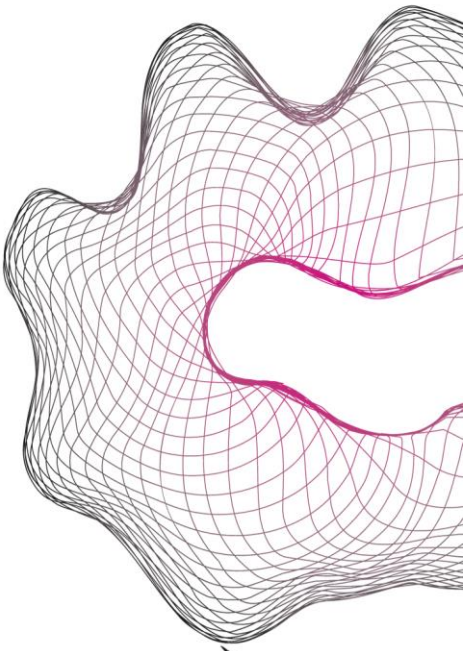
MASTER THESIS



SIMULTANEOUS OPTIMIZATION OF CONTACTS AND SMOOTH MOVEMENTS FOR CONTROL OF HUMANOID ROBOTICS

Hendrik van Gils

FACULTY OF ENGINEERING TECHNOLOGY
DEPARTMENT OF BIOMECHANICAL ENGINEERING

EXAMINATION COMMITTEE
dr. Edwin H.F. van Asseldonk
dr.ir. Arvid Q.L. Keemink
dr.ir. Wesley Roozing

DOCUMENT NUMBER
BE - 887

UNIVERSITY OF TWENTE.

16/12/2022

Abstract

Robotic exoskeletons have the potential to greatly improve the quality of life for paraplegic patients by providing them with greater independence and fewer negative health effects compared to wheelchairs. This study focuses on developing a control strategy for exoskeletons that allows them to mimic the natural behavior of a healthy individual as closely as possible. The current state-of-the-art method available at the Biomechanical Engineering department for doing this uses direct collocation with explicit contact constraints. This has limitations when it comes to tasks with unpredictable contact sequences, limiting it to the generation of gait. This thesis contributes by developing a new framework; which uses direct collocation with implicit contact constraints. This method does not require the user to specify a contact sequence, but only requires an objective function, a dynamic model, a time span over which to optimize and an initial guess. It then simultaneously optimizes the contacts and smooth movements. Four scenarios were tested with a biped model: gait, resisting perturbations, safe falling, and getting up. It is able to converge to a locally optimal solution in around 30 minutes for all tasks. However, for producing gait and safe falling trajectories, more task specific initial guesses were required. Additionally, the collocation errors were large and the mode sequences did not mimic human behaviour. Despite these limitations, the results of this framework demonstrate that this direct implicit method can be used to mimic human behaviour of different balance control related tasks. This makes it interesting for the application in lower limb exoskeletons and walking robots. There is still room for improvement in terms of fully realizing the potential of these devices to improve the mobility and quality of life of paraplegic patients. This framework represents an important step forward in the development of trajectory optimization for exoskeletons.

Acknowledgements

I would like to thank my friends for their support and everyone who was present during the student balance control meetings for the inspiration an motivation.

I would also like to distinctly mention:
Arvid has been an incredible supervisor, not only because of his expertise on many topics, but also because he cares about quality education and a personal approach.
Sjors de Bruin for all the fruitful discussions we had about collocation and trajectory optimization, the gym sessions and being a great friend in general.
Fianna and Hein were very welcoming once we were allowed to work in office again, and I would like to thank them for all the interesting and fun conversations.
Robin, Sem, Jonathan, Freek and Sjors have been great friends, thanks for all the coffee, beer and laughs!
Marije for her never-ending support.

Contents

Abstract i

Acknowledgements ii

1 Introduction 1
1.1 Human Walking is Not Always so Straightforward . . . 1
1.2 Robotic Solutions and Their Limitations . . . 1
1.3 Objective: Implicit Contacts . . . 3
1.4 Outline of the Thesis . . . 3

2 Related Works 4
2.1 Whole-Body Locomotion Planning with Direct Collocation and Explicit Contact Phases . . . 4
2.2 Centroidal Mixed-Integer Locomotion Planning . . . 4
2.3 Centroidal Locomotion Planning through Phase-Based End-Effector Parameterization . . . 5
2.4 Motion Planning through Contact-Invariant Optimization . . . 6
2.5 Direct Trajectory Optimization of Rigid Bodies Through Contact . . . 6
2.6 Optimization of Safe Falling Trajectories for Lower Limb Exoskeletons . . . 7
2.7 Summary of Trajectory Optimization Methods for Planning Through Contacts . 7

3 Background 9
3.1 Definitions and Conventions . . . 9
3.2 Optimization . . . 10
3.2.1 Iterative Optimization Methods . . . 10
3.2.2 Complementarity Constraints . . . 11
3.2.3 Solvers . . . 11
3.3 Dynamics . . . 11
3.3.1 Contact Dynamics . . . 12
3.4 Numerical Integration . . . 13
3.5 Direct Collocation . . . 13
3.5.1 What is Direct Collocation? . . . 14
3.5.2 Collocation Constraints for Euler-backward . . . 14
3.5.3 Intepolation between collocation points . . . 15

4 Method 16
4.1 Trajectory Optimization Technique . . . 16
4.2 Contact Handeling . . . 17
4.3 NLP Formulation . . . 18
4.4 9 DOF Walker . . . 19

4.5 Software Implementation . . . 19
4.6 Test Cases . . . 22
4.6.1 Gait . . . 22
4.6.2 Recovery from Perturbations . . . 23
4.6.3 Safe Falling . . . 23
4.6.4 Getting Up . . . 23

5 Results **24**
5.1 Gait . . . 24
5.2 Recovery from Perturbation . . . 25
5.3 Falling . . . 25
5.4 Recovery . . . 27

6 Discussion **30**
6.1 Comparison to Similar Method by Posa et al. . . . 30
6.2 Versatile Framework with Versatile Performance . . . 31
6.2.1 Large Deviations from Actual Dynamics . . . 31
6.2.1.1 Improving Accuracy . . . 32
6.2.2 The Road to Reliability . . . 32
6.2.2.1 The Role of Initial Guesses . . . 32
6.2.2.2 Improving Convergence by Adjusting the Time Span . . . 33
6.2.2.3 Mesh Refinement to Improve Accuracy and Convergence . . . 33
6.2.3 Abnormal Modes . . . 33
6.2.4 Costly Computation . . . 33
6.2.4.1 Improving the Computational Efficiency . . . 34
6.2.5 Task Specific Discussion . . . 34
6.2.5.1 Rediscovering Gait . . . 34
6.2.5.2 Perturbations: Refusing to Fall . . . 34
6.2.5.3 Safe Falling . . . 35
6.2.5.4 Getting Up . . . 35
6.3 Human-Like Movement . . . 35
6.4 Future Work . . . 35

References **37**

A Verification of Planar Dynamics **40**

B Deriving Collocation Methods and Corresponding Interpolant **41**
B.1 Euler Collocation (1st Order) . . . 41
B.2 Trapezoidal Collocation (1st Order) . . . 42
B.3 Hermite-Simpson Collocation (1st Order) . . . 43
B.4 Trapezoidal Collocation (2nd Order) . . . 45

C NLP Without Succinct Notation **47**

D Analytic Gradients for Dynamics **48**

E Additional Results **49**
E.1 Gait 49
E.2 Perturbation 49
E.3 Falling 56
E.3.1 Experimental Results Optimal Falling 56
E.4 Recovery 56

F Linear Combination for Ground Reaction Forces **63**

Chapter 1

Introduction

1.1 Human Walking is Not Always so Straightforward

Walking is an important action that most people take for granted every day. Walking allows us to efficiently move ourselves from one location to another, across all kinds of surfaces, without getting injured. For example, we can walk to the bathroom on the flat hallway floor. But we can also navigate through mountains on rocky terrain, or over ice. We can carry loads in our arms or on our backs while navigating all these different terrains, regain our balance when we get pushed, slip or trip. Our ability to stand, balance and walk also allows us to participate in society. We can mostly do the same activities as any other human, go on a run together, or play soccer with a team.

While most of us take it for granted, not everyone is as fortunate. It has been estimated that in developed western countries there are more than 4.6 million wheelchair users (WCU) [1], which is about 1% [2] of the population. While the treatment and rehabilitation for conditions causing patients to be in wheelchairs keeps improving, many still have to live the majority of their life with the disadvantages of their wheelchair.

Some disadvantages of wheelchairs are obvious, while others might be less known. The standard (pushrim) wheelchair is inefficient and causes chronic overuse of the shoulders and wrists, resulting in pain [2]. Propelling the chair over a flat surface such as in the home is relatively easy, but movement along outdoor paths (grass and gravel/sand surfaces) and ramps is difficult for the average individual. All of these things combined mean that WCU live a more restricted life and depend more on infrastrucure and others. This may limit their participation in everyday activities, and thus may also limit their quality of life [1, 2].

Other, lesser known disadvantages are caused by the lifestyle of always sitting down. Not using the muscles and bones in your legs causes them to atrophy (waste away). When your muscles get smaller, they get closer to the skin and you get more pressure points [3]. It has been estimated that insurance companies spend over 25% [3] of their spinal cord injury-related medical expenses for pressure sore treatment. Not moving your legs can cause your joints to lock, causing problems when one tries to move them later. It can also cause bowel issues and thrombosis [4]. These additional medical complication will further affect the quality of life for the WCU negatively.

1.2 Robotic Solutions and Their Limitations

Although restoring the physical function on a biological level could — in theory — also prevent patients from requiring a wheelchair, this thesis will focus on robotic exoskeletons. These exoskeletons can actuate the limbs of the patient mechanically, allowing paraplegic (lower-body

paralysis) patients to leave their wheelchair behind.

Multiple interesting methods have been developed to improve the performance of walking robots [5–12]. Luckily, these methods should also improve the performance of exoskeletons, as the combination of an exoskeleton with a human user could be considered as a biped robot with disturbances [5]. Many of these methods rely on Model Predictive Control (MPC), a type of control where a model of the system is used to predict and anticipate future events that allows the optimization of variables of the modelled system. This is an attractive approach, as it allows for reactive capabilities in addition to energy efficient — human-like — movements. This should enable recovery from slipping, falling and external disturbances, and re-planning if the motion is not executing as predicted [6].

Before the actuation of the patients limbs, a motion plan must be generated. A motion plan describes a sequence of positions that move the patient from its initial to its desired state over some time span. It is important that the motion plan is physically feasible, otherwise the robot is unable to execute it. Movements that make and break contact with the environment have several major restrictions:

- Forces can only be created when a foot is touching the ground.
- It is only possible to push feet against the ground, not pull on the ground.
- For legged locomotion without slipping, the location where the force is being generated cannot be moved. To reposition the foot there is a swing phase, during which the foot is unable to generate a force. This sudden loss in control is a restriction that is difficult to deal with.
- The forces that drive the body forward (tangential forces) must remain inside the friction cone to prevent slipping. To increase the tangential forces, the normal forces have to be increased. Which will result in a higher acceleration in the normal direction.

Because of this, generating a well optimized motion plan that is physically feasible may become problematic.

A motion plan can be constructed in different ways. A popular method is trajectory optimization, where the generation of the motion plan is formulated as an optimization problem. The resulting motion plan is a optimal trajectory. While formulating the trajectory optimization problem, several assumptions — each with their own trade-offs — can be made:

- Whole-body or simplified dynamics.
- Physical feasibility as a hard constraint or as an optimization variable.
- Constant or variable time-horizon.
- Contact phases, which describe when and which parts are in contact with the surroundings, can be predefined or included in the optimization.

The current solution available at the Biomechanical Engineering Department [5] can generate optimal trajectories for gait, for a time-horizon of a couple of seconds while taking the whole-body dynamics into account. The contact phases are predefined, such that it can plan efficient gait trajectories.

Predefining the contact phases reduces the applicability of this framework to generate optimal trajectories for tasks and environments that do not fit the predefined contact phases. This limits the advantages, such as the reactive capabilities, that MPC has to offer.

1.3 Objective: Implicit Contacts

This thesis aims to include the optimization of the contact phases, in addition to optimizing the states and controls for whole-body dynamics. Allowing the algorithm to make and break contact whenever it deems fit. This general formulation makes the algorithm able to generate motion plans for almost any task, regardless of the system dynamics or definition of optimality. In theory this could provide the patient ultimate freedom.

This thesis contributes by writing software for trajectory optimization with implicit contacts and analyzing different balance control related tasks, which could not be optimized with the previous framework. Specifically, it will demonstrate whether implicit contact methods can generate optimal motion plans for safe falling and getting up, in addition to efficient gait. All of this will be performed with the same algorithm and minimal change to the parameters. This will improve the performance of lower limb exoskeletons because it allows further development of more capable control strategies for bipedal robots.

Due to the computational cost, it is impossible to be implemented in a MPC fashion. But the results are useful for analysis, and could be used as training data for machine learning methods. It is unknown what the accuracy requirements are for a physically feasible trajectory with locomotion.

1.4 Outline of the Thesis

This thesis is organized as follows. Chapter 2 discusses the state of the art trajectory optimization methods for humanoid robotic and analysis of falling. Chapter 3 gives an overview of the topics required to understand the methods and results of this thesis. The design decisions and final formulation of the algorithm are discussed in Chapter 4. The performance of the algorithm for producing gait and falling trajectories can be found in Chapter 5. Finally the results, shortcomings, possible improvements and recommendations are discussed in Chapter 6.

Chapter 2

Related Works

This chapter will discuss different related works to get an idea of what proposed solutions already exist, and what problems still need to be solved.

2.1 Whole-Body Locomotion Planning with Direct Collocation and Explicit Contact Phases

Roos [5] describe a framework to generate efficient gait trajectories based on whole-body dynamics and known contact phases. The proposed framework splits the problem of generating efficient gait trajectories in two parts:

1. Finding footholds over time.
2. Controlling the robot to efficiently step through the footholds over time.

The goal was to solve the second part. This has been achieved by posing the problem as a nonlinear program (NLP). This was done with trapezoidal collocation by using the whole-body dynamics and linear friction cones as constraints. MuJoCo [13] was used for the dynamics and gradients, and IPOPT [14] as numerical optimizer. This produced effective gaits for different dynamics, terrains, and objective functions.

However, there were shortcomings: Contacts are not optimized, but spaced out by some other algorithm. This limits the framework to optimizing whatever footholds were predefined. This might cause decreased performance in tasks the first part was not designed to do, such as falling.

When using trapezoidal collocation, it is assumed that the dynamics of the system can be approximated to be linear between discrete points in time. This decreases the accuracy of the generated trajectories as it is basically an inaccuracy in the model.

