{v}} + \boldsymbol{h}(\boldsymbol{q}, \boldsymbol{v}) = \boldsymbol{S}^\top \boldsymbol{\tau} + \boldsymbol{J}_s^\top(\boldsymbol{q})\boldsymbol{\lambda}, \tag{82}$$

where $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n_v \times n_v}$ is the mass matrix, and $\boldsymbol{h}(\boldsymbol{q}, \boldsymbol{v}) \in \mathbb{R}^{n_v}$ is the vector of Coriolis, centrifugal, and gravity terms. On the right-hand side of the equation, $\boldsymbol{\tau} \in \mathbb{R}^{n_\tau}$ is the vector of joint torques commanded to the system, and the selection matrix $\boldsymbol{S} =$

$[\mathbf{0}_{n_\tau \times (n_v - n_\tau)} \quad \mathbb{I}_{n_\tau \times n_\tau}]$ selects which degrees of freedom (DoF) are actuated. We consider that all limb joints are actuated, thus $n_\tau = n_j$. The vector $\boldsymbol{\lambda} \in \mathbb{R}^{n_s}$ denotes the forces and torques experienced at the contact points, with n_s being the total dimensionality of all contact wrenches. The support Jacobian $\boldsymbol{J}_s \in \mathbb{R}^{n_s \times n_v}$ maps the contact wrenches $\boldsymbol{\lambda}$ to joint-space torques, and it is obtained by stacking the Jacobians which relate generalized velocities to limb end-effector motion as $\boldsymbol{J}_s = [\boldsymbol{J}_{C_1}^\top \quad \cdots \quad \boldsymbol{J}_{C_{n_c}}^\top]^\top$, with n_c being the number of limbs in contact. For fixed-base robots that are not subject to contact forces, we can simplify the equations of motion to $\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{v}} + \boldsymbol{h}(\boldsymbol{q}, \boldsymbol{v}) = \boldsymbol{\tau}$.

In order to enforce the equations of motion of nonlinear systems, we define a set of equality constraints within our framework, the so-called *defect constraints*. Usually, these constraints are defined using a forward dynamics algorithm, but in this chapter we argue that using inverse dynamics can be more computationally advantageous.

The standard problem of forward dynamics computes the joint accelerations resultant from commanding torques and applying forces to the robot at a given state, i.e.,

$$\dot{\boldsymbol{v}}_k^* = f^{\text{fd}}(\boldsymbol{q}_k, \boldsymbol{v}_k, \boldsymbol{\tau}_k, \boldsymbol{\lambda}_k), \tag{83}$$

where $f^{\text{fd}}(\cdot)$ is the function that solves forward dynamics. The asterisk $(\cdot)^*$ denotes intermediately computed values, whereas terms without an asterisk are NLP variables. Using the semi-implicit Euler method as the integration scheme and $h = (t_F - t_I)/N$ as the integration time step, we can compute the state of the robot after h seconds. First, we integrate $\dot{\boldsymbol{v}}_k^*$ to compute the next generalized velocities $\boldsymbol{v}_{k+1}^* = \boldsymbol{v}_k + h\,\dot{\boldsymbol{v}}_k^*$. Then, we can compute the time derivative of the generalized coordinates, $\dot{\boldsymbol{q}}_{k+1}^*$, from those velocities, $\boldsymbol{v}_{k+1}^*$. In turn, we integrate that time derivative to compute the next generalized coordinates, $\boldsymbol{q}_{k+1}^* = \boldsymbol{q}_k + h\,\dot{\boldsymbol{q}}_{k+1}^*$. After these calculations, we end up with two different values for the state of the system at mesh point $k+1$: one from the discretized NLP variables, and another computed as a result of the controls applied to the system at mesh point k. To enforce dynamical consistency, we define the defect constraints as

$$\boldsymbol{q}_{k+1}^* - \boldsymbol{q}_{k+1} = \mathbf{0} \quad \text{and} \quad \boldsymbol{v}_{k+1}^* - \boldsymbol{v}_{k+1} = \mathbf{0}. \tag{84}$$

However, there is an alternative way to enforce dynamical consistency: with inverse dynamics. In contrast to (83), inverse dynamics computes the joint torques and forces required to meet desired joint accelerations at a given state, i.e.,

$$\boldsymbol{\tau}_k^* = f^{\text{id}}(\boldsymbol{q}_k, \boldsymbol{v}_k, \dot{\boldsymbol{v}}_k^*, \boldsymbol{\lambda}_k), \tag{85}$$

where $f^{\text{id}}(\cdot)$ is the function that solves the inverse dynamics problem, and the desired joint accelerations can be calculated implicitly with $\dot{\boldsymbol{v}}_k^* = (\boldsymbol{v}_{k+1} - \boldsymbol{v}_k)/h$. Similarly to the forward dynamics case, we compute $\dot{\boldsymbol{q}}_{k+1}^*$ from $\boldsymbol{v}_{k+1}$, and integrate it to compute the next generalized coordinates $\boldsymbol{q}_{k+1}^*$. And finally, we define the dynamics defect constraints as

$$\boldsymbol{q}_{k+1}^* - \boldsymbol{q}_{k+1} = \mathbf{0} \quad \text{and} \quad \boldsymbol{\tau}_k^* - \boldsymbol{\tau}_k = \mathbf{0}. \tag{86}$$

Notice that the main difference between equations (84) and (86) is that forward dynamics enforces consistency of the generalized velocities, whereas inverse dynamics enforces consistency of joint torques commanded to the system.

The main subject of this chapter revolves around the two formulations explained above to enforce the nonlinear system dynamics: *forward dynamics* vs. *inverse dynamics*. We developed our framework with both options in mind, and we are able to easily toggle between one approach and the other, which was particularly useful for our experiments.

6.5 EXPERIMENTS AND RESULTS

This section is organized into four subsections:

A. Compares the computation time and number of solver iterations required to find locally-optimal solutions;

B. Evaluates the robustness of each approach as problem discretisation gets more coarse (larger time steps);

C. Analyses the performance of each formulation for the minimisation of a cost function; and finally,

D. Shows hardware validation of the planned motions.

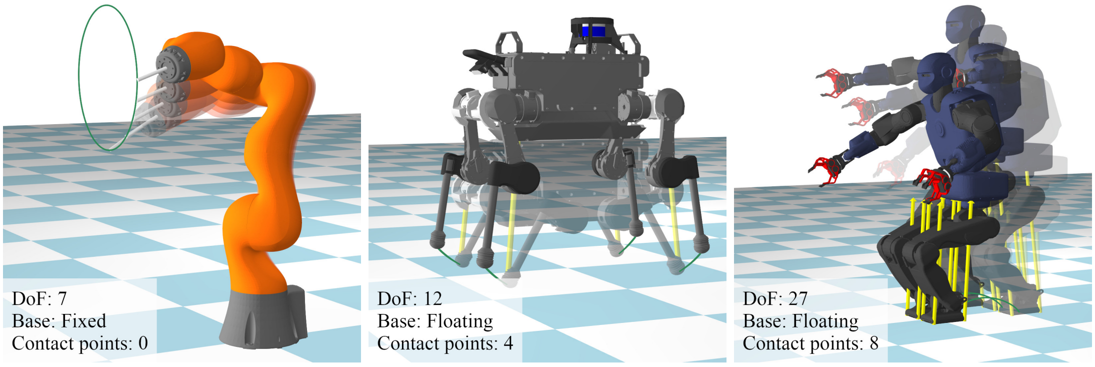
All evaluations were carried out in a single-threaded process on an Intel i7-6700K CPU with clock frequency fixed at 4.0 GHz, and 32 GB 2133 MHz memory. The framework we propose has been implemented in Julia [6], using the rigid-body dynamics library RBD.jl [43], and the optimisation library Knitro [11]. To solve the formulated NLP problems, we used the interior-point method of Waltz *et al.* [75].

6.5.1 *Evaluation of Convergence*

In order to evaluate and compare forward dynamics against inverse dynamics in the context of direct transcription, we used our framework to specify tasks in the form of numerical optimisation problems for different types of robots: a manipulator, a quadruped, and a humanoid. Those robots were selected as they allow us to evaluate the formulations for distinct features: fixed- and floating-base systems, single-point and surface contacts, and low and high dimensionality. For each task on each robot, we solved the optimisation problem twice: first defining the defect constraints with forward dynamics, and then with inverse dynamics. The *only* changing factor was the toggling between forward and inverse dynamics for the definition of the defect constraints; every other aspect of the formulation was kept unchanged.

The performance of general NLP solvers is greatly affected by the linear solver used for solving the linear systems of equations of the problem. For this reason, we tested different state-of-the-art linear solvers exhaustively. For interior-point methods, another important factor that affects performance is the update strategy of the barrier parameter. Therefore, for all of our evaluations, we tested the different strategies available within the Knitro [11] library exhaustively.