Because of the high computational cost, the frequency at which trajectories are generated is too low for actual control as it is not possible to generate updated trajectories to adjust for disturbances and inaccuracies of the model. A possible workaround however, is generating optimal reference trajectories at a lower frequency and tracking them with a higher frequency PD-controller.

2.2 Centroidal Mixed-Integer Locomotion Planning

Koolen [8] describe a trajectory optimization approach for humanoid robot locomotion based on mixed-integer nonlinear programming (MINLP).

To strike a balance between computational costs and usable results, the planning was done for a model with reduced complexity. The configuration of the full robot was summarised as the positions of the center of mass (CoM) and end-effectors. This configuration could be used to approximate kinematic constraints, such as minimum and maximum distances between CoM and end-effector positions, but also preventing cross-over for the feet. There also were contact sequence constraints, which limit swing and stance duration, and slipping contacts. The additional contact constraints could be formulated by introducing auxiliary integer variable $\mathbf{z}$ that assumed value 1 if the contact was active, and 0 otherwise:

$$0 \leq \boldsymbol{\phi} \leq (1 - \mathbf{z})\bar{\boldsymbol{\phi}}, \tag{2.2.1}$$

$$0 \leq \boldsymbol{\lambda} \leq \mathbf{z}\bar{\boldsymbol{\lambda}}, \tag{2.2.2}$$

$$\mathbf{z} \in \{0, 1\}, \tag{2.2.3}$$

where $\boldsymbol{\phi}$ is the smallest distance between an end-effector and a contact surface, $\boldsymbol{\lambda}$ is the contact force generated by an end-effector. $\bar{\boldsymbol{\phi}}$ and $\bar{\boldsymbol{\lambda}}$ are used to indicate the upper bounds of these variables. This resulted in a mixed-integer program, which are notoriously difficult to solve. A basic way to solve this problem is by dropping the integrality constraint (2.2.3) and trying to solve it as if it was continuous:

$$\{\mathbf{z} \in \mathbb{R} \quad | \quad 0 \leq \mathbf{z} \leq 1\}. \tag{2.2.4}$$

This is called the continuous relaxation. The solution this generates is called the fractional solution. As this fractional solution likely does not satisfy the integrality constraints, it must be refined. This was achieved by, intelligently, recursively branching the fractional solution by rounding it to the nearest integer.

There are some notable drawbacks to this approach. A reduced complexity model has to be used, resulting in solutions that are more constrained than necessary. This limits this approach to only generating gait trajectories. In addition, all generated trajectories will be conservative, and thus not optimal, solutions. Finally, even with the simplified model and added constraints the performance of MINLP solvers remains unpredictable.

2.3 Centroidal Locomotion Planning through Phase-Based End-Effector Parameterization

Winkler et al. [6, 7] describe a way to optimize gait trajectories, step-timings and footholds based on centroidal-dynamics and a known sequence of contact phases.

The goal is to find a feasible control trajectory of the foot position and force, and the CoM and orientation of the robot base. The most unique part about the approach, is that for each end-effector there is a predefined sequence of contact phases. And instead of switching a phase from contact to no-contact, the phase durations can be changed. This results in a continuous landscape for the optimization algorithm, which is easier to solve.

For the dynamics a simplified centroidal model was used, and a kinematic model of a range-of-motion box with respect to the CoM was used to keep the end-effectors within the joint limits.

End-effectors in the air are not allowed to exert a force, and end-effectors in contact with the ground should not move, be at terrain height and only apply a pushing force within the friction cone. The assumption that the motion of the legs do not influence the motion of the base was made. This results in no dependencies on joint angles and their nonlinearities, but this is only

true for massless limbs.

The final result was a controller that could efficiently generate feasible trajectories. In simulation it even worked on non-flat terrain, and in practice it was successful in tracing a couple of steps on a flat ground.

However, there are a few shortcomings: It is impossible to extend this method with whole-body dynamics as it would overconstrain the problem [5]. This means that this approach would only work for robots whose limbs have very low mass compared to the base.

This method relies on a finite sequence of contact phases. It is like a budget that must be met. Too few phases result in infeasible trajectories, but too many may result in inefficient jumping at the destination.

2.4 Motion Planning through Contact-Invariant Optimization

Mordatch et al. [9] describe a flexible framework for generating trajectories which uses continuous auxiliary variables, in contrast to the integer values discussed in Section 2.2.

Essentially this method tries to find the optimal solution by minimizing a composite objective function $J(\boldsymbol{z})$ in the form

$$J(\boldsymbol{z}) = J_{\text{CI}}(\boldsymbol{z}) + J_{\text{Physics}}(\boldsymbol{z}) + J_{\text{Task}}(\boldsymbol{z}). \tag{2.4.1}$$

Where $\boldsymbol{z}$ are the decision variables, J_{CI} is the contact-invariant cost, J_{Physics} penalizes physics violations and J_{Task} specifies the task objectives.

This method introduces the continuous auxiliary contact-related variable c_i, which functions as a weight. The idea is that if some end-effector must be in contact, its corresponding c_i is high. If c_i is small, it is not important. These auxiliary variables are then implemented in both J_{CI} and J_{Physics}, such that setting all c's to zero minimizes J_{CI}, but causes contact forces to become expensive. Thus the optimal trade-off is a high c_i value if contact is of importance, and low if it is not. This composite objective function is rewritten as a quadratic program (QP) and solved with some off-the-shelf QP solver. The paper that presents the framework uses a simplified physics model where the limbs have no mass, but the method should be able to use more realistic descriptions as well, although this has not been implemented yet. This framework manages to generate trajectories for many different movements.

It is notable that J_{Physics} is included in the object function instead of the constraints. This implies that the solutions are not necessarily feasible in simulation, let alone in the real world. No error functions or practical demonstrations were described to verify the usability of this method.

2.5 Direct Trajectory Optimization of Rigid Bodies Through Contact

Posa et al. [10, 11] discuss a method to efficiently plan through non-smooth motions induced by contact to generate gait patterns.

This method relies on complementarity constraints with slack variables to simultaneously optimize contacts and smooth movements. Instead of an auxiliary variable that contains information regarding the importance of contact, a gap function $\phi(\boldsymbol{q})$ is defined that analytically describes the

distance between an end-effector and the nearest surface. Using this gap function, the following constraints can be formulated:

$$\boldsymbol{\lambda}^{\perp} \geq \mathbf{0}, \tag{2.5.1}$$

$$\boldsymbol{\phi}(\boldsymbol{q}) \geq \mathbf{0}, \tag{2.5.2}$$

$$\boldsymbol{\phi}^{\top}(\boldsymbol{q})\boldsymbol{\lambda}^{\perp} = \mathbf{0}. \tag{2.5.3}$$

Where $\boldsymbol{\lambda}^{\perp}$ is normal force. The normal force must be positive as one can only push against the ground. To provide a more tractable optimization problem the equality constraint from Eqn. 2.5.3 can be replaced with an inequality constraint with slack variable $\boldsymbol{\epsilon}$,

$$\boldsymbol{\phi}^{\top}(\boldsymbol{q})\boldsymbol{\lambda}^{\perp} \leq \boldsymbol{\epsilon}. \tag{2.5.4}$$

The optimizer runs multiple times with decreasing values for $\boldsymbol{\epsilon}$, with $\boldsymbol{\epsilon} = 0$ for the final run. When this formulation has a smooth gradient, it essentially allows the algorithm to "reason" that it must make contact to exert force, and avoid contact to move. This same method can also be used to include slip and friction forces, resulting in more realistic dynamics. Some Sequential Quadratic Programming (SQP) solver was used to solve the problem. The method seemed to performed well in planar simulations.

Besides computational costs, the need for an analytical gap function to solve contacts might not be ideal. In practice a controller would not have access to a gap function, but a height map and/or ray-casts. No error functions or practical demonstrations were described to verify the usability of this method.

2.6 Optimization of Safe Falling Trajectories for Lower Limb Exoskeletons

Masha Khalili et al. [15] developed a human-exoskeleton fall model to reduce injury to the patient.

The method uses a three link inverted pendulum in the sagittal plane, and only the hip is considered to be a contact point. An additional constraint that limits the joint torques was added. The objective is to minimize the angular momentum and total linear velocity of the hip at impact.

The resulting impact velocities were similar to real human falls. But the approach could be improved by using a more accurate model: free floating frame, multiple contacts, and forces instead of velocities.

2.7 Summary of Trajectory Optimization Methods for Planning Through Contacts

Roos [5] demonstrated a state of the art framework for whole-body dynamics with hybrid direct collocation, but failed to properly track the planned trajectory in simulation.

Koolen [8] used a centroidal model to plan a trajectory for the CoM. For this trajectory it searched for contacts using kinematic constraints and mixed-integer programming.

Winkler et al. [6, 7] set a fixed sequence of contacts and varied the duration of these contact phases. The contacts were kinematically constrained with respect to the CoM of the centroidal

Table 2.1: Overview of trajectory optimization methods through contacts.

Method	Contact Model	Contact Optimization	Dynamics	Physical Feasibility
[5]	Rigid	Fixed timing	Whole-body	Failed in MPC environment
[8]	Rigid	Partially[1]	Simplified	Untested
[6, 7]	Rigid	Partially[2]	Simplified	Simulation and practice
[9]	Smoothed	Optimized	Simplified[3]	Untested
[10–12]	Rigid	Optimized	Whole-body	Untested

[1] Secondary with kinematic constraints.
[2] Fixed number of phases with kinematic constraints.
[3] Possibility to extend to whole-body dynamics.

model, which was used for the dynamics.

Mordatch et al. [9] constructed a smoothed contact-invariant cost function, similar to complementarity constraints, that penalizes forces when not in contact with a surface. With simplified dynamics as a cost function, a composite cost function was formulated as a quadratic program.

Posa et al. [10, 11] and Patel et al. [12] exploited complementarity constraints to formulate a contact model for whole-body dynamics. Traditional direct collocation methods were used to pose the NLP.
An overview of these optimization methods for planning through contacts is shown in Table 2.1.

A drawback that all discussed papers have is that none of them discuss the physical feasibility between collocation points with error functions, which are available for all direct methods. Although the physical demonstration of Winkler et al. [6, 7] proves that the accuracy and computational cost are sufficient for an MPC implementation, the required feedback control, deviance from the original trajectory and error functions could help as a reference.

As the long-term goal is to have one method to generate motion plans for any task, it makes sense to turn to the methods of Posa et al. [10, 11] and Mordatch et al. [9]. These methods have proven to be able to adapt their contact phases to a multitude of tasks, whereas other methods have only been demonstrated for the generation of gait.

Due to the inability to include constraints in the method of Mordatch et al. [9], it is difficult to enforce realistic behaviour. Therefore it was decided to apply the implicit contact method as described by Posa et al. [10, 11].

Chapter 3

Background

This chapter will attempt to get the reader to a level of understanding on several topics that are deemed relevant for understanding the methods and results.

To fully understand the contents of this thesis, a good grasp of optimization (3.2), dynamics (3.3), numerical integration (3.4) and direct collocation (3.5) is required.

3.1 Definitions and Conventions

Throughout this thesis several definitions and conventions are used, this section will give a short overview. The definitions can be found in Table 3.1.

Conventions:
Vectors are written in bold lowercase: $\boldsymbol{a}$.
Matrices are written in bold uppercase: $\mathbf{A}$.
Subscript is often used to indicate indices: $\boldsymbol{a} = [a_0, a_1, a_2]$.
For brevity, arguments of a function may be omitted: $f(a_0, b_0, c_0) = f_0$.
Dots above a variable indicate its derivative with respect to time: $\dot{a} = \frac{da}{dt}$.

Table 3.1: Overview of definitions used throughout this thesis.

Definition	**Meaning**
Tractable	A tractable NLP has numerical properties that are beneficial for convergence.
Feasibility	A NLP is feasible when the constraint violations are below the threshold of the solver. For collocation this does not mean the generated trajectory is physically feasible, as the dynamics error between these discrete points may still be too large to be executed in practice.
Physical feasibility	A generated trajectory is physically feasible if it is realistic enough to be executed in practice. This can be estimated by analysing the deviation between the generated trajectory and trajectory prescribed by the equations of motion with the generated states and controls.
Knot point	Point in time that is used in direct collocation to formulate the decision variables and divide the solution in segments.
Segment	The continuous time between two knot points.
Collocation point	Point on a segment that is constrained by the approximation of the dynamics using some quadrature method.

3.2 Optimization

Mathematical optimization is at the core of direct collocation [16–18], so the knowledge of this section will be applied in Section 3.5.

Mathematical optimization is a field that is concerned with finding the minimum value of a function, given a set of constraints. A general formulation of an optimization problem is:

$$\begin{aligned} \underset{\boldsymbol{z}}{\text{minimize}} \quad & J(\boldsymbol{z}) && \text{(3.2.1a)} \\ \text{subject to} \quad & \boldsymbol{z}^{\text{LB}} \leq \boldsymbol{z} \leq \boldsymbol{z}^{\text{UB}}, && \text{(3.2.1b)} \\ & \mathbf{f}(\boldsymbol{z}) = \mathbf{0}, && \text{(3.2.1c)} \\ & \mathbf{g}(\boldsymbol{z}) \leq \mathbf{0}. && \text{(3.2.1d)} \end{aligned}$$

Where $\boldsymbol{z}$ contains the decision variables, which values can be varied to find the lowest point of $J(\boldsymbol{z})$ that satisfies all constraints. The constraints can be generalized to boundary, inequality and equality constraints. Boundary constraints (3.2.1b) set a lower and upper bound ($\boldsymbol{z}_{\text{LB}}$ and $\boldsymbol{z}_{\text{UB}}$ respectively) to the decision variables. All values between $-\infty$ and ∞ may be chosen as boundaries. Equality (3.2.1c) and inequality (3.2.1d) constraints are both functions of $\boldsymbol{z}$.

Historically, constrained optimization problems have also been called programs. A special name for programs with nonlinear terms in either its $J(\boldsymbol{z})$, $\mathbf{f}(\boldsymbol{z})$, or $\mathbf{g}(\boldsymbol{z})$ is a nonlinear program (NLP).

3.2.1 Iterative Optimization Methods

Iterative optimization methods start out with an initial guess, $\boldsymbol{z}_0$, of what the solution might be. From $\boldsymbol{z}_0$ the solver iterates towards a solution until it converges to a (local) minimum.

Different methods exist for converging to a solution. A simple — but effective — method of finding the lowest value of $J(\boldsymbol{z})$ is going in the direction that has the steepest descending slope, while conforming to the constraints. These methods are referred to as gradient-based methods. The slope can be calculated with

$$\frac{\partial J(\boldsymbol{z})}{\partial \boldsymbol{z}} = \left[\frac{\partial J(\boldsymbol{z})}{\partial z_0}, \cdots, \frac{\partial J(\boldsymbol{z})}{\partial z_n} \right]. \tag{3.2.2}$$

This can be done both numerically or analytically. The numerical approach is easier to implement, but requires more function evaluations and therefore has a higher computational cost. Implementing the analytical approach in an efficient manner requires more effort, but the resulting gradients will be up to machine precision with fewer function evaluations during runtime.