In the remainder of this subsection, we present the task specifications for each robot and indicate all the parameters for reproducibility. Then, we present the results we obtained for each task, which evaluated the solver's performance in terms of computation time and number of iterations taken by the solver until a locally-optimal solution was found.



(a) Manipulator (b) Quadruped (c) Humanoid

Figure 34: Left to right: KUKA iiwa tracing a circular path, ANYmal jumping in-place, and TALOS jumping forward.

6.5.1.1 *Manipulator*

We evaluated the different formulations using a fixed-base robot arm with seven DoF (shown in Figure 34a). We specified the end-effector to trace a circular path given by $[0.5, 0.2\cos\theta, 0.8 + 0.2\sin\theta] \forall\theta \in [0, 2\pi]$. The total duration was set to 2.0 s and the trajectory was discretized at 150 Hz, resulting in a total of 301 mesh points.

6.5.1.2 *Quadruped*

The quadruped robot we used is shown in Figure 34b. This system is more complex than the manipulator due to its floating-base, more DoF (three motors per leg), and because it needs to handle contact forces. We defined a jumping task by enforcing the contact forces to be zero for a short period of time. The trajectory was discretized at 100 Hz, the total duration of the motion was 2.0 s, and the interval specified for the flight-phase was $[1.0, 1.2]$ s. We did not constrain feet positions during the flight-phase, which allowed the solver to converge to a solution where the feet swing most naturally according to the system dynamics.

6.5.1.3 *Humanoid*

Finally, we considered the humanoid robot shown in Figure 34c. This robot is more complex than the quadruped because it has 27 DoF[3] (seven per arm, six per leg, and one at the torso), and its feet cannot be simplified to single-point contacts. We also defined a jumping task for this robot. The motion duration was 1.2 s, discretized at 125 Hz, and the interval for the flight-phase was $[0.5, 0.8]$ s.

For all the tasks, the initial guess was a fixed standing configuration and zero velocities, torques, and contact forces.

---

[3] The real robot has more DoF: grippers, neck, and one more DoF at the torso. For simplicity, we assumed those joints were fixed to zero.

The results of the experiments on these robots are shown in Table 17, where smaller numbers indicate better performance. The rows of the table are grouped according to robot, dynamics, and linear solver. Each row shows the time taken to solve the optimisation problem for each barrier update strategy, as well as the total number of iterations (within parenthesis). The last column shows the time spent on each iteration, averaged over all the update strategies.

Table 17: Computation time (in seconds) and number of iterations (within parenthesis) for each robot. Truncated values ('—') denote cases that did not converge. The best computation time for each dynamics and each robot is highlighted in bold.

		Linear Solver	Barrier strategy (`bar_murule`)			Average time per iteration (s)
			`adaptiv`	`dampmpc`	`quality`	
KUKA iiwa	Fwd D	MA27	**0.40** (5)	0.49 (6)	0.43 (5)	0.08 ± 0.002
		MA57	0.41 (5)	0.48 (6)	0.42 (5)	0.08 ± 0.001
		MA97	0.46 (5)	0.56 (6)	0.50 (5)	0.09 ± 0.002
	Inv D	MA27	**0.20** (4)	0.26 (5)	0.23 (4)	0.05 ± 0.002
		MA57	0.22 (4)	0.28 (5)	0.24 (4)	0.05 ± 0.001
		MA97	0.27 (4)	0.33 (5)	0.29 (4)	0.07 ± 0.002
ANYmal B	Fwd D	MA27	2.26 (7)	2.80 (9)	2.34 (7)	0.31 ± 0.021
		MA57	2.80(10)	2.49 (9)	32.13(99)	0.29 ± 0.020
		MA97	**2.21** (7)	2.76 (9)	2.26 (7)	0.31 ± 0.009
	Inv D	MA27	2.99(13)	2.60(11)	2.18 (9)	0.23 ± 0.005
		MA57	2.90(13)	2.54(11)	**2.10** (9)	0.22 ± 0.004
		MA97	3.21(13)	2.81(11)	2.37 (9)	0.25 ± 0.007
TALOS	Fwd D	MA27	—	—	14.82(15)	0.94 ± 0.115
		MA57	—	13.47(19)	44.23(66)	0.68 ± 0.020
		MA97	13.42(18)	**11.48**(15)	12.92(16)	0.76 ± 0.026
	Inv D	MA27	8.38(15)	8.02(13)	38.59(70)	0.57 ± 0.035
		MA57	7.36(15)	**6.59**(13)	36.88(73)	0.50 ± 0.014
		MA97	7.43(15)	6.61(13)	41.84(82)	0.50 ± 0.012

In general, we can see that the computation time depends mostly on the complexity of the system, regardless of linear solver or barrier update strategy; i.e., solving the manipulator task was faster than solving the quadruped task, which in turn was faster than the humanoid task. More importantly, for each robot and given the same choice of linear solver and update strategy, the computation time of inverse dynamics was better than forward dynamics. We can also see that the number of iterations required

---

[3] This table (and future tables) show the minimum time value observed over 10 trials. Reporting the minimum time is more reliable than the median or the mean, since all measured noise is positive, as explained in [15].

to solve the problem did not change significantly (apart from a few exceptions). This indicates that the difficulty of the problem itself did not change with the different dynamics defects; it just took longer to solve using forward dynamics—as supported by the information in the last column of the table.

6.5.2 *Robustness to Coarser Problem Discretisation*

In the next experiment, we compare the ability of each formulation to handle trajectories discretized using fewer mesh points. We defined the same quadruped jumping task repeatedly, but transcribed it with different resolutions. First, we divided the trajectory into equally spaced segments with a time step of $h = 0.01$; we solved the optimisation problem and took the resulting trajectory as our baseline. Then, we incrementally changed h, making the problem more coarse each time, and compared the obtained trajectories against the baseline. The problems were initialized with a nominal configuration repeated for each point, and zero velocities, torques and contact forces. The results of this experiment are shown in Figure 35 and Table 18.

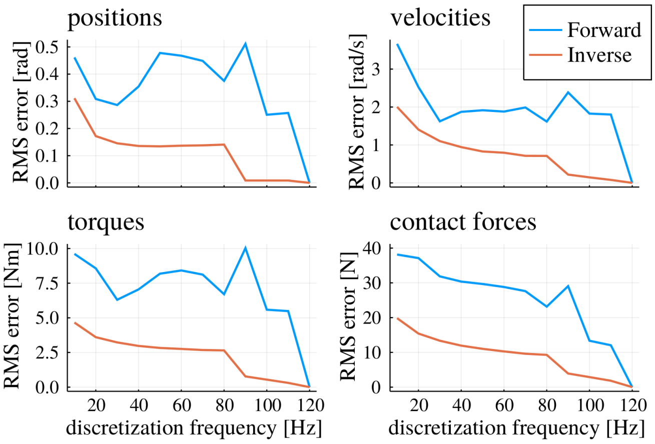


Figure 35: Root-mean-square error (RMSE) of joint positions, velocities, torques, and contact forces of each formulation for different discretisations, using a baseline of 120 Hz.

The plots in Figure 35 show that the solutions deviate more from the baseline as the number of mesh points used to discretize the problem decreases (in the x-axis, from right to left). But more importantly, the plots reveal that the rate at which deviation occurs is significantly different depending on the formulation. We can see that the root-mean-square error (RMSE) of the formulation using inverse dynamics is significantly lower than that of the forward.

Table 18 shows the computation time (in seconds) and the number of iterations required to solve the quadruped task using different discretisation. We can see that the time required to solve the problem using inverse dynamics follows a clear pattern: it decreases as the problem gets more coarse; and the same goes for the number of iterations. In contrast, a pattern does not seem to exist for forward dynamics.

The results shown in Figure 35 and Table 18 provide strong evidence that defining the defect constraints with inverse dynamics is the approach more robust to different problem discretisation, both in terms of deviation from realistic solutions and in terms of computation performance.

Table 18: Computation time and number of iterations required by different problem discretisations, for the quadruped jump.

Frequency	Forward Dyn.		Inverse Dyn.	
	Time (s)	Iter.	Time (s)	Iter.
100 Hz	3.289	10	3.295	13
90 Hz	3.916	14	2.774	13
80 Hz	5.227	21	1.821	9
70 Hz	4.062	19	1.344	8
60 Hz	1.518	9	1.193	8
50 Hz	2.226	16	0.983	8
40 Hz	1.099	9	0.785	8
30 Hz	0.731	8	0.534	7
20 Hz	0.433	7	0.354	7

6.5.3 *Optimisation with an Objective Function*

Thus far, we have analysed the trajectory optimisation performance for feasibility problems. However, in optimisation, it is common to define a cost function to be minimised (or a value function to be maximised). In this next experiment, we evaluate the performance of our formulation when a cost function is considered. We minimise the actuator torques and ground-reaction contact forces with $\min_{\xi} \sum_{k=1}^{M-1} \frac{\boldsymbol{\tau}_k^\top \boldsymbol{\tau}_k + \boldsymbol{\lambda}_k^\top \boldsymbol{\lambda}_k}{M-1}$. We tested this cost function on the quadruped jump task, with the MA57 linear solver and `adaptive` barrier parameter update strategy.