A similar approach may be used to satisfy the equality and inequality constraints, although these are usually vector valued functions. This results in a Jacobian with more than a single row.

What is important to note, is that in order to use gradient-based methods the objective and constraint functions must be consistent. Functions are consistent if the sequence of arithmetic operations performed between different function calls is identical [16]. A more intuitive explanation is that the function must be deterministic, have no logical branches, and have an output that varies smoothly with the inputs [18]. If $\frac{\partial J(\boldsymbol{z})}{\partial \boldsymbol{z}}$, $\frac{\partial \mathbf{f}(\boldsymbol{z})}{\partial \boldsymbol{z}}$, or $\frac{\partial \mathbf{g}(\boldsymbol{z})}{\partial \boldsymbol{z}}$ is ill-defined, then the chance of converging to a feasible solution becomes very small [17, 18].

3.2.2 Complementarity Constraints

Complementarity constraints can be used as a method to replace logical statements with consistent equations. Usually complementarity constraints enforce that:

$$ab = 0, \tag{3.2.3}$$
$$a \geq 0, \tag{3.2.4}$$
$$b \geq 0. \tag{3.2.5}$$

As Eqn. 3.2.3 is a difficult constraint for most solvers, this thesis uses a more tractable [10, 11] method, where it is also expressed as an inequality constraint:

$$ab \leq 0. \tag{3.2.6}$$

A succinct notation for complementarity constraints is often used:

$$0 \leq a \perp b \geq 0. \tag{3.2.7}$$

3.2.3 Solvers

A multitude of solvers exist. Among these solvers there is no single best choice. Depending on the problem that must be solved and which software can be used a choice should be made. Two solvers were used for this thesis; `fmincon` (MATLAB [19]), and IPOPT [14] (C++), which also has a MATLAB interface [20].

When a license is available, `fmincon` is easy to use and it supports different algorithms (`interior-point`, `trust-region-reflective`, `sqp` and `active-set`), which is useful for prototyping.

IPOPT is a popular interior-point solver that is open source. The main advantages it has over `fmincon` are that no licenses are required and that it is incredibly fast.

3.3 Dynamics

In this thesis, dynamics refers to (rigid) multibody dynamics, which is a set of Ordinary Differential Equations (ODEs) that choreograph the translational and rotational displacements of interconnected rigid bodies, given the applied forces. They are often called Equations of Motion (EoM), and presented as

$$\mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \mathbf{B}\boldsymbol{u} + \mathbf{J}^{\top}(\boldsymbol{q})\boldsymbol{\lambda}, \tag{3.3.1}$$

or similar [21]. In Eqn. 3.3.1 $\boldsymbol{q}$ contains the generalized coordinates, which are a set of variables to uniquely describe the configuration of the system. The mass matrix, $\mathbf{M}(\boldsymbol{q})$, describes the translational and rotational inertias of the system. $\mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}$ contains fictitious forces such as Coriolis and centripetal forces. $\boldsymbol{g}(\boldsymbol{q})$ contains the potential forces, in a system without springs these are the forces caused by gravity. $\mathbf{B}$ is called the selection matrix, it determines which generalized coordinates of the system may be actuated directly with control vector $\boldsymbol{u}$. $\mathbf{J}(\boldsymbol{q})$ is the contact jacobian, which relates the contact forces $\boldsymbol{\lambda}$ to $\boldsymbol{q}$.

These EoM can be derived and verified (Appendix A) using multiple methods. These EoM

can be rewritten as

$$\ddot{\boldsymbol{q}} = \mathbf{M}^{-1}(\boldsymbol{q}) \underbrace{\left(\mathbf{B}\boldsymbol{u} + \mathbf{J}^{\top}(\boldsymbol{q})\boldsymbol{\lambda} - \mathbf{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{g}(\boldsymbol{q})\right)}_{\mathbf{F}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}, \boldsymbol{\lambda})}, \tag{3.3.2}$$

$$\ddot{\boldsymbol{q}} = \mathbf{M}^{-1}(\boldsymbol{q})\mathbf{F}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}, \boldsymbol{\lambda}), \tag{3.3.3}$$

allowing exact calculations of the accelerations, $\ddot{\boldsymbol{q}}$, given the current $\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}$ and $\boldsymbol{\lambda}$. Here $\mathbf{F}$ are all the generalized forces acting on the system.

For the sake of readability and programming purposes, these EoM are rewritten to a system of nonlinear first order state equations

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} \\ \dot{\boldsymbol{q}} \end{bmatrix}, \quad \dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{q}} \\ \ddot{\boldsymbol{q}} \end{bmatrix}, \tag{3.3.4}$$

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}), \tag{3.3.5}$$

and used as such from now on. Here $\boldsymbol{x}$ is called the state of the system.

3.3.1 Contact Dynamics

Accurately modelling the dynamics of contact requires more care than modelling smooth movements. The dynamics of contact for rigid body systems can be modelled with different methods [21], this section only covers the general notation and equations required to model contact.

For modelling contact dynamics, certain notations can help. Most of the contact models indicate the distance between a rigid body and a contact surface with a variable, called the gap and denoted as $\boldsymbol{\phi}$ from now on. Given that the exact locations of the contact surfaces are known, the gap may be expressed as a function of the generalized coordinates: $\boldsymbol{\phi}(\boldsymbol{q})$. The contact dynamics should avoid penetration of the contact surfaces, so they should only be in effect during contact:

$$\boldsymbol{\phi}(\boldsymbol{q}) \leq \mathbf{0}. \tag{3.3.6}$$

The ground reaction forces (GRF) are expressed with $\boldsymbol{\lambda}$ in vector:

$$\boldsymbol{\lambda} = \begin{bmatrix} \boldsymbol{\lambda}^{\perp} \\ \boldsymbol{\lambda}^{\parallel} \end{bmatrix}. \tag{3.3.7}$$

Where $\perp$ stands for perpendicular, and $\parallel$ for parallel to the surface. Accompanying these forces is the stacked Jacobian matrix:

$$\mathbf{J}(\boldsymbol{q}) = \begin{bmatrix} \mathbf{J}^{\perp}(\boldsymbol{q}) \\ \mathbf{J}^{\parallel}(\boldsymbol{q}) \end{bmatrix}. \tag{3.3.8}$$

Where $\mathbf{J}^{\perp}(\boldsymbol{q})$ projects $\boldsymbol{\lambda}^{\perp}$, and $\mathbf{J}^{\parallel}(\boldsymbol{q})$ projects $\boldsymbol{\lambda}^{\parallel}$ onto $\boldsymbol{q}$. Friction cones should be enforced at all times, and are described as:

$$\left\| \boldsymbol{\lambda}^{\parallel} \right\| \leq \mu \boldsymbol{\lambda}^{\perp}. \tag{3.3.9}$$

Where μ is the friction coefficient. Contacts can be divided in sticking and slipping contacts, sticking contacts have no velocity parallel to the contact surface and therefore do not need any additional friction forces. Slipping contacts however, do have a velocity parallel to the contact

surface,

$$\mathbf{J}^{\parallel}(\boldsymbol{q})\dot{\boldsymbol{q}} \neq 0, \tag{3.3.10}$$

and therefore a friction force opposite to the slip direction and proportional to the normal force:

$$\boldsymbol{\lambda}^{\parallel} = -\mu\boldsymbol{\lambda}^{\perp}\frac{\mathbf{J}^{\parallel}(\boldsymbol{q})\dot{\boldsymbol{q}}}{\left\|\mathbf{J}^{\parallel}(\boldsymbol{q})\dot{\boldsymbol{q}}\right\|}. \tag{3.3.11}$$

3.4 Numerical Integration

This section will give a short introduction of numerical methods for integrating ODEs, these methods can be used to model a mechanical system with the EoM shown in Section 3.3.

Using time-stepping methods these EoM can be solved numerically, giving us an estimate of how the state of the system will change over time. There are explicit and implicit methods; explicit methods use y_k (current point in time) to calculate y_{k+1} (next point in time), implicit methods use both y_k and y_{k+1} to calculate y_{k+1}. The simplest approach is an explicit first-order method called Euler-forward. Using the equations

$$\boldsymbol{f}_k = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\lambda}_k), \tag{3.4.1a}$$

$$h_k = t_{k+1} - t_k, \tag{3.4.1b}$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h_k\boldsymbol{f}_k, \tag{3.4.1c}$$

the state of the system at the next discrete point in time can be calculated. The Euler-forward method is easy to implement, but it has a low numerical stability — especially for stiff ODEs. Generally, a stiff ODE means that the system includes terms that vary rapidly, and that an impractically small step size is required to appropriately approximate it. When a larger step size is used, the dynamics become numerically unstable. To combat this, higher-order or implicit methods may be used. No higher-order integration methods are used in this thesis, as the implicit methods used sufficed.

Implicit methods are key to understanding trapezoidal collocation — the integration scheme mainly used in this thesis. Next to the explicit Euler method, an implicit Euler method, called Euler-backward, also exists. Using the equation

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h_k\boldsymbol{f}_{k+1}, \tag{3.4.2}$$

the state of the system at the next discrete point in time can be calculated. Implicit methods such as Euler-backward are much better suited at solving stiff problems, because larger steps can be made while maintaining numerical stability. However, there is a small problem: $\boldsymbol{x}_{k+1}$ has to be known in order to calculate $\boldsymbol{x}_{k+1}$ itself. Because of this, additional computation is usually required for implicit methods, when compared to explicit methods. There are also other methods to integrate ODEs, which will be covered in Section 3.5.

3.5 Direct Collocation

This section will discuss what direct collocation is, and how it can be applied to optimize the control of dynamical systems.

3.5.1 What is Direct Collocation?

Direct collocation is a method to numerically solve ODEs by directly transcribing the state and control as a candidate solution, which are discrete decision variables. The transcription from continuous to discrete is indicated with $\rightarrow$.

$$\boldsymbol{x}(t) \rightarrow \mathbf{X} = [\boldsymbol{x}_0, \boldsymbol{x}_k, \cdots, \boldsymbol{x}_N], \tag{3.5.1}$$

$$\boldsymbol{u}(t) \rightarrow \mathbf{U} = [\boldsymbol{u}_0, \boldsymbol{u}_k, \cdots, \boldsymbol{u}_N], \tag{3.5.2}$$

these decision variables can be chosen "freely" at discrete points in time:

$$t \rightarrow \boldsymbol{t} = [t_0, t_k, \cdots, t_N]. \tag{3.5.3}$$

These discrete points in time are called knot points, and the time between knot points are called segments (N is used to indicate the number of segments). Collocation constraints are applied to collocation points, which may or may not coincide with the knot points, on segments such that the solution adheres to the dynamics prescribed by the ODEs at these collocation points. Where these collocation points lie on a segments is dependent on the collocation method used.

Collocation constraints try enforce that the change of state between two knot points is equal to the integral of the dynamics between those two points [18]:

$$\int_{t_k}^{t_{k+1}} \dot{\boldsymbol{x}} dt = \int_{t_k}^{t_{k+1}} \boldsymbol{f} dt. \tag{3.5.4}$$

The integral of the dynamics between two knot points is approximated with some quadrature method.

This formulation of the problem makes it possible to solve a set of ODEs, but the system is not being controlled optimally yet. To achieve this, an objective function — specifying what is meant with optimal — and additional constraints can be added. To solve the resulting NLP, it is handed over to an optimization solver which will then try to minimize the objective function while satisfying the constraints, which should result in a (locally) optimal solution. This solution not only consists out of the optimal sequence of states, but also the optimal sequence of control inputs.

3.5.2 Collocation Constraints for Euler-backward

As discussed before, the change of state between two states must be equal to the integral of the dynamics. From Equations 3.4.2 and 3.5.4 we arrive at Eqn. 3.5.5

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_k = h_k \boldsymbol{f}_{k+1}. \tag{3.5.5}$$

This collocation constraint can then be rewritten as an equality constraint in an optimization problem:

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_k - h_k \boldsymbol{f}_{k+1} = 0. \tag{3.5.6}$$

Because the decision variables are always "known", no additional computation is required for implicit methods. And as stated in Section 3.4, implicit methods are numerically more stable and allow for larger time steps and thus lower computational cost.

3.5.3 Intepolation between collocation points

The solution of a NLP posed with direct collocation are matrices containing the optimal states and controls at the knot points. Those would be useful by themselves, but there is more information that can be extracted from the solution.

Each collocation method has its own interpolation method, a piecewise polynomial to calculate the states and controls between the knot points. It is the interpolation function that can be used for both feed forward control and reference tracking. The interpolation polynomial is one order higher than the quadrature method used to construct the collocation constraints. This means that if — for control purposes — it is desirable to have n-times differentiable trajectories, the quadrature method should be chosen accordingly.

The derivation and interpolation function of Euler collocation and other collocation methods are described in Appendix B.

Chapter 4

Method

This chapter will discuss how the NLP was formulated, and explain the design decisions that were made.

4.1 Trajectory Optimization Technique

As direct collocation is not the only trajectory optimization technique, it makes sense to elaborate why it is used.

- Indirect methods were initially developed for applications where accuracy is critical [22], such as aeronautics and astrodynamics. However, the region of convergence is smaller than with direct methods, meaning that an accurate initial guess is more important [23], which is difficult for dynamics with frequent discrete events. Additional (adjoint) variables — which are not present in direct methods — must also be initialized, causing further complications [16].

- Shooting methods integrate the dynamics by simulation instead of approximation by a quadrature. Although useful for problems with complicated dynamics and simple controls [23], neither the dynamics nor controls of locomotion are simple. It is also difficult to apply path constraints [23], which will make avoiding penetration of contacts particularly difficult.

- Differential dynamic programming also simulates the system forward in time and optimizes based on the result of that simulation [18]. In contrast to shooting methods and due to the time dependent nature of the dynamics, the problem is broken up into sub-problems. Each sub-problem is solved optimally, resulting in an optimal solution for the original problem. A big benefit of differential dynamic programming is that it results in an optimal policy instead of a trajectory, guaranteeing optimal behaviour with disturbances. In theory differential dynamic programming is not well suited for handling complex dynamic systems [5].

- Gradient free methods are more general optimization techniques and therefore a bit of an odd addition to this list. In theory they are interesting as they do not experience the same issues with discrete event as gradient based optimization does. Experiments were performed with the genetic algorithm (as implemented by MATLAB), but it showed poor convergence for relatively simple problems. Evolution strategies such as CMA-ES [24] were also considered, but because of the inability to include nonlinear constraints it would be difficult to guarantee physically feasible solutions.

4.2 Contact Handeling

The main objective was to include the optimization of contacts, while simultaneously optimizing the smooth movements. The implicit contact method of Posa et al. [10, 11] was chosen as it seemed to offer the best balance between reliable convergence and realistic trajectories. The MINLP method [8] seems promising, but the performance is too unreliable. Changing phase durations [6, 7] works well, but requires lightweight limbs. As this is not an option for exoskeletons, the generated trajectories would not be feasible in reality. Mordatch et al. [9] include the contacts and dynamic constraints in the objective function, and can therefore not guarantee real world feasibility.