Both formulations converged to very similar solutions: the RMSE between the two trajectories was 0.038. The final objective value was $3.567\,801 \times 10^4$ and $3.567\,804 \times 10^4$ for forward and inverse dynamics, respectively. Despite converging to similar solutions, the formulation employing inverse dynamics finished in 6.208 s, showing better performance than the formulation using forward dynamics, which took 14.570 s. The time in seconds corresponds to the minimum value measured over a total of 10 samples.

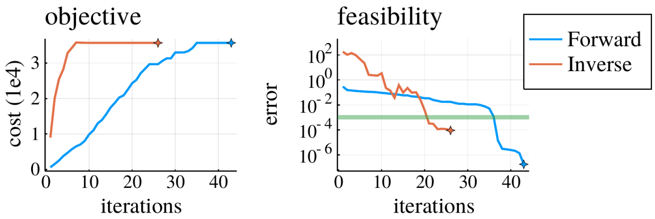


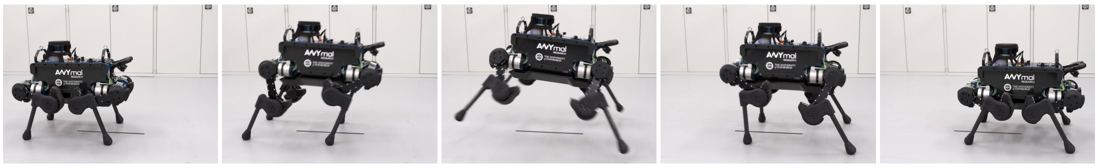
Figure 36: Evolution of the cost and the feasibility error during convergence. The faint-green line at $y = 10^{-3}$ denotes the tolerance under which we consider a problem to be feasible.

Figure 36 shows the evolution of the cost and feasibility error throughout the optimisation. The star-shaped marker denotes the point at which the local minimum of the problem was found. In the left plot, we can see that inverse dynamics reached values close to the optimal cost much earlier than forward dynamics. In the right plot, we can see that inverse dynamics required fewer iterations than forward dynamics to cross the faint-green line, which marks the point at which the error becomes acceptable to be considered feasible. Inverse dynamics converged in 26 iterations, and forward dynamics converged in 43 iterations. Inadvertently, one advantage of the forward formulation was that its final feasibility error was smaller than that of inverse dynamics.

When minimising a cost function, the locally-optimal solutions computed with either formulation are essentially the same. However, when an objective function is not considered, the formulations may diverge to different solutions. Experimentally, we have observed that the solutions computed with inverse dynamics are easier to perform in real hardware. The reasons behind this divergence are not yet clear to us, and this is something that should be investigated in future work.

6.5.4 *Hardware Validation*

We conducted real-world experiments with ANYmal [37] and TALOS [62] to validate the trajectories computed with our framework. The motion planning is performed offline, and then the trajectories are sent to the controller for playback. To execute the whole-body motions, we commanded each joint with feedforward torque and feedback on joint position and velocity. For the quadruped, we updated the references for each joint's position, velocity, and torque at 400 Hz. The decentralized motor controller at every joint closes the loop, compensating for friction effects. On the humanoid, we updated the references at 2 kHz, and a centralized controller compensates for the motor dynamics and friction.



(a) Initial state (b) Take-off (c) Full-flight phase (d) Landing (e) Final state

Figure 37: Snapshots of ANYmal [37] performing a 0.5 m-long jump. The length of the black tape on the ground is 0.5 m.

Figure 37 and Figure 38 contain snapshots of the jumps realized with the quadruped and with the humanoid, respectively. These experiments can be seen in our supplementary video: `https://youtu.be/pV4s7hzUgjc`. Jumping motion is challenging to execute in real hardware because it includes a severely underactuated phase when the robot is fully off the ground. Nonetheless, our controller is able to execute our planned trajectories reliably, attesting the dynamical consistency of our formulation.

As a final note, we would like to mention that we observed interesting emerging behaviours from our framework. For example, on the task in which the humanoid robot jumps, the resulting motions swing the arms upwards to build up energy before the take-off instant (see Figure 38). This reminded us of the feature that Erez and Todorov [22] observed in their results: an emergent coordination between legs

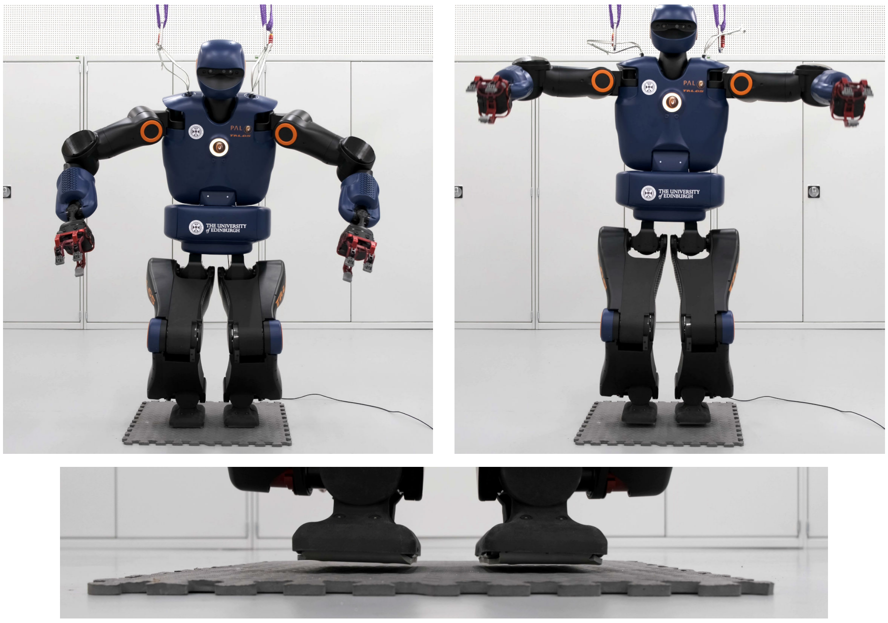


Figure 38: Snapshots of the humanoid TALOS [62] jumping.

and opposite arms during a running gait. Both in [22] and our work, these features originated without any explicit modelling, reaffirming the power of dynamic trajectory optimisation.

6.6 CONCLUSION

In this chapter, we have observed that direct transcription implementations relying on forward dynamics to define defect constraints can be reformulated with inverse dynamics to see an increase in performance, for both feasibility or minimisation problems, and without sacrificing the feasibility of the solutions to the optimisation problem. An additional reason to prefer inverse dynamics is robustness to coarser discretisation, both in terms of computation efficiency and faithfulness of solutions with respect to finer discretisation.

In the next chapter, we will conclude this thesis by looking back at our contributions, summarising our work's limitations, and listing possible avenues for future work.

7

CONCLUSION AND FUTURE WORK

In this thesis, we have presented our latest research work in the field of robotics, where we have focused on optimising dynamic trajectories for legged robots that maximise robustness against external disturbances. We started by looking at how to plan robust single whole-body configurations only, and then we looked at how to optimise robust trajectories, first for fixed-base robot arms, then for legged robots without making or breaking contacts (standing balance behaviours), and finally for legged robots with contact switching. We also investigated the use of inverse dynamics instead of forward dynamics for improving the core of our trajectory optimisation framework.

The main outcomes of our work is this thesis, which collects the body of knowledge that we have built, and a mature trajectory optimisation framework for legged robots capable of manipulating their environment. We have shown multiple robot hardware experiments, whereby the trajectories computed by our framework were put to the test for solving real-world tasks. The results of these experiments have shown that our system is reliable and versatile. In our industrial demonstrator, a quadruped robot equipped with an arm was able to navigate the scaffolding and successfully execute all the challenges we prepared: turn a wheel, pull a lever, open a gate whilst standing on a ramp, and pulling a rope in order to lift a heavy bucket. Moreover, we have shown that the smallest unrejectable force (SUF) is a valuable metric, both for analysing existing trajectories and for maximising the robustness of new trajectories. Finally, our preliminary research on the optimisation of feet locations showed promising results.

In the remaining of this chapter, we will discuss the limitations of our current approach and propose ways to tackle those limitations, and we will also suggest interesting avenues for future work.

7.1 FRAMEWORK LIMITATIONS

We now present the three main limitations of our framework in its current state.

7.1.1 *Lack of Collision Avoidance*

Collision avoidance is completely neglected in the current implementation of our framework. The trajectories we have shown in the videos and figures of this thesis are collision-free thanks to the way tasks are formulated and thanks to the initial guess provided to the solver. Nonetheless, it is the end-user's (i.e., human operator) responsibility to double-check the solutions visually at first and then in simulation before playing them back on the hardware. This is important to ensure there are no self-collisions or collisions with obstacles in the environment (including e.g. robot limbs clipping underneath the floor).

In general, collision avoidance is very expensive and it increases the computational burden on solvers. It is not clear to us whether collision avoidance should be part of our NLP formulation. Perhaps a workaround would be to either provide a better initial guess by means of a sampling-based method with explicit collision avoidance (similar to the one we implemented in Chapter 2), or by checking the solution for collisions at the end of the optimisation and re-running the optimisation if collisions occur—or possibly both.

7.1.2 *Inaccurate Dynamics Models*

In our trajectory optimisation work, we have always considered the full-order system dynamics of the robots that we used. This means that we consider the mass, inertia, and other dynamic quantities of every link of the robot, instead of using other more-simplistic models. However, this does not mean that our model is perfect, nor does it mean that we take into account the dynamics of every physical component inside the robots. For example, we do not model the mass and inertia of the cables inside the robot, nor their motion while the robot is moving. One can argue that these should be neglected as they are very small but, in fact, the combined motion of all the cabling inside robots does indeed affect the dynamics of the real robot—it just so happens that most robot controllers see these as just another disturbance (in this case internal) to be compensated for.

Perhaps a more important aspect than unmodelled loose cables inside the robot is the unmodelled dynamics of actuators. We do not model the gearing friction, stiction, and other dynamical aspects of robot actuators. Unfortunately, it is well-known that actuator dynamics should not be ignored, as they plan an important role in motion control [41]. Neglecting actuator dynamics is one of the main reasons why the sim-to-real gap of complex actuators is much more significant than it is for direct drive motors. Since analytical models for actuator dynamics are very difficult to obtain, one could resort to machine learning techniques, similar to the work of Hwangbo *et al.* [38]. An interesting avenue for future work would be to incorporate such a neural network (NN) as a component within our trajectory optimisation framework. This would result in a hybrid framework, in the sense that there would be a model-based component for the trajectory optimisation and a learning-based component for the actuators' dynamics (which would have to be differentiable for the sake of computing first- and second-order derivatives).

7.1.3 *Too Expensive for Continuous Re-planning*

For many real-world tasks, such as the ones we showed throughout our experiments, it is sufficient to take the current state of the world, plan a motion in "one-shot" for completing the task, and then executing that trajectory. However, there are scenarios where this approach might fail, especially if the state of the world changes as the robot executes the task—meaning that the planned trajectories become invalid. Real-time control schemes, such as model predictive control (MPC), are usually employed for dealing with those scenarios. Currently, the computation times of our framework for planning robust trajectories are too slow to be compatible with the budget available in those real-time control schemes. There are a few possible approaches for tackling this challenge, if we want to use our framework for such scenarios.

In the first approach, we can try to decrease the computational time by, e.g., making the robot model more simple, making the problem smaller (through coarser discretisation), or decreasing the complexity of the problem constraints.

Another approach would be to change the NLP solver used for solving the trajectory optimisation problems. For example, we could attempt to formulate our robustness optimisation problem using a DDP-based approach, such as [49], instead of using a direct transcription approach that relies on off-the-shelf solvers. However, the disadvantage of this approach is that enforcing general constrains with DDP-based approaches is pretty much an active research topic of its own, whereas commercial solvers are able to deal with a wide range of constraints out of the box.

Finally, a third approach would be to speed up the computation of the robustness metric itself. Perhaps there is a way the metric can be learnt, represented by a surrogate model, or stored in a look-up table for quick retrievals. In the 'Future Work' section below, we will come back to this point.

7.2 FUTURE WORK

We now suggest interesting avenues for future work in this line of research. The suggestions are sorted from (what we believe to be) the most straightforward to the most challenging.

7.2.1 *Parallelising the Framework*

Simultaneous methods, like direct transcription, discretise states and controls over time as decision variables. Representing the entire trajectory in this way is amenable for parallelisation.

Currently, our trajectory optimisation framework uses a single thread for all computations, but it is possible to use multiple threads in order to evaluate the constraints' functions and derivatives simultaneously. This could impart some overhead for small problems, but for larger problems with many mesh points it should greatly improve performance.

7.2.2 *Analysing Whole-Body Robustness*

Throughout this thesis, we have focused on improving robustness against external disturbances applied at the end-effector. This is because we were interested in applications where the robot had to interact with objects, but we did not want to model the dynamics of those objects. However, the mathematical derivation of the SUF applies to any point on any rigid body of the robot mechanism. In other words, it is possible to compute the SUF at the robot's base or knees, rather than just at the end-effector.

An interesting path for future work would be the development of a software tool for analysing robot trajectories where a robot and a trajectory are given as inputs, and then as an output it would show the SUF spheres at multiple points along the robot's kinematic chain. This kind of visualization would allow us to better understand and have a very clear visual representation of bottlenecks and weaknesses in trajectories.

In short, a tool for visualising the SUF for different points of the robot during a trajectory would allow us to spot if the robot is susceptible to failure at specific parts of its body, even for small disturbances, and at which instants of the motion.

7.2.3 *Learning the Robustness Metric*

As mentioned in the previous section, 'Framework Limitations', our current approach for calculating the SUF requires a significant amount of computational power. An interesting way to tackle this would be to approximate the robustness metric with a surrogate model. For example, we could train a NN using an offline-generated dataset of ground truth samples. Evaluating the SUF through the NN model would lead to much faster inference times, compared to calculating the SUF with the optimisation approach that we have been using. So, in short, the goal would be to decrease inference time by sacrificing the accuracy of the output.

Orsolino *et al.* [56] have already shown promising results in this direction. They trained a multilayer perceptron in order to learn a stability metric for quadrupedal locomotion. Then, they used the model for learning locomotion through reinforcement learning, and they also showed that the model could be embedded in a trajectory optimisation framework—which is just what we would like to do.

In our case, faster function and derivative evaluations of the learnt model would allow planning frameworks (not only ours) to consider the robustness criterion at more-compatible rates for re-planning motion during execution. It would also alleviate the burden of developers, since they would no longer need to define and formulate all the mathematical constraints that we have derived in our work; instead, they would simply be able to embed a pre-trained model in their optimisation framework, by introducing an extra term in their cost functions.

7.2.4 *Taking Into Account Force-Feedback During Execution*

In our work, we have disregarded the dynamics model of objects being manipulated by the robot. We maximized robustness against external disturbances at the end-effector in order to demonstrate that we do not have to necessarily model the object dynamics (which may not be available at planning time), as the controller is able to track the reference trajectory and compensate for the object dynamics through feedback terms at the controller level. However, in extreme cases, controller feedback terms are not enough to execute the desired motion appropriately.

An immediate extension of our work would be to take into account force-feedback during execution of the task. We could plan the motion just like we have done throughout this chapter before starting the actual manipulation task. However, once the robot starts executing the task, the object being manipulated will exert a set of forces that in turn apply torque at the robot's joints. An interesting direction for future work would be to take into account those torques at the joint level to estimate the force being applied to the robot as a consequence of the robot-object interaction, and then replanning the motion taking into account that force estimation. This would mean that we would still be able to maximize the SUF, but we would now have a much better model of the task taking place.

7.2.5 *Maximising Robustness Through Environment Exploitation*

Throughout this thesis (except for parts of Chapter 3), we have focused on isotropic robustness, i.e., being robust against external disturbances from any given direction. However, for some tasks, e.g., the ones shown in Figure 39, it may be more appropriate to be able to resist disturbances along a particular direction.

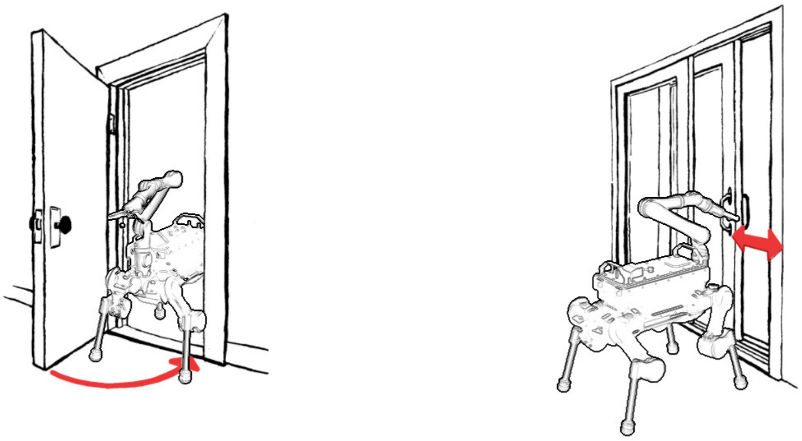


Figure 39: Sketches of a quadruped robot equipped with an arm opening and closing two different types of doors. The door on the left rotates around a pivot point at its hinges, whereas the door on the right slides left and right on a frame.