The difficulty of including contact dynamics in gradient-based optimization, is that seemingly instantaneous events must be modelled with functions that are consistent. This can be achieved by splitting the inconsistent contact dynamics into multiple functions that are consistent. One way of achieving this is with complementarity constraints (Eqn. 2.5.4).

However, this exact formulation introduces the problem that contact points can only start to decelerate at the moment of impact. But numerical optimization happens at discrete moments in time, meaning that objects making contact will still have a negative velocity with respect to the contact surface, resulting in penetration. At the cost of realism, GRF are allowed a single discrete step in time before impact. This can be represented as a complementarity constraint,

$$\mathbf{0} \leq \boldsymbol{\phi}^{\top}(\boldsymbol{q}_{k+1}) \perp \boldsymbol{\lambda}_k^{\perp} \geq \mathbf{0}, \tag{4.2.1}$$

where k indicates the current knot point. The discontinuous behaviour of contact gets approached as the density of knot points over some time interval increases.

While Posa et al. [10, 11] stated that adding a slack variable (Eqn. 2.5.4) to the complementarity constraints improved performance, it has not been used as it only proved to increase the probability of convergence to local minima.

To include friction in gradient based optimization methods, the inconsistent friction equations (3.3.9 and 3.3.11) were rewritten as multiple equations that are consistent. To do so, $\boldsymbol{\lambda}^{\parallel}$ was split into additional variables, $\boldsymbol{\lambda}^{\parallel-}$ and $\boldsymbol{\lambda}^{\parallel+}$ such that:

$$\boldsymbol{\lambda}^{\parallel-} \geq \mathbf{0}, \tag{4.2.2}$$

$$\boldsymbol{\lambda}^{\parallel+} \geq \mathbf{0}, \tag{4.2.3}$$

$$\boldsymbol{\lambda}^{\parallel} = \boldsymbol{\lambda}^{\parallel+} - \boldsymbol{\lambda}^{\parallel-}. \tag{4.2.4}$$

Where the '$-$' and '$+$' indicate the forces perpendicular in both directions with respect to $\boldsymbol{\lambda}^{\perp}$ (Figure 4.1).

Additionally, slack variable $\boldsymbol{\gamma}$ was added, which is identical to the absolute velocity of a contact point parallel to the contact surface. With these additional variables, the equations describing the contact dynamics can be represented as consistent complementarity constraints:

$$\mathbf{0} \leq \mu\boldsymbol{\lambda}^{\perp} - \boldsymbol{\lambda}^{\parallel-} - \boldsymbol{\lambda}^{\parallel+} \perp \boldsymbol{\gamma} \geq \mathbf{0}, \tag{4.2.5}$$

$$\mathbf{0} \leq \boldsymbol{\gamma} - \left[\mathbf{J}^{\parallel}(\boldsymbol{q})\right]^{\top} \dot{\boldsymbol{q}} \perp \boldsymbol{\lambda}^{\parallel+} \geq \mathbf{0}, \tag{4.2.6}$$

$$\mathbf{0} \leq \boldsymbol{\gamma} + \left[\mathbf{J}^{\parallel}(\boldsymbol{q})\right]^{\top} \dot{\boldsymbol{q}} \perp \boldsymbol{\lambda}^{\parallel-} \geq \mathbf{0}. \tag{4.2.7}$$

Where Eqn. 4.2.5 ensures that the GRF stays within the friction cone, and for slipping contact

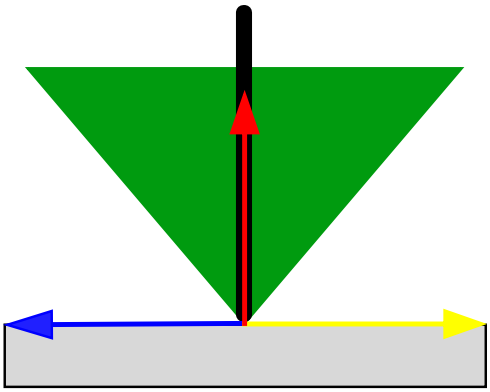


Figure 4.1: The modelled ground reaction forces. The black line represents a limb with a contact point at the bottom. The gray rectangle represents a contact surface. The red vector pointing upwards represents the normal force $\left(\lambda^{\perp}\right)$. The blue vector pointing to the left represents the negative parallel friction force $\left(\lambda^{\parallel -}\right)$. The yellow vector pointing to the right represents the positive parallel friction force $\left(\lambda^{\parallel +}\right)$. The green triangle represents the friction cone, where the linear combination of all three vectors should remain within.

lie on one of the edges of the friction cone. Eqn. 4.2.6 enforces that in case of slip in the positive direction, no force in the direction of slip may be applied. Together with Eqn. 4.2.7 this ensures that slipping contacts always have a friction force in the opposite direction.

4.3 NLP Formulation

By combining the contact constraints with boundary and collocation constraints and an objective function, an optimization problem can be formulated:

$$
\begin{aligned}
\underset{\boldsymbol{x},\boldsymbol{u},\boldsymbol{\lambda},\boldsymbol{\gamma}}{\text{minimize}} \quad & \sum_{k=1}^{N} \left(w_u \boldsymbol{u}_k^{\top} \boldsymbol{u}_k + w_\lambda \boldsymbol{\lambda}_k^{\top} \boldsymbol{\lambda}_k \right) && \text{(4.3.1a)} \\
\text{subject to} \quad & \boldsymbol{x}^{\mathrm{LB}} \leq \boldsymbol{x}_k \leq \boldsymbol{x}^{\mathrm{UB}}, && \text{(4.3.1b)} \\
& \boldsymbol{u}^{\mathrm{LB}} \leq \boldsymbol{u}_k \leq \boldsymbol{u}^{\mathrm{UB}}, && \text{(4.3.1c)} \\
& \boldsymbol{x}_1^{\mathrm{LB}} \leq \boldsymbol{x}_1 \leq \boldsymbol{x}_1^{\mathrm{UB}}, && \text{(4.3.1d)} \\
& \boldsymbol{x}_N^{\mathrm{LB}} \leq \boldsymbol{x}_N \leq \boldsymbol{x}_N^{\mathrm{UB}}, && \text{(4.3.1e)} \\
& \boldsymbol{x}_{k+1} - \boldsymbol{x}_k - \frac{h_k}{2} \left(\boldsymbol{f}_k + \boldsymbol{f}_{k+1} \right) = \boldsymbol{0}, && \text{(4.3.1f)} \\
& \boldsymbol{0} \leq \boldsymbol{\phi}^{\top}(\boldsymbol{q}_{k+1}) \perp \boldsymbol{\lambda}_k^{\perp} \geq \boldsymbol{0}, && \text{(4.3.1g)} \\
& \boldsymbol{\phi}^{\top}(\boldsymbol{q}_N) \boldsymbol{\lambda}_N^{\perp} \leq \boldsymbol{0}, && \text{(4.3.1h)} \\
& \boldsymbol{0} \leq \mu \boldsymbol{\lambda}_k^{\perp} - \boldsymbol{\lambda}_k^{\parallel -} - \boldsymbol{\lambda}_k^{\parallel +} \perp \boldsymbol{\gamma}_k \geq \boldsymbol{0}, && \text{(4.3.1i)} \\
& \boldsymbol{0} \leq \boldsymbol{\gamma}_k - \left[\mathbf{J}^{\parallel}(\boldsymbol{q}_k) \right]^{\top} \dot{\boldsymbol{q}}_k \perp \boldsymbol{\lambda}_k^{\parallel +} \geq \boldsymbol{0}, && \text{(4.3.1j)} \\
& \boldsymbol{0} \leq \boldsymbol{\gamma}_k + \left[\mathbf{J}^{\parallel}(\boldsymbol{q}_k) \right]^{\top} \dot{\boldsymbol{q}}_k \perp \boldsymbol{\lambda}_k^{\parallel -} \geq \boldsymbol{0}. && \text{(4.3.1k)}
\end{aligned}
$$

Where $\boldsymbol{x}$, $\boldsymbol{u}$, $\boldsymbol{\lambda}$, $\boldsymbol{\gamma}$ are the decision variables that will be adjusted to minimize the objective function (4.3.1a), while satisfying the boundary (4.3.1b to 4.3.1e), equality (4.3.1f), and comple-

mentarity (4.3.1g to 4.3.1k) constraints. See Appendix C for the formulation without succinct notation.

For gait generation the objective is to minimize the joint torques and GRF squared. Minimization of joint torques results in human-like motions, and minimization of the GRF makes taking multiple steps more appealing. Weights, w_u and w_λ, were added such that the priority for the objective function may be tuned. The actual objective, getting from point a to point b, was omitted from the objective function and included in the boundary constraints on purpose, as it results in a more tractable NLP [17].

The joint angles and velocities, and torques are bounded by 4.3.1b and 4.3.1c respectively. The initial and desired states are bounded by 4.3.1d and 4.3.1e respectively. As contact points may only push against the ground, the GRF are bounded to positive values by (4.3.1g, 4.3.1j and 4.3.1k). The final boundary constraint is the slack variable, which is also bounded to all positive values by (4.3.1i).

For the equality constraints (4.3.1f), there are only the collocation constraints. Trapezoidal collocation was used as it has been proven to be reliable [5, 18, 22] and simple to implement.

Finally, the inequality constraints specify that the gap function must be positive and ensure that normal forces are applied if and only if the gap at the next collocation point is equal to zero 4.3.1g. Eqn. 4.3.1h was added to also constrain the normal force at the final collocation point. Equations 4.3.1i, 4.3.1j and 4.3.1k together ensure that the GRF stays within the friction cone, and that the friction force during slip is in the opposite direction.

4.4 9 DOF Walker

For the dynamic model, a 9 DOF bipedal humanoid figure with lower limb exoskeleton in the sagittal plane (Figure 4.2), from [25], was used. The kinematics, mass matrix, fictitious forces matrix, potential forces vector and model parameters were derived and supplied by Arvid Q.L. Keemink and Ander Vallinas Prieto.

The model consists of a free-floating-base, located at the CoM of the upper body. All other bodies — feet, and upper and lower legs — were defined with angles relative to their parent bodies. The floating base cannot be actuated, but all joints may be actuated within the torque limitations.

In order to allow for realistic trajectory optimization of both gait and falling, a multitude of eligible contact points were required. Contact points were placed on the head; and hip, knee, ankle, heel and toe of both legs, for a total of 11 contact points. At each of these contact points a horizontal and vertical force may be applied. An overview of the used model parameters is given in Table 4.1.

4.5 Software Implementation

This section discusses the final design of the algorithm.
Both `fmincon` and IPOPT are supported, on average the computational cost is reduced tenfold by using IPOPT.

The time span over which the trajectories will be optimized is defined at the start, and will not be optimized.

For the number of collocation points, it is possible to insert a vector with n integer values, this will cause the optimization algorithm to run n times using the interpolated previous result as the new $\boldsymbol{z}_0$.

Table 4.1: Overview of used model parameters.

Parameter	**Value**	**Unit**
Gravitational acceleration	9.81	m s^{-2}
Friction coefficient	1	[−]
Length upper body	0.8562	m
Mass upper body	70.9819	kg
Inertia upper body	16.3196	kg m^2
CoM x in body frame	-0.0398	m
CoM y in body frame	0.4615	m
Length upper leg	0.42	m
Mass upper leg	12.11	kg
Inertia upper leg	0.9302	kg m^2
CoM x in rotated hip frame	-0.0027	m
CoM y in rotated hip frame	0.1876	m
Length lower leg	0.433	m
Mass lower leg	7.0238	kg
Inertia lower leg	0.3794	kg m^2
CoM x in rotated knee frame	0.0044	m
CoM y in rotated knee frame	0.152	m
Length heel	0.0583	m
Length toe	0.1467	m
Height foot	0.0908	m
Mass foot	1.8762	kg
Inertia foot	0.2657	kg m^2
CoM x in rotated ankle frame	-0.0685	m
CoM y in rotated ankle frame	0.0287	m
Maximum hip torque	100	N m
Maximum knee torque	50	N m
Maximum ankle torque	100	N m
Minimum angle hip	0.8π	rad
Maximum angle hip	$\frac{5}{3}\pi$	rad
Minimum angle knee	-0.5π	rad
Maximum angle knee	0	rad
Minimum angle ankle	-0.25π	rad
Maximum angle ankle	0.25π	rad

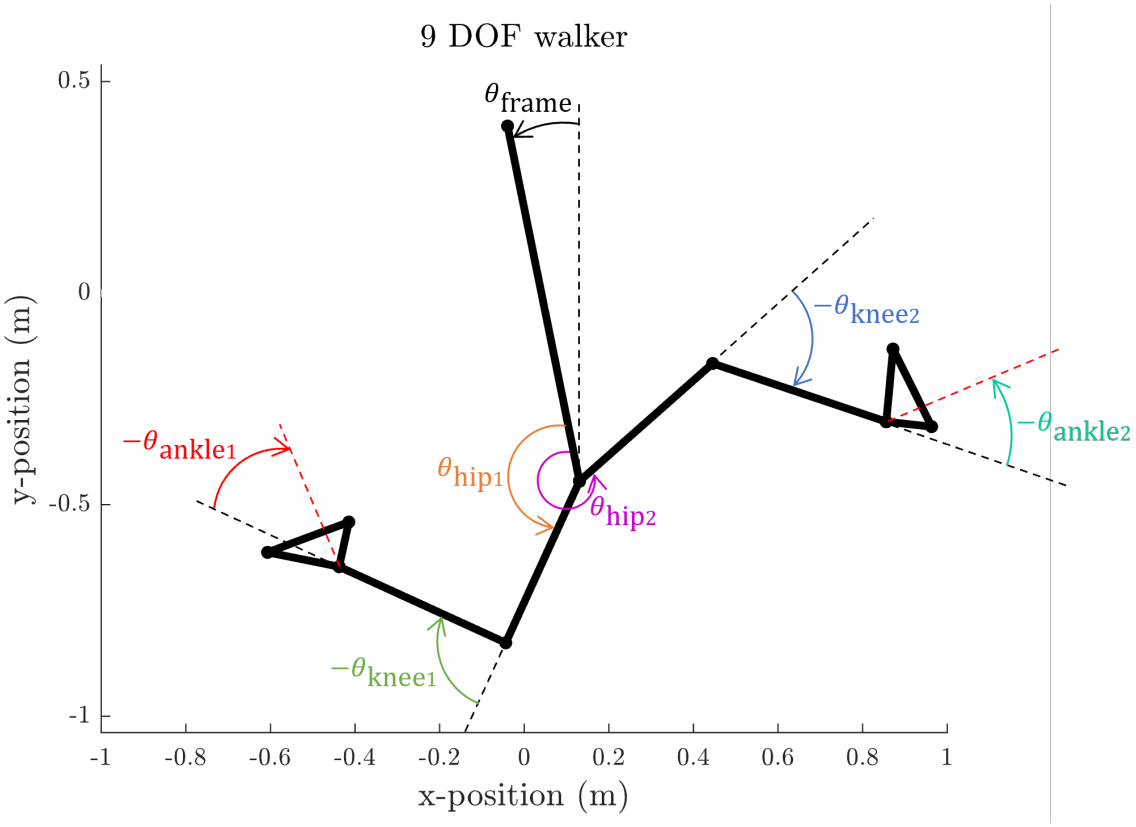


Figure 4.2: Model of the 9 DOF biped portraying how the joint angles are defined. Note that the first leg is configured with the minimum angles. The second leg is configured with the maximum hip and ankle angles.