It would be interesting to maximise the robustness along pre-determined directions by changing our formulation to find the maximum-volume ellipsoid inscribed in the projection of the residual force polytope, rather than fitting a maximum-volume ball. The most straightforward way to do this would be to pre-specify the ellipsoid profile that we would like to maximise (i.e., the shape of the ellipsoid), and then modify our mathematical constraints to take that into account instead of considering a uniform ball. Then, the ρ parameter (see Equation 43) in the decision variables of our optimisation problem would represent the scaling factor of the ellipsoid, rather than the radius of the maximum-volume ball inscribed in the polytope.

7.2.6 *Minimising the Loss of Robustness*

Our framework maximises the robustness of a trajectory assuming reliable execution. However, the system dynamics around a nominal trajectory are not linear, and given large enough perturbations the robot may be driven into areas of low robustness.

It would be interesting to implement and derive a new formulation that accounts for the robustness of neighbouring states and the noise at the control stage. This would allow solvers to compute trajectories that are robust in nominal cases and near that nominal region, instead of just being very robust in the nominal path and yet susceptible to a quick loss of robustness due to small trajectory deviations.

A

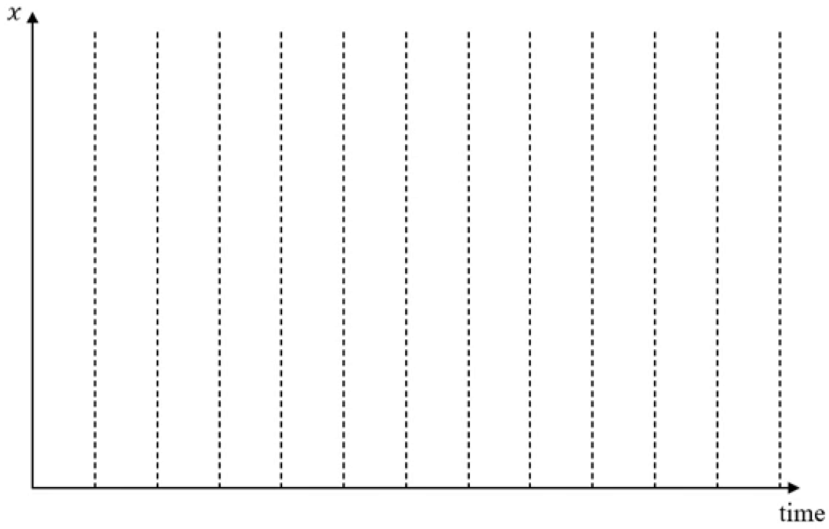
APPENDIX

In this appendix, we give a brief and (hopefully) intuitive explanation of how direct transcription works.

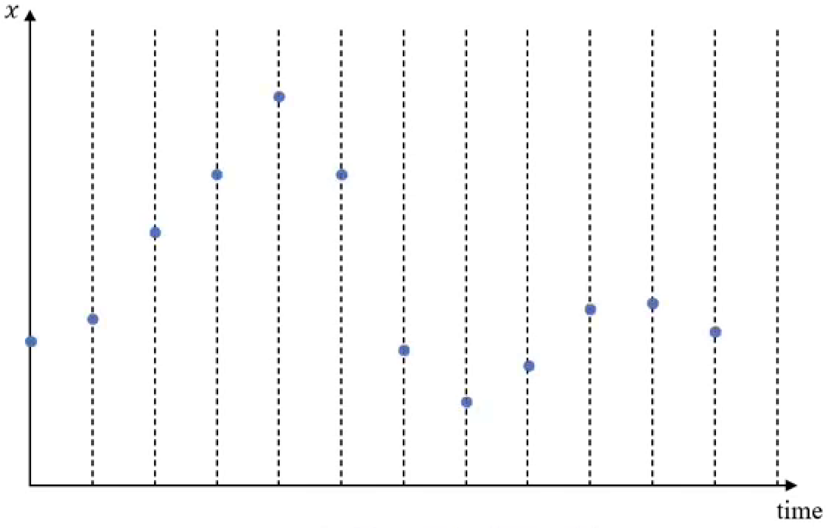
A.1 DIRECT TRANSCRIPTION ILLUSTRATED

Consider the two axes in the plot below. The horizontal axis represents time, and the vertical axis represents the state of a system, e.g., the state of a robot, rocket, or satellite. This means that, while the horizontal axis represents only one dimension, the vertical axis can represent multiple dimensions.

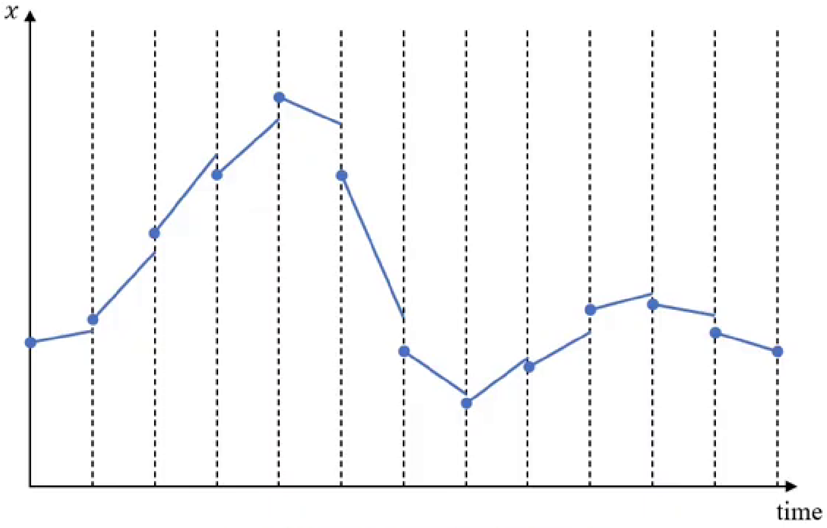
Direct transcription works by splitting time into segments, like so:



Next, at the beginning of each segment in time, we represent the state of the system as decision variables. These states are denoted by the blue circles in the plot:



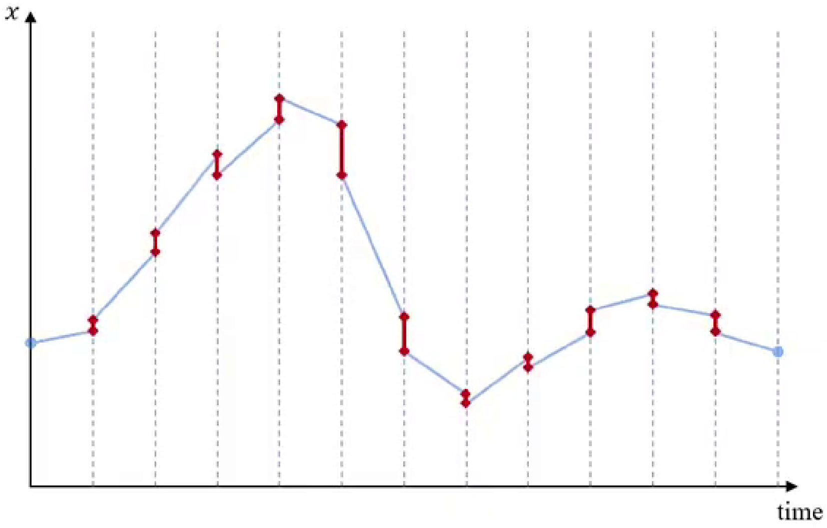
For each segment, in addition to the system state, we also represent (as decision variables) the control inputs commanded to the system for the duration of that segment. Those control inputs determine how system states evolve throughout the duration of their respective time "slice". In the plot below, the state evolution within each slice is represented as the blue line segment originating from the blue circles.



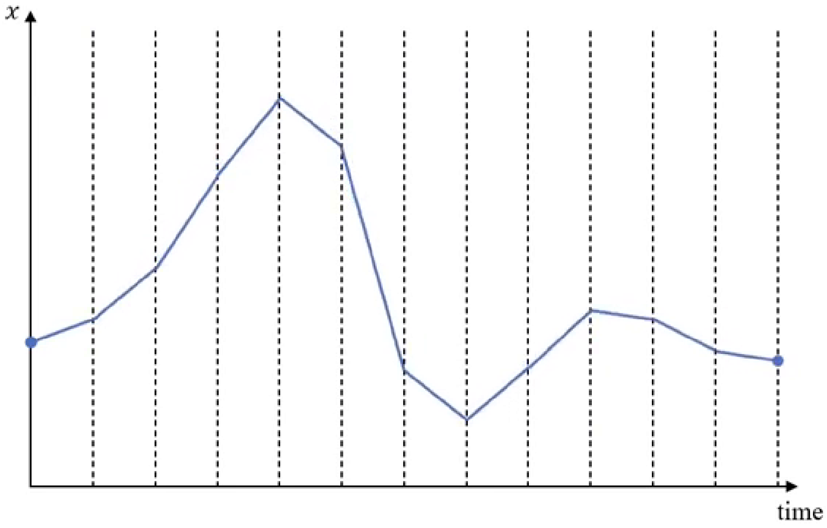
When solving an optimization problem, we need to assign an initial value to the decision variables we choose for our formulation. Initial guesses *strongly* affect how quickly solvers converge to a final solution: the closer an initial guess is to a feasible solution, the less numerical work the solver needs to do to “tune” the values of the decision variables—in order to make up a valid trajectory, that is.

Researchers and scientists often try different initialization approaches. These approaches can be as simple as setting all decision variables to zeros (or random values), linearly interpolating between initial and target states, or as elaborate as querying a pre-built cache/library of feasible trajectories.

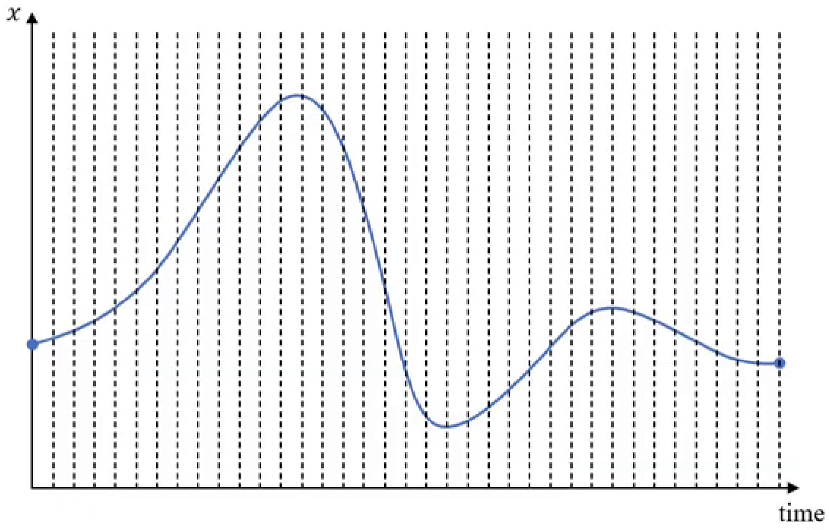
These guesses do not have to be accurate. In fact, more often than not, they are not physically feasible due to state mismatches from one time segment to the next. These mismatches are called *dynamics defects*. In the plot below, they are highlighted in red.



After transcribing the optimal control problem and providing an initial guess for the trajectory, we end up with a mathematical formulation of a nonlinear programming problem. This is then passed on to a numerical solver, which solves the optimization problem—closing the gaps in the process.



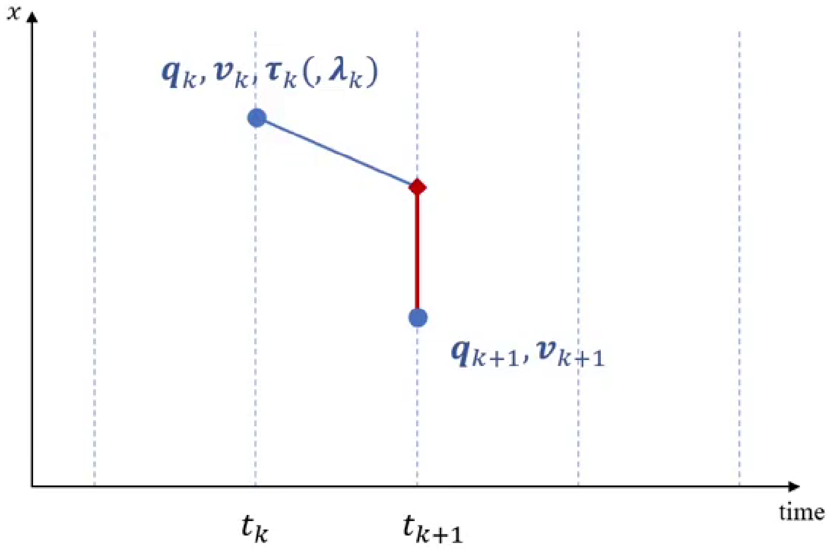
Finally, in order to better approximate the nonlinear dynamics of high-dimensional systems, trajectories are often discretized using small time steps:



In summary, direct transcription discretizes a trajectory in time, and takes the state of the robot and the control inputs as decision variables. Then, it tries to find a trajectory that satisfies the system dynamics, as well as other task-related constraints.

A.2 DYNAMICS DEFECTS ILLUSTRATED

Let us now better understand what the dynamics defects are by looking at an isolated instance of such a defect:



In the plot above, the blue circle at time step t_k represents the state given by the joint positions and joint velocities q_k and v_k. The blue line coming out of that circle shows how the state progresses over time (as a consequence of the torques and forces applied to the robot). The red diamond represents the state where the robot ends up according to the decision variables from time step t_k, whereas the bottom blue circle represents the state of the robot at that point according to the decision variables from time step t_{k+1}. The red line represents the mismatch between the evolved state and the discretized state.

In summary, there is redundancy in the way direct transcription represents a trajectory, and so we must ensure that the underlying values are consistent. For that, we must be able to calculate the defects, and define constraints so that the solver eliminates those defects. In order to quantify them, we have to solve either the forward or the inverse dynamics problem—which we compared in Chapter 6.

BIBLIOGRAPHY

[1] Hervé Audren and Abderrahmane Kheddar. '3D Robust Stability Polyhedron in Multicontact'. In: *IEEE Transactions on Robotics (T-RO)* (2018) (cit. on p. 25).

[2] Sébastien Barthélemy and Philippe Bidaud. 'Stability Measure of Postural Dynamic Equilibrium Based on Residual Radius'. In: *Advances in Robot Kinematics: Analysis and Design*. Springer, 2008 (cit. on p. 10).

[3] C. D. Bellicoso, K. Krämer, M. Stäuble, D. Sako, F. Jenelten, M. Bjelonic and M. Hutter. 'ALMA — Articulated Locomotion and Manipulation for a Torque-Controllable Robot'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019 (cit. on pp. 43, 61, 62).

[4] A. Ben-Tal and A. Nemirovski. 'Robust Truss Topology Design via Semidefinite Programming'. In: *SIAM Journal on Optimization* 7.4 (1997), pp. 991–1016 (cit. on p. 25).

[5] John T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Second. Vol. 19. Society for Industrial and Applied Mathematics (SIAM), 2010 (cit. on pp. 1, 30, 32, 46, 83).

[6] Jeff Bezanson, Alan Edelman, Stefan Karpinski and Viral B. Shah. 'Julia: A Fresh Approach to Numerical Computing'. In: *SIAM Review* 59.1 (2017), pp. 65–98 (cit. on pp. 34, 55, 69, 89).

[7] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004 (cit. on p. 29).

[8] Angelo Bratta, Romeo Orsolino, Michele Focchi, Victor Barasuol, Giovanni Gerardo Muscolo and Claudio Semini. 'On the Hardware Feasibility of Nonlinear Trajectory Optimization for Legged Locomotion based on a Simplified Dynamics'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020 (cit. on p. 41).

[9] Timothy Bretl and Sanjay Lall. 'Testing Static Equilibrium for Legged Robots'. In: *IEEE Transactions on Robotics (T-RO)* (2008) (cit. on p. 10).

[10] Felix Burget and Maren Bennewitz. 'Stance selection for humanoid grasping tasks by inverse reachability maps'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015 (cit. on pp. 8–10, 13).

[11] R. H. Byrd, J. Nocedal and R. A. Waltz. 'Knitro: An Integrated Package for Nonlinear Optimization'. In: *Large-Scale Nonlinear Optimization*. Ed. by G. Di Pillo and M. Roma. Springer US, 2006 (cit. on pp. 34, 35, 41, 46, 55, 89).

[12] A. Campeau-Lecours, H. Lamontagne, S. Latour, P. Fauteux, V. Maheu, F. Boucher, C. Deguire and L. C. L'Ecuyer. 'Kinova modular robot arms for service robotics applications'. In: *Rapid Automation: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2019 (cit. on pp. 44, 55).

[13] Stéphane Caron, Quang-Cuong Pham and Yoshihiko Nakamura. 'Leveraging Cone Double Description for Multi-contact Stability of Humanoids with Applications to Statics and Dynamics'. In: *Robotics, Science and System (RSS)*. 2015 (cit. on pp. 2, 10, 44, 49, 62, 63, 66, 87).

[14] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse and Nicolas Mansard. 'The Pinocchio C++ library — A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives'. In: *IEEE International Symposium on System Integrations (SII)*. 2019 (cit. on p. 84).

[15] Jiahao Chen, Jarrett Revels and Alan Edelman. 'Robust benchmarking in noisy environments'. In: *Proceedings of the 20th IEEE High Performance Extreme Computing Conference*. Waltham, USA, 2016 (cit. on p. 91).