The slack variable for the complementarity constraints (Eqn. 2.5.4) is included as described by Posa et al. [10], but set to zero by default as it only increased convergence to local minima.

By default necessary functions, such as the objective $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda})$, gap $\boldsymbol{\phi}(\boldsymbol{q})$ and parallel contact Jacobian $\mathbf{J}^{\parallel}(\boldsymbol{q})$ are defined for the walker model (Section 4.4). The objective and dynamics functions can be changed easily, but the gradients and all functions required for contact should also be defined. So these should be derived by the user. Models for a pendulum, monoped, and walker with pin-feet are also available.

An efficient method for calculating the gradients of the dynamics can be found in Appendix D. The initial and desired states and boundaries for all decision variables can be modified by the user.

Unless a specific initialization is specified, linear interpolation between the initial and desired state is performed. Additionally, a small amount of white noise — with a user specified seed — was added to decision variables included in the objective function, as this should improve solver performance [17] by avoiding infeasible minima.

After this, the problem is packed into one vector:

$$\boldsymbol{z} = [\boldsymbol{z}_0 \dots \boldsymbol{z}_k \dots \boldsymbol{z}_N]^\top, \tag{4.5.1}$$

$$\boldsymbol{z}_k = [\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{\lambda}_k, \boldsymbol{\gamma}_k]^\top. \tag{4.5.2}$$

Then the problem gets solved with whichever solver that was picked.

After it has been solved, the solutions gets interpolated with spline functions that correspond with the collocation method. As trapezoidal collocation was used, the decision variables were interpolated with:

$$\boldsymbol{x}(\tau) = \boldsymbol{x}_k + \boldsymbol{f}_k\tau + \frac{\tau^2}{2h_k}\left(\boldsymbol{f}_{k+1} - \boldsymbol{f}_k\right), \tag{4.5.3}$$

$$\boldsymbol{u}(\tau) = \boldsymbol{u}_k + \frac{\tau}{h_k}\left(\boldsymbol{u}_{k+1} - \boldsymbol{u}_k\right), \tag{4.5.4}$$

$$\boldsymbol{\lambda}(\tau) = \boldsymbol{\lambda}_k + \frac{\tau}{h_k}\left(\boldsymbol{\lambda}_{k+1} - \boldsymbol{\lambda}_k\right), \tag{4.5.5}$$

$$\dot{\boldsymbol{x}}(\tau) = \boldsymbol{f}_k + \frac{\tau}{h_k}(\boldsymbol{f}_{k+1} - \boldsymbol{f}_k). \tag{4.5.6}$$

Where τ is the continuous time between two knot points. The derivation of these splines can be found in Appendix B.

With these interpolation functions and the EoM, a function that describes the deviation from the EoM can be defined. This will be referred to as the error in the dynamics:

$$\boldsymbol{\varepsilon}(\tau) = \boldsymbol{f}(\boldsymbol{x}(\tau), \boldsymbol{u}(\tau), \boldsymbol{\lambda}(\tau)) - \dot{\boldsymbol{x}}(\tau). \tag{4.5.7}$$

The integral of the absolute value of $\boldsymbol{\varepsilon}(\tau)$ can be computed for each segment to determine how much the solution has deviated from the actual dynamics in that segment:

$$\boldsymbol{\eta}_k = \int_{t_k}^{t_{k+1}} |\boldsymbol{\varepsilon}(\tau)| d\tau. \tag{4.5.8}$$

4.6 Test Cases

In order to demonstrate the freedom that optimization of contacts presents, multiple tasks were chosen: gait, safe falling and getting back up. These tasks were chosen as they reflect balance control related tasks that are still an active area of research.

All scenarios use the same dynamic model from Section 4.4. The number of knot points, objective function weights and boundaries were varied. The resulting solutions were analyzed by their objective value, constraint violation and error functions (4.5.7 and 4.5.8). A brief overview of the used setting can be found in Table 4.2.

The solutions were generated on a laptop with an AMD Ryzen 5 PRO 4650U processor, running at 2.7 GHz.

4.6.1 Gait

The first test case was generating optimal gait trajectories. The objective function weights — w_u and w_λ from Eqn. 4.3.1a — were both set to 1. Through experimentation a time span of 2 seconds and walking distance of 2 meters was chosen as it guaranteed 2 or more steps. After that, the number of knot points was increased from 4 to 60 in increments of 2. These results were analyzed to adapt the algorithm such that it more reliably produced a humanoid gait trajectory.

This was achieved by setting w_u to 10 and running the optimization twice: First, with 4 knot points for humanoid gait and low computational cost. This trajectory was then used as the initial guess for a second optimization with 49 knot points. This number of knot points was chosen as it keeps the 4 initial knot points in place, but should result in acceptable constraint violations and more closely resemble the real dynamics.

Table 4.2: Overview of (some of) the settings used for different tasks. Where N indicates the number of knot points, w_u the weight of the joint torques squared cost and w_λ the weight of the GRF squared.

Task	N	**Time (s)**	w_u	w_λ	**Initialization**
Gait	49[1]	2	10	1	linear interpolation
Perturbation	60	2	1	10	initial state
Falling	60	2	1	10	physics first[2]
Recovery	60	3	10	1	linear interpolation

[1] Refining the result of an optimization with 4 knot points.
[2] Refining the result of a simulation without control or objective function.

4.6.2 Recovery from Perturbations

An additional, unforeseen, test case was optimal recovery from the initial perturbations meant to provoke falling behaviour. The only difference with falling — which will be discussed next — is that the initial guess is to keep the initial position.

4.6.3 Safe Falling

The second test case was generating optimal safe falling trajectories. In the unfortunate event a patient with a robotic exoskeleton falls, it would be nice to limit the severity of the pilots injury. This can be done by minimizing the impact forces, especially on the head and hip.

An initial state of standing straight and some initial (rotational) velocity on the CoM was chosen to provoke falling behaviour. A time span of 2 seconds and objective function weights w_u and w_λ of 1 and 10 respectively were chosen. Specific weights for the head and hip could be added, but weren't necessary to generate safe falling trajectories.

As the algorithm would rather cheat than fall over, it was decided to run the algorithm without control or objective function first. This result was then used to optimize the objective function with control.

There were no additional constraints for the final state, and the initial guess was to keep the initial state.

4.6.4 Getting Up

The final test case was generating optimal trajectories to get up after having been severely disturbed. As the model does not have arms and has limited joint mobility, it would be impossible to get up from lying down. Therefore an initial state with one knee on the ground was chosen. The desired state was to return to the standing position used for the safe falling optimization.

Chapter 5

Results

This chapter will present the results of generating optimal trajectories for gait (5.1), recovery from perturbations (5.2), safe falling (5.3) and getting up (5.4).

It is important to state that the time span and desired state had to be tweaked to allow for acceptable convergence and seemingly realistic results. As this is difficult to convey through tables, graphs or images, it does not have a prominent place in this chapter. Nonetheless, it is a notable result that must be mentioned.

All results are from the direct collocation, not regular simulation. This means that there is a discrepancy between the displayed and real behaviour of the system.

Each of the images displaying snapshots of the orientation to give an idea of the motion use eight frames of the generated trajectory, that are equally spaced in time. The initial position of the CoM and floor are indicated with the vertical (x=0) and horizontal (y=0) black lines.

5.1 Gait

Experimental results showed that a goal of 2 meters in 2 seconds should be feasible and result in 2 or more steps. With these parameters the number of knot points was increased, the results of which can be seen in Figure 5.1a (and more extensively in Table E.1).

There is a visible positive trend for the solver time when the number of knot points increases. The collocation error has a steep drop for the first 30 knot points, but remains around the same values for higher numbers of knot points.

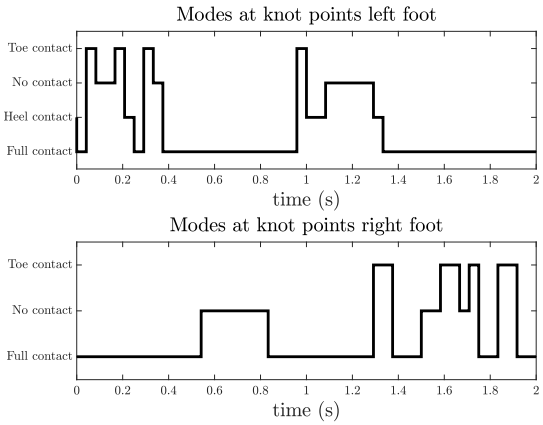
The final result of generating a gait trajectory with 4 knot points and using the result as the initial guess for a second optimization with 49 knot points is shown in Figure 5.2. The generated trajectories and results for the trajectory with 4 knot points can be found in Appendix E.1.

Figure 5.2a shows that two steps were taken to get from its initial position to the desired position. The feet seem to slip, but in the animation it can be seen that it is merely repositioning its feet.

Figure 5.1b shows how it switches between contact modes. It is possible to distinguish two steps for the right foot and some slight repositioning of both feet. It is notable that the mode sequence of the trajectory does not seem to mimic the mode sequence of human gait with a heel strike and toe push-off.

Figure 5.2b shows the mean deviation from the actual dynamics. Because the mean of different units — m s^{-2} and rad s^{-2} — was taken, it is not specific on the error. The lack of units is irrelevant here, as errors of this magnitude will be impossible to track. To further

(a) Performance values for gait generation with varying number of knot points. Performance is expressed in solver time (black) and the maximum absolute collocation error (red).



(b) The mode sequence plot of both feet for the gait optimal trajectory that is displayed in Figure 5.2a.

illustrate this point, the largest error in a single segment was 3.7 rad s^{-1} on the left knee around 0.85 seconds. The optimization of this trajectory took 38 minutes.

5.2 Recovery from Perturbation

Results of resisting a perturbation can be seen in Figure 5.3. The perturbation was modelled with an initial velocity of -1 m s^{-1} in the x-direction and 1 rad s^{-1} on the CoM. This optimization used 60 knot points. The generated position, velocity and control trajectories can be seen in Appendix E.2.

Figure 5.3a shows how a single step backwards was taken to recover from the applier perturbation. The mode sequence (Figure 5.3b) illustrates how the solver is free to vary contact modes. There is some contact between the toe and the ground while moving the leg backwards, and some full contact before it actually comes to a standstill.

Optimizing this trajectory took 14 minutes and the largest error in a single segment was 1.7 rad s^{-1} on the right knee around 0.56 seconds. An error of this order would cause tracking errors.

5.3 Falling

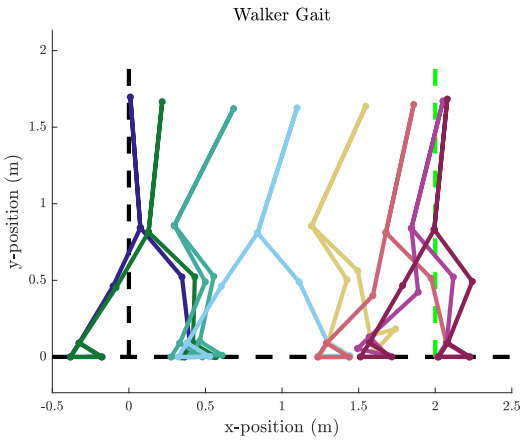
Results of optimal falling trajectories can be seen in Figure 5.4 and Figure 5.5. Falling was initiated with an initial velocity of -1 m s^{-1} in the x-direction and 2 rad s^{-1} on the CoM. This optimization used 60 knot points. The generated position, velocity and control trajectories can be seen in Appendix E.3

Figure 5.4a shows how it tries to recover with a few steps backwards, but falls over in the end. Note how it moves its upper body forward and bends its knees while falling. This protects its head and decreases the final impact velocity on the hip.

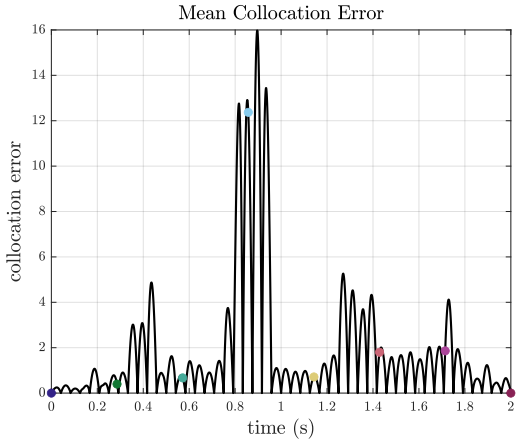
The modes in Figure 5.4b demonstrate how the solver is able to choose whichever contacts it deems fit. Knee and hip contact was omitted from the modes plot for clarity.

The ground reaction forces on the knee and hip are displayed in Figure 5.5. With peak forces of 2607 and 992 N on the hips and knees respectively.

The optimization of this trajectory took 83 minutes and the largest error in a single segment was 2.5 rad s^{-1} in the acceleration of the left ankle in the first segment. An error of this order

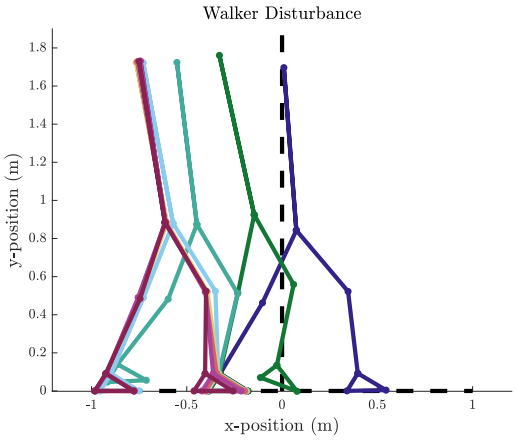


(a) Eight frames of the generated gait trajectory. The initial position of the CoM is indicated with the vertical black line (x=0) and the goal with the vertical green line (x=2). The floor is indicated with the horizontal line (y=0).