[16] Pasquale Chiacchio, Yann Bouffard-Vercelli and François Pierrot. 'Force Polytope and Force Ellipsoid for Redundant Manipulators'. In: *Journal of Robotic Systems* 14.8 (1997), pp. 613–620 (cit. on pp. 27, 33).

[17] X. Chu, Q. Hu and J. Zhang. 'Path Planning and Collision Avoidance for a Multi-Arm Space Maneuverable Robot'. In: *IEEE Transactions on Aerospace and Electronic Systems* 54.1 (2018), pp. 217–232 (cit. on p. 24).

[18] R. Deits and R. Tedrake. 'Footstep planning on uneven terrain with mixed-integer convex optimization'. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. 2014 (cit. on pp. 7, 15).

[19] Andrea Del Prete and Nicolas Mansard. 'Robustness to Joint-Torque-Tracking Errors in Task-Space Inverse Dynamics'. In: *IEEE Transactions on Robotics (T-RO)* (2016) (cit. on pp. 2, 23, 44, 60, 62).

[20] Andrea Del Prete, Steve Tonneau and Nicolas Mansard. 'Fast Algorithms to Test Robust Static Equilibrium for Legged Robots'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2016 (cit. on pp. 11, 12).

[21] Jared Di Carlo, Patrick M. Wensing, Benjamin Katz, Gerardo Bledt and Sangbae Kim. 'Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control'. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 1–9 (cit. on p. 2).

[22] T. Erez and E. Todorov. 'Trajectory optimization for domains with contacts using inverse dynamics'. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2012, pp. 4914–4919 (cit. on pp. 84, 86, 94, 95).

[23] M. F. Fallon, P. Marion, R. Deits, T. Whelan, M. Antone, J. McDonald and R. Tedrake. 'Continuous humanoid locomotion over uneven terrain using stereo fusion'. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. 2015 (cit. on p. 20).

[24] Maurice Fallon, Scott Kuindersma, Sisir Karumanchi, Matthew Antone, Toby Schneider, Hongkai Dai et al. 'An Architecture for Online Affordance-based Perception and Whole-body Planning'. In: *Journal of Field Robotics* (2015) (cit. on p. 8).

[25] Martin L. Felis. 'RBDL: an efficient rigid-body dynamics library using recursive algorithms'. In: *Autonomous Robots (AuRo)* (2016), pp. 1–17. ISSN: 1573-7527 (cit. on p. 84).

[26] H. Ferrolho, W. Merkt, Y. Yang, V. Ivan and S. Vijayakumar. 'Whole-Body End-Pose Planning for Legged Robots on Inclined Support Surfaces in Complex Environments'. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. 2018, pp. 944–951 (cit. on p. 24).

[27] Henrique Ferrolho and contributors. *TORA.jl*. 2020. URL: `https://github.com/JuliaRobotics/TORA.jl` (cit. on p. 85).

[28] Henrique Ferrolho, Vladimir Ivan, Wolfgang Merkt, Ioannis Havoutis and Sethu Vijayakumar. 'Inverse Dynamics vs. Forward Dynamics in Direct Transcription Formulations for Trajectory Optimization'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021 (cit. on pp. 60, 66).

[29] Henrique Ferrolho, Wolfgang Merkt, Vladimir Ivan, Wouter Wolfslag and Sethu Vijayakumar. 'Optimizing Dynamic Trajectories for Robustness to Disturbances Using Polytopic Projections'. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020 (cit. on pp. 60, 62, 63, 67–69, 71).

[30] Henrique Ferrolho, Wolfgang Merkt, Carlo Tiseo and Sethu Vijayakumar. 'Residual force polytope: Admissible task-space forces of dynamic trajectories'. In: *Robotics and Autonomous Systems (RAS)* (2021) (cit. on pp. 44, 45, 50, 62, 63).

[31] Marco Frigerio, Jonas Buchli, Darwin G. Caldwell and Claudio Semini. 'RobCoGen: a code generator for efficient kinematics and dynamics of articulated robots, based on Domain Specific Languages'. In: *Journal of Software Engineering for Robotics (JOSER)* 7.1 (Special Issue on Domain-Specific Languages and Models for Robotic Systems 2016), pp. 36–54 (cit. on p. 84).

[32] Komei Fukuda and Alain Prodon. 'Double Description Method Revisited'. In: *Combinatorics and Computer Science (CCS)*. Springer, 1995 (cit. on p. 26).

[33] M. Giftthaler, M. Neunert, M. Stäuble, J. Buchli and M. Diehl. 'A Family of Iterative Gauss-Newton Shooting Methods for Nonlinear Optimal Control'. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018 (cit. on pp. 31, 46).

[34] P. G. Gormley. ‘Stereographic Projection and the Linear Fractional Group of Transformations of Quaternions’. In: *Proceedings of the Royal Irish Academy. Mathematical and Physical Sciences* 51 (1945) (cit. on pp. 46, 63, 85).

[35] Jesse Haviland and Peter Corke. ‘Maximising Manipulability During Resolved-Rate Motion Control’. In: *arXiv e-prints arXiv:2002.11901* (Feb. 2020) (cit. on p. 24).

[36] H. Hirukawa, S. Hattori, K. Harada, S. Kajita, K. Kaneko, F. Kanehiro, K. Fujiwara and M. Morisawa. ‘A Universal Stability Criterion of the Foot Contact of Legged Robots - Adios ZMP’. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2006 (cit. on p. 25).

[37] M. Hutter et al. ‘ANYmal - toward legged robots for harsh environments’. In: *Advanced Robotics* (2017) (cit. on pp. 44, 55, 94).

[38] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun and Marco Hutter. ‘Learning agile and dynamic motor skills for legged robots’. In: *Science Robotics* 4.26 (2019), eaau5872 (cit. on p. 98).

[39] 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)*. Springer, 2019 (cit. on p. 16).

[40] Noémie Jaquier, Leonel Dario Rozo, Darwin G Caldwell and Sylvain Calinon. ‘Geometry-aware Tracking of Manipulability Ellipsoids’. In: *Robotics, Science and Systems (RSS)*. Pittsburgh, Pennsylvania, 2018 (cit. on p. 24).

[41] Mohamed Kara Mohamed and Alexander Lanzon. ‘Effect of unmodelled actuator dynamics on feedback linearised systems and a two stage feedback linearisation method’. In: *IEEE Conference on Decision and Control*. 2013, pp. 841–846 (cit. on p. 98).

[42] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz and N. Mansard. ‘Whole-body model-predictive control applied to the HRP-2 humanoid’. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 3346–3351 (cit. on p. 2).

[43] Twan Koolen and contributors. *RigidBodyDynamics.jl*. 2016. URL: `https://github.com/JuliaRobotics/RigidBodyDynamics.jl` (cit. on pp. 84, 89).

[44] Twan Koolen and Robin Deits. ‘Julia for robotics: simulation and real-time control in a high-level programming language’. In: *International Conference on Robotics and Automation (ICRA)*. 2019 (cit. on p. 84).

[45] Benoit Landry, Joseph Lorenzetti, Zachary Manchester and Marco Pavone. ‘Bilevel Optimization for Planning through Contact: A Semidirect Method’. In: *International Symposium on Robotics Research (ISRR)*. 2019 (cit. on p. 48).

[46] Benoit Landry, Zachary Manchester and Marco Pavone. 'A Differentiable Augmented Lagrangian Method for Bilevel Nonlinear Optimization'. In: *Robotics, Science and System (RSS)*. 2019 (cit. on p. 45).

[47] Yuntao Ma, Farbod Farshidian, Takahiro Miki, Joonho Lee and Marco Hutter. 'Combining Learning-Based Locomotion Policy With Model-Based Manipulation for Legged Mobile Manipulators'. In: *IEEE Robotics and Automation Letters (RA-L)* (2022) (cit. on pp. 60–62).

[48] Zachary Manchester and Scott Kuindersma. 'DIRTREL: Robust Nonlinear Direct Transcription with Ellipsoidal Disturbances and LQR Feedback'. In: *Robotics, Sciences and Systems (RSS)*. 2017 (cit. on pp. 23, 25).

[49] 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)*. 2020 (cit. on pp. 31, 46, 99).

[50] David Mayne. 'A Second-order Gradient Method for Determining Optimal Trajectories of Non-linear Discrete-time Systems'. In: *International Journal of Control* 3.1 (1966), pp. 85–95 (cit. on p. 31).

[51] Biren Mehta and Stefan Schaal. 'Forward Models in Visuomotor Control'. In: *Journal of Neurophysiology* 88.2 (2002), pp. 942–953 (cit. on p. 84).

[52] Michael P. Murphy, Benjamin Stephens, Yeuhi Abe and Alfred A. Rizzi. 'High degree-of-freedom dynamic manipulation'. In: *Unmanned Systems Technology XIV*. International Society for Optics and Photonics. SPIE, 2012 (cit. on pp. 60–62).