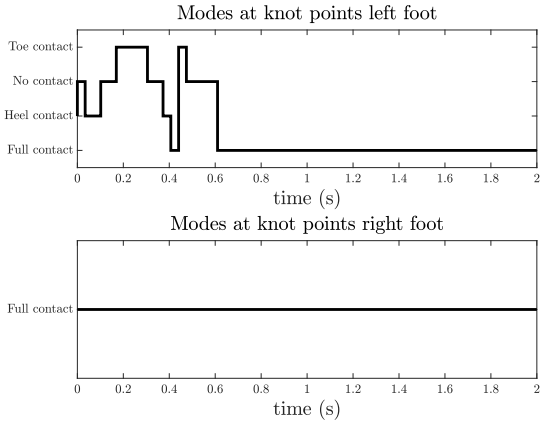


(b) The mean collocation error of the generated gait trajectory. The mean of variables with different units (meters and radians) was taken, therefore the mean collocation error has no meaningful unit. The coloured dots correspond with the frames of Figure 5.2a.

Figure 5.2: Refined gait with 49 knot points and 4 knot points as initialization. Time goes from **cobalt blue** to **raspberry red**.

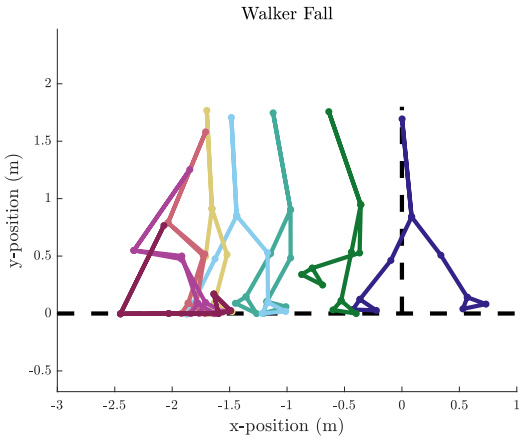


(a) Eight frames of the generated trajectory to resist a perturbation.

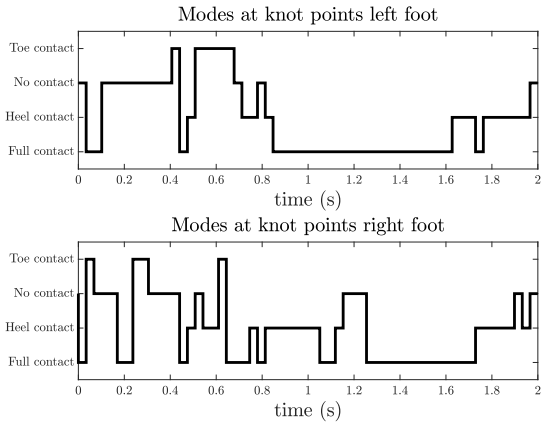


(b) The mode sequences of both feet while trying to resist a perturbation.

Figure 5.3: Resisting a perturbation with 60 knot points. Time goes from **cobalt blue** to **raspberry red**.



(a) Eight frames of the generated optimal falling trajectory.



(b) The mode sequences of both feet while trying to optimize falling.

Figure 5.4: Optimal falling with 60 knot points. Time goes from **cobalt blue** to **raspberry red**.

would cause tracking errors.

5.4 Recovery

Results of recovering from a kneeling to neutral stance can be seen in Figure 5.6 and Figure 5.7. This optimization used 60 knot points. The generated position, velocity and control trajectories can be seen in Appendix E.4.

Figure 5.6a shows how it slowly returns to a neutral stance using its right knee and toe, while sliding its left foot over the floor. The modes in Figure 5.6b show how initially only the right toe is used, as the heel cannot make contact yet due to kinematic constraints. Knee contact was omitted from the modes plot for clarity.

From Figure 5.7 it can be seen that the GRF generated on the left foot do not oppose the direction of slip. Which is notable because the contact constraints were supposed to disallow this exact behaviour. Something else that is notable about the GRF are the oscillations.

The optimization of this trajectory took 37 minutes and the largest error in a single segment was 1.2 rad s^{-1} on the right ankle around 0.89 seconds. An error of this order would cause tracking errors.

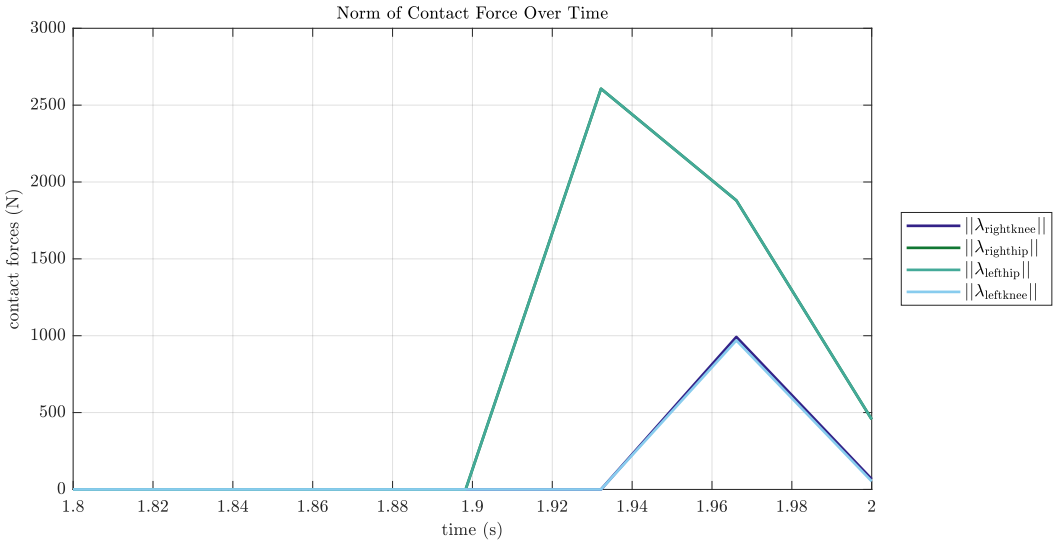
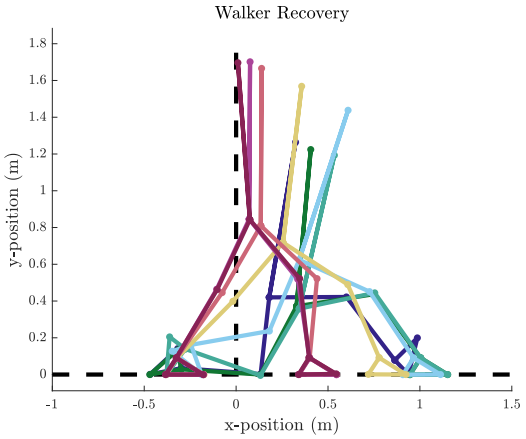
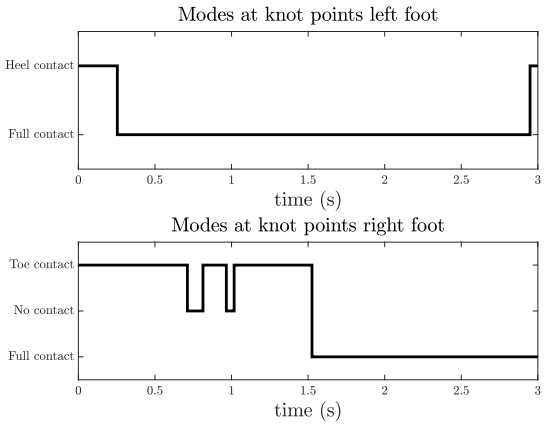


Figure 5.5: Norm of the contact forces on the knees and hips over time of the generated falling trajectory.



(a) Eight frames of the generated trajectory to recover to a neutral stance.



(b) The mode sequences of both feet while recovering to a neutral stance.

Figure 5.6: Recovering from kneeling to neutral stance with 60 knot points. Time goes from **cobalt blue** to **raspberry red**.

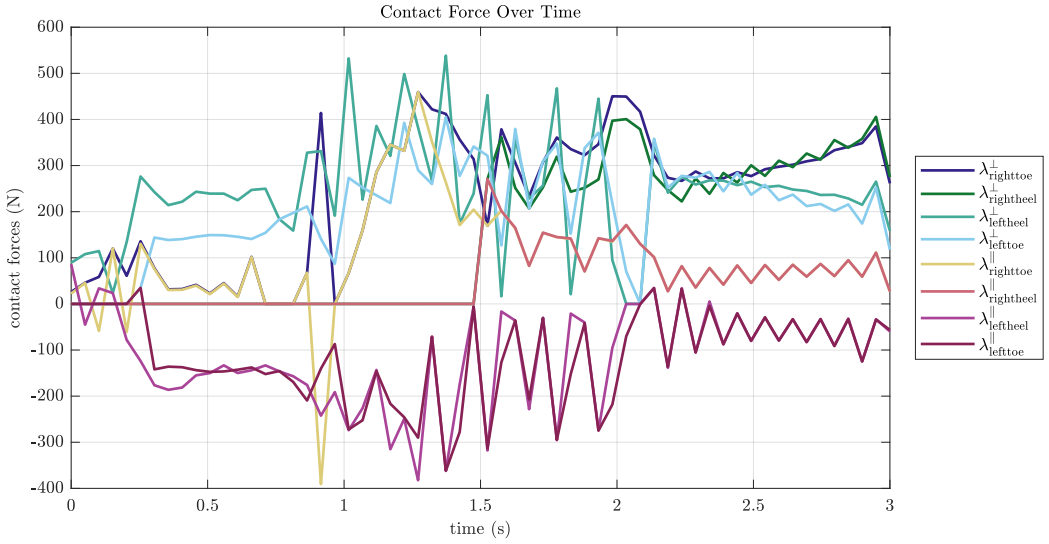


Figure 5.7: Contact forces over time of the generated recovery trajectory.

Chapter 6

Discussion

This thesis set out to include the contact phases into the optimization of trajectories for robotics, and analyze whether this method is able to generate motion plans for several balance control related tasks. The final goal was achieved for a model that represents a person in a lower limb exoskeleton in the sagittal plane.

The discrete behaviour of contact dynamics poses a fundamentally difficult problem for gradient based optimization. The problem was approached by researching different direct implicit contact methods. The method pioneered by Posa et al. [10, 11] was deemed to be the most suitable approach. This method essentially converts the problematic discrete behaviour of contact dynamics into a search problem. Contact forces are found that satisfy complementarity constraints, which approximate the dynamics of contacts.

A framework — heavily inspired by the work of Kelly [18] — was developed to perform direct trapezoidal collocation using the chosen implicit contact method. This was achieved by using IPOPT [14, 20] in MATLAB [19].

The results substantiate the claim that this implicit contact method is indeed able to generate motion plans for tasks such as gait, recovery from perturbations, safe falling and getting up. Although not effortless, a single framework is capable of performing vastly different tasks with relatively little input effort for the user. These results can serve as a foundation for the development of more comprehensive control techniques of lower limb exoskeletons. This could be achieved by using the results of implicit methods as training data, or studying the behaviour to create more task specific methods.

The direct implicit method can still be improved, especially the physical feasibility and region of convergence require additional attention.

This thesis contributes by demonstrating that this direct implicit optimization method can be used to generate optimal — human-like — behaviour of different balance control related tasks such as: gait, recovery from perturbations, safe falling and getting up. This makes it interesting for the application in lower limb exoskeletons and walking robots.

Another contribution is the framework to perform trajectory optimization with implicit contacts.

6.1 Comparison to Similar Method by Posa et al.

The contact phases were included in the optimization as described by Posa et al. [10, 11]. The method caused unrealistic GRF in the final segment. An additional constraint (Eqn. 4.3.1h) was required to prevent this. Posa et al. also presented a novel formulation for the complementarity constraints, which uses a slack variable (Eqn. 2.5.4). It was claimed that this improved

convergence and computational efficiency. However, when implemented in the framework it only caused convergence to local minima, which were often infeasible.
Posa et al. stated that their method was able to discover optimal, human-like, mode sequences while generating gait trajectories. These had a cyclic, swing → heel strike → full contact → toe push-off → swing, pattern for each foot. The same cannot be said for the gait generated by the framework developed for this thesis. This may be, from most to least likely, caused by differences in the problem formulation:

Posa et al. give no details on the used dynamic model and initialization other than how it looks and the initial mode sequence.

The objective function used by Posa et al. also included the task (called cost of transport), but other literature [17, 18] seems to contradict this approach as it would cause a significantly less tractable NLP.

The solver used by Posa et al. is some of the shelf SQP solver — probably SNOPT [26] — unfortunately SNOPT requires a license. As the development was done with `fmincon` [19], tests were also performed with its SQP algorithm. SQP converged faster on simple problems than its interior-point equivalent, but struggled with larger problems such as the biped.

Because IPOPT [14] is better suited for large problems than `fmincon` and does not require a license, it was used for the biped. IPOPT comes packaged with the linear solver called MUMPS [27, 28], but sources [20] suggest that other linear solvers, such as those from HSL [29], might yield better results.

Posa et al. demonstrated gait generation and some form of object manipulation. This thesis expands on the method by demonstrating that it is also capable of resisting perturbations, fall safely and get up. Furthermore, the generated trajectories have more ground clearance during gait than the results of Posa et al., which seems more human-like.

6.2 Versatile Framework with Versatile Performance

In general, the framework is able to generate efficient trajectories for a variety of tasks. However, there are some recurring shortcomings of the framework that will be discussed here.

6.2.1 Large Deviations from Actual Dynamics

A notable shortcoming about direct implicit method — which is rarely discussed in literature — are the large deviations from the actual dynamics. If the generated positions, velocities and controls were to be executed, the result would be nowhere near the results that are shown. As a consequence, it is not guaranteed to be physically feasible. If the sequence of states can be physically executed, it would require a very high feedback control effort — which is undesirable. This means that as of now, increasing the accuracy of the used collocation methods should be a first priority.

The error in the dynamics is relatively high, and for this application it is unknown what an acceptable error is. Roos [5] stated that an infeasibility on the order of magnitude of -5 was required, but that would not produce any results with this method. Additionally, this only decreases the error at the collocation points, but does not guarantee anything between the collocation points.

6.2.1.1 Improving Accuracy

There are multiple causes for this high error, one is that the problem is difficult and that the constraints are highly nonlinear. Another cause is that the applied constraints fail to accurately capture the dynamics.

The accuracy can be increased in multiple ways: by increasing the number of segments, collocation points, or the order of the method. Each of these has its own trade-offs.

Increasing the number of segments is a method that increases both the accuracy and the computational cost, which is well documented [12, 18, 22]. However, this only works up to a certain point.

A more sophisticated method is increasing the number of collocation points by employing higher order collocation methods. A good example has been described by Patel et al. [12], where third and fifth order orthogonal (Radau) collocation methods were used to improve the accuracy with almost an order of magnitude with identical computational cost. Patel et al. [12] also improved the realism of the contact dynamics by enforcing the complementarity constraints at the collocation points instead of knot points.