[53] S. M. Neuman, T. Koolen, J. Drean, J. E. Miller and S. Devadas. 'Benchmarking and Workload Analysis of Robot Dynamics Algorithms'. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019 (cit. on p. 84).

[54] Michael Neunert, Markus Stäuble, Markus Giftthaler, Carmine D. Bellicoso, Jan Carius, Christian Gehring, Marco Hutter and Jonas Buchli. 'Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds'. In: *IEEE Robotics and Automation Letters (RA-L)* 3.3 (2018), pp. 1458–1465 (cit. on p. 2).

[55] R. Orsolino, M. Focchi, C. Mastalli, H. Dai, D. G. Caldwell and C. Semini. 'Application of Wrench-Based Feasibility Analysis to the Online Trajectory Optimization of Legged Robots'. In: *IEEE Robotics and Automation Letters (RA-L)* 3.4 (2018), pp. 3363–3370 (cit. on pp. 2, 23, 25, 33, 41, 44, 45, 50, 62, 63).

[56] Romeo Orsolino, Siddhant Gangapurwala, Oliwier Melon, Mathieu Geisert, Ioannis Havoutis and Maurice Fallon. 'Rapid Stability Margin Estimation for Contact-Rich Locomotion'. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 8485–8492 (cit. on p. 100).

[57] D. Pardo, L. Möller, M. Neunert, A. W. Winkler and J. Buchli. 'Evaluating Direct Transcription and Nonlinear Optimization Methods for Robot Motion Planning'. In: *IEEE Robotics and Automation Letters (RA-L)* 1.2 (2016), pp. 946–953 (cit. on p. 84).

[58] Michiel Plooij, Wouter Wolfslag and Martijn Wisse. 'Robust feedforward control of robotic arms with friction model uncertainty'. In: *Robotics and Autonomous Systems (RAS)* 70 (2015), pp. 83–91 (cit. on p. 23).

[59] Michael Posa, Cecilia Cantu and Russ Tedrake. 'A direct method for trajectory optimization of rigid bodies through contact'. In: *The International Journal of Robotics Research (IJRR)* 33.1 (2014), pp. 69–81 (cit. on pp. 35, 47, 69, 85).

[60] Nicolas Schweighofer, Michael A Arbib and Mitsuo Kawato. 'Role of the cerebellum in reaching movements in humans. I. Distributed inverse dynamics control'. In: *European Journal of Neuroscience* (1998) (cit. on p. 84).

[61] Jean-Pierre Sleiman, Farbod Farshidian, Maria Vittoria Minniti and Marco Hutter. 'A Unified MPC Framework for Whole-Body Dynamic Locomotion and Manipulation'. In: *IEEE Robotics and Automation Letters (RA-L)* (2021) (cit. on pp. 60, 62).

[62] O. Stasse et al. 'TALOS: A new humanoid research platform targeted for industrial applications'. In: *IEEE-RAS International Conference on Humanoid Robotics (Humanoids)*. 2017 (cit. on pp. 84, 94, 95).

[63] T. Stouraitis, I. Chatzinikolaidis, M. Gienger and S. Vijayakumar. 'Online Hybrid Motion Planning for Dyadic Collaborative Manipulation via Bilevel Optimization'. In: *IEEE Transactions on Robotics (T-RO)* 36.5 (2020) (cit. on p. 87).

[64] Theodoros Stouraitis, Iordanis Chatzinikolaidis, Michael Gienger and Sethu Vijayakumar. 'Dyadic collaborative Manipulation through Hybrid Trajectory Optimization'. In: *Proceedings of The 2nd Conference on Robot Learning (CoRL)*. Vol. 87. PMLR, 2018, pp. 869–878 (cit. on p. 35).

[65] O. von Stryk and R. Bulirsch. 'Direct and Indirect Methods for Trajectory Optimization'. In: *Annals of Operations Research* (1992) (cit. on p. 30).

[66] Sung-Hee Lee, Junggon Kim, F. C. Park, Munsang Kim and J. E. Bobrow. 'Newton-type algorithms for dynamics-based robot movement optimization'. In: *IEEE Transactions on Robotics (T-RO)* 21.4 (2005), pp. 657–667 (cit. on p. 84).

[67] Yuval Tassa, Nicolas Mansard and Emo Todorov. 'Control-limited differential dynamic programming'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 1168–1175 (cit. on p. 31).

[68] Russ Tedrake and the Drake Development Team. *Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems*. 2016. URL: `https://drake.mit.edu` (cit. on p. 14).

[69] G. Terzakis, M. Lourakis and D. Ait-Boudaoud. 'Modified Rodrigues Parameters: An Efficient Representation of Orientation in 3D Vision and Graphics'. In: *Journal of Mathematical Imaging and Vision* (2018) (cit. on pp. 46, 63, 85).

[70] Hans Raj Tiwary. 'On the Hardness of Computing Intersection, Union and Minkowski Sum of Polytopes'. In: *Discrete & Computational Geometry* 40.3 (2008), pp. 469–479 (cit. on pp. 45, 63).

[71] Emanuel Todorov. 'A convex, smooth and invertible contact model for trajectory optimization'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2011, pp. 1071–1076 (cit. on p. 85).

[72] Steve Tonneau, Nicolas Mansard, Chonhyon Park, Dinesh Manocha, Franck Multon and Julien Pettré. 'A reachability-based planner for sequences of acyclic contacts in cluttered environments'. In: *Robotics Research*. Springer, 2018 (cit. on p. 7).

[73] Steve Tonneau, Daeun Song, Pierre Fernbach, Nicolas Mansard, Michel Taïx and Andrea Del Prete. 'SL1M: Sparse L1-norm Minimization for contact planning on uneven terrain'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019 (cit. on pp. 65, 87).

[74] Nikolaus Vahrenkamp, Tamim Asfour and Rüdiger Dillmann. 'Robot placement based on reachability inversion'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2013 (cit. on p. 9).

[75] R. A. Waltz, J. L. Morales, J. Nocedal and D. Orban. 'An Interior Algorithm for Nonlinear Optimization That Combines Line Search and Trust Region Steps'. In: *Mathematical Programming* 107.3 (2006), pp. 391–408 (cit. on pp. 55, 89).

[76] 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 (RA-L)* 3.3 (2018), pp. 1560–1567 (cit. on pp. 35, 86, 87).

[77] Alexander W. Winkler. 'Optimization-based motion planning for legged robots'. PhD thesis. Zurich: ETH Zurich, 2018 (cit. on p. 35).

[78] W. J. Wolfslag, C. McGreavy, G. Xin, C. Tiseo, S. Vijayakumar and Z. Li. 'Optimisation of Body-ground Contact for Augmenting the Whole-Body Loco-manipulation of Quadruped Robots'. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020 (cit. on pp. 42, 43, 45, 51, 63).

[79] Guiyang Xin, Hsiu-Chin Lin, Joshua Smith, Oguzhan Cebe and Michael Mistry. 'A Model-Based Hierarchical Controller for Legged Systems Subject to External Disturbances'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018 (cit. on pp. 2, 23, 44, 60, 62).

[80] Y. Yang, V. Ivan, W. Merkt and S. Vijayakumar. 'Scaling sampling-based motion planning to humanoid robots'. In: *IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2016 (cit. on pp. 7, 15).

[81] Yiming Yang, Vladimir Ivan, Zhibin Li, Maurice Fallon and Sethu Vijayakumar. 'iDRM: Humanoid motion planning with realtime end-pose selection in complex environments'. In: *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. 2016 (cit. on pp. 7–10).

[82] Yiming Yang, Wolfgang Merkt, Henrique Ferrolho, Vladimir Ivan and Sethu Vijayakumar. 'Efficient Humanoid Motion Planning on Uneven Terrain Using Paired Forward-Inverse Dynamic Reachability Maps'. In: *IEEE Robotics and Automation Letters (RA-L)* (2017) (cit. on pp. 8–10, 12–14).

[83] Tsuneo Yoshikawa. 'Manipulability of Robotic Mechanisms'. In: *The International Journal of Robotics Research (IJRR)* 4.2 (1985), pp. 3–9 (cit. on pp. 24, 33, 45).

[84] Franziska Zacharias, Christoph Borst and Gerd Hirzinger. 'Capturing robot workspace structure: representing robot capabilities'. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2007 (cit. on p. 9).

[85] Jianzhe Zhen and Dick den Hertog. 'Computing the Maximum Volume Inscribed Ellipsoid of a Polytopic Projection'. In: *INFORMS Journal on Computing* 30.1 (2018), pp. 31–42 (cit. on pp. 42, 45, 51, 63).

[86] Günter M. Ziegler. *Lectures on Polytopes*. Vol. 152. Springer Science & Business Media, 2012 (cit. on p. 26).

[87] Simon Zimmermann, Roi Poranne and Stelian Coros. 'Go Fetch! - Dynamic Grasps using Boston Dynamics Spot with External Robotic Arm'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021 (cit. on pp. 60–62).