Even better is keeping the order of the dynamics into account when constructing the collocation constraints. In most collocation applications and literature, the dynamics are rewritten as nonlinear first order state equations. The collocation constraints are then applied to $\boldsymbol{x}$. But usually $\boldsymbol{x}$ includes two different orders: $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. Essentially a second order system gets treated as a first order system. This results in that the velocities end up getting constrained at the collocation points, but not the accelerations. In other words, it causes the dynamics $\ddot{\boldsymbol{q}}_k = \boldsymbol{f}_k$ to be violated at the collocation points, resulting in larger errors between the collocation points. Martín et al. [22] applied appropriate collocation constraints to $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$, and while doing so demonstrated an error reduction of more than one order of magnitude with identical computational time. Therefore, it should become the norm to formulate the collocation constraints with the same order as the dynamics. A more thorough derivation is shown in Appendix B.4.

6.2.2 The Road to Reliability

The reliability of producing acceptable results is not great, the framework has to be dialed in specifically, which is more of an art than a science. This is a complex problem to solve, as it is difficult to predict what the effects of certain changes will be. But there are plenty of options to consider.

6.2.2.1 The Role of Initial Guesses

The initial guess is known to have a substantial impact on the results of trajectory optimization [18]. The framework developed for this thesis linearly interpolates between the initial and desired positions, *i.e.* slides across the floor from its initial to its desired position. This was a conscious decision, as no additional user input is required. However, it could be a big reason why the developed framework shows a small region of convergence. At the cost of generality and simplified usability, more task specific guesses could be made that actuate limbs and include control and GRF.

A small experiment was performed where the initial guess was the output of a phase functioned neural network [30], which was trained on motion capture data. This seemed to perform more reliably for the generation of gait, but no thorough analysis was performed.

6.2.2.2 Improving Convergence by Adjusting the Time Span

The time span over which will be optimized has to be chosen by the user. However, it can be difficult for the user to determine a good estimate, which can result in poor convergence.

Winkler et al. [6, 7] introduced decision variables to individually estimate the optimal duration of each segment. This allows the framework to adjust the user's estimate, which may improve convergence. Additionally, it might result in more accurate solutions. However, this will result in a less sparse Jacobian and therefore a less tractable NLP.

In case this trade-off is not worth it, an alternative method is to make the total time span a decision variable. Each segment is then scaled the same amount. This reduces the number of new decision variables, but still allows the solver to correct the users estimate. This method has been demonstrated by Kelly in the demos of the OptimTraj [18] library for MATLAB.

6.2.2.3 Mesh Refinement to Improve Accuracy and Convergence

For the generation of gait, using the result of an optimization with 4 knot points as an initial guess for a second optimization with 49 knot points resulted in better results — with similar computational cost — than just using 49 knot points. This is a crude implementation of mesh refinement, an iterative strategy where the resolution of knot and/or collocation points is increased on segments of previous solutions. Usually, mesh refinement is employed to efficiently achieve a certain accuracy. But, based on the experimental gait results, it might also improve the convergence properties.

A more sophisticated implementation of mesh refinement would use the error functions to decide whether a segment gets subdivided or a higher order quadrature is used to approximate the integral of the dynamics. The order gets increased if the error is near the threshold, additional knot points are added if the error is much larger than the threshold [18].

6.2.3 Abnormal Modes

Although the framework was able to "reason" about contacts, as discussed in Section 6.1, the mode sequences that were discovered usually do not seem to make a lot of sense.

Related to these abnormal modes, is the oscillating behaviour which can be seen in virtually all GRF, control and velocity graphs. It seems to be particularly noticeable in scenarios where it tries to remain still. It seems to struggle to find a good balance, with forces evenly distributed over the contact points. Although it is unsure whether this is caused by the controls, GRF or something else, it seems likely that this contributes to worse tractability.

While keeping in mind that the goal is to — in some way — implement this to control robotics in reality, the oscillating control is especially worrisome. It could cause unwanted vibrations and instability. Therefore, it could be useful to regularize the control input during optimization, such that it does not fluctuate as much.

Alternatively, it would be interesting to thoroughly analyze the differences of generated trajectories with pin feet. This might give an indication of what causes the oscillations.

In the work of Roos [5] it can be seen that higher numbers of knot points result in a more noisy looking control profile, therefore this is also an important variable to keep in mind.

6.2.4 Costly Computation

The computational cost of the developed framework is unpractical. MPC use cases generally update the reference trajectory at 10 Hz, most of the optimizations with the developed framework took thousands of seconds. This is not surprising as explicit whole-body methods already were too slow [5] for MPC use cases.

The best conceivable use case would be generating perfectly labeled training data for a neural network. This trained network might then be used in practice. But, the computational efficiency of the developed framework can still be improved.

6.2.4.1 Improving the Computational Efficiency

There are two concrete examples where there is room for improvement: the hessian and the contact model.

The Hessian serves a similar purpose as the Jacobian. Currently the Hessian is estimated by a limited-memory quasi-Newton method (L-BFGS), which was included with IPOPT [14].

As was described in Subsection 3.2.1, if derivatives can be computed with a reasonable computational effort, it usually results in faster and more robust convergence. Therefore, it would be worthwhile to include a user-specified analytic Hessian, in addition to the Jacobian.

The computational cost increases with the number of dimensions and constraints of the problem, and the number of dimensions is equal to the number of decision variables times the number of knot points. So it would make sense to avoid unnecessary decision variables and constraints.

It is possible to enforce friction cones with less decision variables and constraints by using a linear combination to compute the GRF. This is discussed in more detail in Appendix F.

As being able to have slipping contacts is not essential for all contact points, they could be removed where they are not deemed to be necessary. This would also reduce both the number of constraints and decision variables.

6.2.5 Task Specific Discussion

The previous subsections discussed shortcomings and possible improvements of the developed framework. This section will not repeat those same points but solely focus on the task specific results.

6.2.5.1 Rediscovering Gait

The framework is able to discover gait trajectories for a humanoid biped that resemble the gait of a human. To do this the user only has to specify the time span and distance that has to be traversed. The contact phases, states and controls are handled automatically.

It was notable that the accuracy of the solution did not keep increasing with more collocation points. This was likely due to the solver optimizing an artifact of the model formulation, which resulted in a lot of short contact phases. Changing the objective function to penalize the change in ground reaction forces did not seem to improve this.

6.2.5.2 Perturbations: Refusing to Fall

With little to no alteration except removing a desired end position and adding initial velocities, the framework was also able to correctly restore balance from perturbations.

These perturbations were initially intended to provoke falling behaviour. However, if the initial conditions were not difficult enough, it would result in proper recovery. But, if the initial conditions were too difficult, it defied the prescribed physics in favour of adhering to the initial guess.

Restoring the balance from the perturbations was done in a way that resembles the ankle, hip and step strategy — balance strategies also employed by humans.

It would be interesting to extend the framework in such a way that forces may be applied, that are not considered contact forces. This would make it easier and more realistic to test perturbations and falling.

6.2.5.3 Safe Falling

As discussed, it was difficult to get the framework to optimize falling. This was solved by using an initial guess, more similar to a falling motion.

The trajectories that were generated seemed promising, but optimal falling would benefit from further analysis.

There was also an impact on the knee, which is not entirely realistic. This happened because the walker is allowed to completely stretch its knee, and its limbs are one dimensional.

It would also be interesting to add arms to the model, as they are often used to break the fall.

6.2.5.4 Getting Up

The framework was also capable of generating trajectories to restore to a neutral posture from difficult positions. It is notable that the framework does not lift its left foot, but decides to slide it instead. From the contact force plot it can be seen that the ground reaction forces to not oppose the direction of slip, which indicates that the solver is cheating the model.

It is not clear why this happened, but likely due to time steps that are too large. Slipping behaviour is not always desirable, it would have been interesting to see results with the option of slipping disabled.

6.3 Human-Like Movement

This thesis tends to judge the results on whether they seem human-like. The best argument to do so is that through some millions of years of evolution, humans (and other animals) have become very good at performing tasks "optimally". This optimality could be defined as minimizing pain and energy expenditure while performing a task. Another argument is that the adoption of robotics in every day use might be more accepted if they perform like a human would. So if seeming human-like is a good thing, how do you objectively evaluate what is human-like and what is not? The objective function should do the job, except that was not always the case. The value of the used objective function did not significantly get lower if solutions were more human-like than others for the same task. An interesting question that remains is: what objective function captures human-like movement for a multitude of tasks, while remaining tractable?

6.4 Future Work

Future work should first and foremost emphasize physical feasibility. To achieve this, the collocation method should be changed to a second order method as described by Martín et al. [22] and the complexity should be decreased.

To improve reliability and computational cost of the framework, it could help to: remove the ability to have slipping contacts, use a model with pin feet and include the time span as a decision variable.

By not allowing slipping contacts in the optimization, 3 out of 4 complementarity constraints, and 1 out of 3 decision variables can be removed. This would make the NLP less complicated,

while still being able to perform basic balance control tasks. Therefore, it would be interesting to study the performance difference when the slipping contacts have been removed.

The framework has a hard time with consistently discovering logical mode sequences on a small scale, *i.e.* heel-toe. But, the big picture — when which limb is in contact — is the most interesting aspect of implicit contacts. Therefore, the biped model with pin feet could be used to plan a trajectory to get the big picture. With the direction of movement known, the implicit result could be converted to an explicit input with small scale modes scheduled in a sensible way. This could result in more physical feasibility, efficient trajectories and computational efficiency.

Currently the user estimate of the time span is crucial to generate efficient gait trajectories. By including it as a decision variable, the model might become more robust to poor user input.

The developed framework has demonstrated that it is capable of a variety of balance control related tasks on a flat floor, but the real world is not flat. Therefore, it would be interesting to study the performance on terrain with varying friction and slope.

An important next step is to apply the results from these implicit methods in reality. This could give a better estimate of what the required physical feasibility is.

References

[1] Emma M. Smith, Brodie M. Sakakibara, and William C. Miller. A review of factors influencing participation in social and community activities for wheelchair users activities for wheelchair users. *Disability and Rehabilitation: Assistive Technology*, 11(5):361–374, 2016. doi: 10.3109/17483107.2014.989420.

[2] Claire L. Flemmer and Rory C. Flemmer. A review of manual wheelchairs. *Disability and Rehabilitation : Assistive Technology*, 11(3), 2016. doi: 10.3109/17483107.2015.1099747.

[3] Glen W. White, Mark R. Mathews, and Stephen B. Fawcett. Reducing Risk of Pressure Sores: Effects of Watch Prompts and Alarm Avoidance on Wheelchair Push-ups. *Journal of Applied Behavior Analysis*, 22(3):287–295, 1989. doi: 10.1901/jaba.1989.22-287.

[4] Ghan-shyam Lohiya, Lilia Tan-figueroa, Steve Silverman, and Hung Van Le. The Wheelchair Thrombosis Syndrome. *Journal of the National Medical Association*, 98(7):1188–1192, 2006. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2569471/`. PMID: 16895294; PMCID: PMC2569471. Last accessed on 12/12/2022.

[5] Robert A. Roos. Master of Science Thesis: Design of a Direct Collocation Model Predictive Control Framework for Legged Locomotion Based on Whole-Body Dynamics and Explicit Contact Phases, 2020. URL `https://essay.utwente.nl/85467/1/Roos_MA_ET.pdf`. Last accessed on 12/12/2022.

[6] Alexander W. Winkler, C. Dario Bellicoso, Marco Hutter, and Jonas Buchli. Gait and Trajectory Optimization for Legged Systems through Phase-based End-Effector Parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567, 2018. doi: 10.1109/LRA.2018.2798285.

[7] Alexander W. Winkler. Doctoral Thesis: Optimization-based motion planning for legged robots. *ETH zürich Library*, 2018. doi: 10.3929/ethz-b-000272432.

[8] Frans A. Koolen. Doctoral Thesis: Balance control and locomotion planning for humanoid robots using nonlinear centroidal models by, 2020. URL `https://hdl.handle.net/1721.1/128291`. Last accessed on 12/12/2022.

[9] Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics*, 31(4):1–8, 2012. ISSN 07300301. doi: 10.1145/2185520.2185539.

[10] Michael Posa and Russ Tedrake. Direct Trajectory Optimization of Rigid Body Dynamical Systems through Contact. *Algorithmic Foundations of Robotics X*, 86:527–542, 2013. doi: 10.1007/978-3-642-36279-8_38.

[11] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *International Journal of Robotics Research*, 33(1):69–81, 2014. ISSN 17413176. doi: 10.1177/0278364913506757.

[12] Amir Patel, Stacey Leigh Shield, Saif Kazi, Aaron M. Johnson, and Lorenz T. Biegler. Contact-implicit trajectory optimization using orthogonal collocation. *IEEE Robotics and Automation Letters*, 4(2):2242–2249, 2019. ISSN 23773766. doi: 10.1109/LRA.2019.2900840.

[13] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

[14] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. ISSN 0025-5610, 1436-4646. doi: 10.1007/s10107-004-0559-y.

[15] Mahsa Khalili, Jaimie F. Borisoff, and H.F. Machiel van der Loos. Developing safe fall strategies for lower limb exoskeletons. *2017 International Conference on Rehabilitation Robotics (ICORR)*, pages 314–319, 2017. doi: 10.1109/ICORR.2017.8009266.

[16] John T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Society for Industrial and Applied Mathematics, second edition, 2010. ISBN 978-0-89871-688-7 978-0-89871-857-7. doi: 10.1137/1.9780898718577.

[17] Matthew P. Kelly. Transcription methods for trajectory optimization: a beginners tutorial. *arXiv math*, 2017. doi: 10.48550/arXiv.1707.00284.

[18] Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *Society for Industrial and Applied Mathematics*, 59(4):849–904, 2017. ISSN 0036-1445, 1095-7200. doi: 10.1137/16M1062569.

[19] MATLAB. *version 9.13 (R2022b)*. The MathWorks Inc., Natick, Massachusetts, 2022. URL `https://nl.mathworks.com/products/matlab.html`.

[20] Enrico Bertolazzi. ebertolazzi/mexIPOPT, 2022. URL `https://github.com/ebertolazzi/mexIPOPT/releases/tag/1.1.4`. Last accessed on 12/12/2022.

[21] Russ Tedrake. *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. (Course Notes for MIT 6.832), 2022. URL `https://underactuated.csail.mit.edu`. Last accessed on 12/12/2022.

[22] Siro Moreno-Martın, Llus Ros, and Enric Celaya. Collocation methods for second order systems. *Robotics: Science and Systems Conference (RSS)*, pages 1–11, 2022. doi: 10.15607/RSS.2022.XVIII.038.

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

[24] Nikolaus Hansen. The CMA Evolution Strategy: A Tutorial. *ArXiv e-prints*, 2016. doi: 10.48550/arXiv.1604.00772.

[25] Joep T.J. van de Rijt. Master Thesis: Design of a Momentum Based Optimal Controller for a Lower Limb Humanoid, 2022. URL `https://essay.utwente.nl/92234/`. Last accessed on 12/12/2022.

[26] Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization. *Society for Industrial and Applied Mathematics Review*, 47(1):99–131, 2005. doi: 10.1137/S0036144504446096.

[27] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001. doi: 10.1137/S0895479899358194.

[28] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006. doi: 10.1016/j.parco.2005.07.004.

[29] HSL. A collection of Fortran codes for large scale scientific computation., 2019. URL `http://www.hsl.rl.ac.uk/`. Last accessed on 12/12/2022.

[30] Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned Neural Networks for Character Control. *ACM Transaction on Graphics*, 36(4):1–13, 2017. doi: 10.1145/3072959.3073663.

[31] R. J. Barnes. Matrix Differentiation, 2006. URL `https://atmos.washington.edu/~dennis/MatrixCalculus.pdf`. Last accessed on 12/12/2022.

Appendix A

Verification of Planar Dynamics

After deriving dynamics, it is important to verify whether everything is correct. This appendix will detail five requirements planar dynamics should satisfy.

Conventions:
Generalized coordinates $\boldsymbol{q} \in \mathbb{R}^n$.
Mass matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$.
$\dot{\mathbf{M}} = \frac{d\mathbf{M}}{dt}$.
Fictitious forces matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$.

- $\mathbf{M}$, should be symmetric:

$$\mathbf{M} = \mathbf{M}^\top. \tag{A.0.1}$$

- $\mathbf{M}$ should be positive definite:

$$\mathbf{M} \geq \mathbf{0}. \tag{A.0.2}$$

- $\mathbf{C}$, should be asymmetric:

$$\mathbf{C} \neq \mathbf{C}^\top. \tag{A.0.3}$$

- $\left[\dot{\mathbf{M}} - 2\mathbf{C}\right]$ should be a skew-symmetric matrix:

$$-\left[\dot{\mathbf{M}} - 2\mathbf{C}\right] = \left[\dot{\mathbf{M}} - 2\mathbf{C}\right]^\top. \tag{A.0.4}$$

- $\left[\dot{\mathbf{M}} - 2\mathbf{C}\right]$ should be positive definite:

$$\left[\dot{\mathbf{M}} - 2\mathbf{C}\right] \geq \mathbf{0}. \tag{A.0.5}$$

Appendix B

Deriving Collocation Methods and Corresponding Interpolant

This appendix will show the derivation for collocation methods and how to correctly interpolate the resulting solutions.

It is not recommended to implement any of the 1st order approximations, unless the system dynamics are 1st order as well, of course.

Conventions:

$$\tau = t - t_k, \tag{B.0.1}$$
$$h_k = t_{k+1} - t_k, \tag{B.0.2}$$
$$\boldsymbol{q}\left(t_k\right) = \boldsymbol{p}_k, \tag{B.0.3}$$
$$\dot{\boldsymbol{q}}\left(t_k\right) = \boldsymbol{v}_k, \tag{B.0.4}$$
$$\ddot{\boldsymbol{q}}\left(t_k\right) = \boldsymbol{g}_k, \tag{B.0.5}$$
$$\boldsymbol{x}_k = \left[\boldsymbol{p}_k, \boldsymbol{v}_k\right]^\top, \tag{B.0.6}$$
$$\dot{\boldsymbol{x}}_k = \left[\boldsymbol{v}_k, \boldsymbol{g}_k\right]^\top, \tag{B.0.7}$$
$$\dot{\boldsymbol{x}}_k = \boldsymbol{f}_k. \tag{B.0.8}$$

Where t is continuous time, t_k the time at a knot point that separates continuous time in segments, τ is the time in a segment, h_k the segment length, $\boldsymbol{q}$ are the generalized coordinates, $\boldsymbol{p}_k$, $\boldsymbol{v}_k$ are the positions and velocities at knot point k, $\boldsymbol{g}_k$ are the EoM at knot point k and x_k the state vector at knot point k.

B.1 Euler Collocation (1st Order)

The Euler method approximates integration with a first order polynomial:

$$\boldsymbol{x}(\tau) = a + b\tau, \tag{B.1.1}$$
$$\dot{\boldsymbol{x}}(\tau) = b. \tag{B.1.2}$$

There are two unknowns (a and b), so find two knowns:

$$\boldsymbol{x}_k = \boldsymbol{x}(0), \tag{B.1.3}$$
$$\dot{\boldsymbol{x}}_k = \dot{\boldsymbol{x}}(0), \tag{B.1.4}$$
$$\dot{\boldsymbol{x}}_k = \dot{\boldsymbol{x}}(h_k). \tag{B.1.5}$$

Note that there is a choice for $\dot{\boldsymbol{x}}_k$ between Euler-forward and Euler-backward integration,B.1.4 and B.1.5 respectively. As discussed in Section 3.4, Euler-backward is numerically more stable and in direct collocation it does not require any additional computation. So continue with that, solving for the unknowns:

$$a = \boldsymbol{x}_k, \tag{B.1.6}$$
$$b = \dot{\boldsymbol{x}}_{k+1}. \tag{B.1.7}$$

Substituting these in B.1.1 and B.1.2 results in the equations that may be used to interpolate:

$$\boldsymbol{x}(\tau) = \boldsymbol{x}_k + \boldsymbol{f}_{k+1}\tau, \tag{B.1.8}$$
$$\dot{\boldsymbol{x}}(\tau) = \boldsymbol{f}_{k+1}. \tag{B.1.9}$$

It is trivial to see that this is an inaccurate approximation of $\boldsymbol{f}$, but it works nonetheless. To find the collocation constraint, evaluate $\boldsymbol{x}_{k+1} = \boldsymbol{x}(h_k)$:

$$\boldsymbol{x}_{k_1} = \boldsymbol{x}_k + h_k \boldsymbol{f}_{k+1}. \tag{B.1.10}$$

B.2 Trapezoidal Collocation (1st Order)

Trapezoidal collocation has proven to be robust [5, 16, 18] and is simple to implement and interpolate. In trapezoidal collocation, integration is approximated with a second order polynomial:

$$\boldsymbol{x}(\tau) = a + b\tau + c\tau^2, \tag{B.2.1}$$
$$\dot{\boldsymbol{x}}(\tau) = b + 2c\tau. \tag{B.2.2}$$

There are three unknowns (a, b and c), so find three knowns:

$$\boldsymbol{x}_k = \boldsymbol{x}(0), \tag{B.2.3}$$
$$\dot{\boldsymbol{x}}_k = \dot{\boldsymbol{x}}(0), \tag{B.2.4}$$
$$\dot{\boldsymbol{x}}_{k+1} = \dot{\boldsymbol{x}}(h_k). \tag{B.2.5}$$

Solve:

$$\boldsymbol{x}_k = a, \tag{B.2.6}$$
$$\dot{\boldsymbol{x}}_k = b, \tag{B.2.7}$$
$$\dot{\boldsymbol{x}}_{k+1} = b + 2h_k c. \tag{B.2.8}$$

Rewriting gives:

$$a = \boldsymbol{x}_k, \tag{B.2.9}$$
$$b = \dot{\boldsymbol{x}}_k, \tag{B.2.10}$$
$$c = \frac{1}{2h_k}\left(\dot{\boldsymbol{x}}_{k+1} - \dot{\boldsymbol{x}}_k\right). \tag{B.2.11}$$

Substituting these unknowns in B.2.1 and B.2.2 results in the equations that may be used to interpolate:

$$\boldsymbol{x}(\tau) = \boldsymbol{x}_k + \boldsymbol{f}_k\tau + \frac{\tau^2}{2h_k}\left(\boldsymbol{f}_{k+1} - \boldsymbol{f}_k\right), \tag{B.2.12}$$
$$\dot{\boldsymbol{x}}(\tau) = \boldsymbol{f}_k + \frac{\tau}{h_k}\left(\boldsymbol{f}_{k+1} - \boldsymbol{f}_k\right). \tag{B.2.13}$$

To find the collocation constraint, evaluate $\boldsymbol{x}_{k+1} = \boldsymbol{x}(h_k)$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x_k} + \boldsymbol{f}_k h_k + \frac{h_k^2}{2h_k}\left(\boldsymbol{f}_{k+1} - \boldsymbol{f}_k\right), \tag{B.2.14}$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{x_k} + \boldsymbol{f}_k h_k + \frac{h_k}{2}\boldsymbol{f}_{k+1} - \frac{h_k}{2}\boldsymbol{f}_k, \tag{B.2.15}$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{x_k} + \frac{h_k}{2}\left(\boldsymbol{f}_k + \boldsymbol{f}_{k+1}\right). \tag{B.2.16}$$

B.3 Hermite-Simpson Collocation (1st Order)

Hermite-Simpson collocation integration is approximated with a third order polynomial, again increasing both the accuracy and computational cost:

$$\boldsymbol{x}(\tau) = a + b\tau + c\tau^2 + d\tau^3, \tag{B.3.1}$$
$$\dot{\boldsymbol{x}}(\tau) = b + 2c\tau + 3d\tau^2. \tag{B.3.2}$$

There are four unknowns (a, b, c and d), so find four knowns:

$$\boldsymbol{x}_k = \boldsymbol{x}(0), \tag{B.3.3}$$
$$\dot{\boldsymbol{x}}_k = \dot{\boldsymbol{x}}(0), \tag{B.3.4}$$
$$\dot{\boldsymbol{x}}_{k+\frac{1}{2}} = \dot{\boldsymbol{x}}(\frac{h_k}{2}), \tag{B.3.5}$$
$$\dot{\boldsymbol{x}}_{k+1} = \dot{\boldsymbol{x}}(h_k). \tag{B.3.6}$$

Here $k + \frac{1}{2}$ is called the midpoint or collocation point, it falls between two meshpoints. Solve:

$$\boldsymbol{x}_k = a, \tag{B.3.7}$$
$$\dot{\boldsymbol{x}}_k = b, \tag{B.3.8}$$
$$\dot{\boldsymbol{x}}_{k+\frac{1}{2}} = b + 2\frac{h_k}{2}c + 3\left(\frac{h_k}{2}\right)^2 d, \tag{B.3.9}$$
$$\dot{\boldsymbol{x}}_{k+1} = b + 2h_k c + 3h_k^2 d. \tag{B.3.10}$$

Rewriting gives:

$$a = \boldsymbol{x}_k, \tag{B.3.11}$$
$$b = \dot{\boldsymbol{x}}_k, \tag{B.3.12}$$
$$c = \frac{1}{2h_k}\left(-3\dot{\boldsymbol{x}}_k + 4\dot{\boldsymbol{x}}_{k+\frac{1}{2}} - \dot{\boldsymbol{x}}_{k+1}\right), \tag{B.3.13}$$
$$d = \frac{1}{3h_k^2}\left(2\dot{\boldsymbol{x}}_k - 4\dot{\boldsymbol{x}}_{k+\frac{1}{2}} + 2\dot{\boldsymbol{x}}_{k+1}\right). \tag{B.3.14}$$

Substituting these unknowns in B.3.1 and B.3.2 results in the equations that may be used to interpolate:

$$\boldsymbol{x}(\tau) = \boldsymbol{x}_k + \boldsymbol{f}_k\tau + \frac{\tau^2}{2h_k}\left(-3\boldsymbol{f}_k + 4\boldsymbol{f}_{k+\frac{1}{2}} - \boldsymbol{f}_{k+1}\right) + \frac{\tau^3}{3h_k^2}\left(2\boldsymbol{f}_k - 4\boldsymbol{f}_{k+\frac{1}{2}} + 2\boldsymbol{f}_{k+1}\right), \tag{B.3.15}$$
$$\dot{\boldsymbol{x}}(\tau) = \boldsymbol{f}_k + \frac{\tau}{h_k}\left(-3\boldsymbol{f}_k + 4\boldsymbol{f}_{k+\frac{1}{2}} - \boldsymbol{f}_{k+1}\right) + \frac{\tau^2}{h_k^2}\left(2\boldsymbol{f}_k - 4\boldsymbol{f}_{k+\frac{1}{2}} + 2\boldsymbol{f}_{k+1}\right). \tag{B.3.16}$$

To find the collocation constraint, evaluate $\boldsymbol{x}_{k+1} = \boldsymbol{x}(h_k)$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{f}_k h_k + \frac{h_k^2}{2h_k}\left(-3\boldsymbol{f}_k + 4\boldsymbol{f}_{k+\frac{1}{2}} - \boldsymbol{f}_{k+1}\right) + \frac{h_k^3}{3h_k^2}\left(2\boldsymbol{f}_k - 4\boldsymbol{f}_{k+\frac{1}{2}} + 2\boldsymbol{f}_{k+1}\right), \tag{B.3.17}$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{f}_k h_k - \frac{3}{2}h_k\boldsymbol{f}_k + 2h_k\boldsymbol{f}_{k+\frac{1}{2}} - \frac{1}{2}h_k\boldsymbol{f}_{k+1} + \frac{2}{3}h_k\boldsymbol{f}_k - \frac{4}{3}h_k\boldsymbol{f}_{k+\frac{1}{2}} + \frac{2}{3}h_k\boldsymbol{f}_{k+1}, \tag{B.3.18}$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{1}{6}h_k\boldsymbol{f}_k + \frac{4}{6}h_k\boldsymbol{f}_{k+\frac{1}{2}} + \frac{1}{6}h_k\boldsymbol{f}_{k+1}, \tag{B.3.19}$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{h_k}{6}\left(\boldsymbol{f}_k + 4\boldsymbol{f}_{k+\frac{1}{2}} + \boldsymbol{f}_{k+1}\right). \tag{B.3.20}$$

There is still one unknown, $\boldsymbol{f}_{k+\frac{1}{2}}$. And as

$$\boldsymbol{f}_{k+\frac{1}{2}} = \boldsymbol{f}(\boldsymbol{x}_{k+\frac{1}{2}}, \boldsymbol{u}_{k+\frac{1}{2}}, \boldsymbol{\lambda}_{k+\frac{1}{2}}), \tag{B.3.21}$$

evaluating $\boldsymbol{x}_{k+\frac{1}{2}} = \boldsymbol{x}(\frac{h_k}{2})$ will make it known:

$$\boldsymbol{x}_{k+\frac{1}{2}} = \boldsymbol{x}_k + \frac{1}{2}h_k\boldsymbol{f}_k + \frac{h_k^2}{4}\frac{1}{2h_k}\left(-3\boldsymbol{f}_k + 4\boldsymbol{f}_{k+\frac{1}{2}} - \boldsymbol{f}_{k+1}\right) + \frac{h_k^3}{8}\frac{1}{3h_k^2}\left(2\boldsymbol{f}_k - 4\boldsymbol{f}_{k+\frac{1}{2}} + 2\boldsymbol{f}_{k+1}\right), \tag{B.3.22}$$
$$\boldsymbol{x}_{k+\frac{1}{2}} = \boldsymbol{x}_k + \frac{1}{2}h_k\boldsymbol{f}_k - \frac{3}{8}h_k\boldsymbol{f}_k + \frac{1}{2}h_k\boldsymbol{f}_{k+\frac{1}{2}} - \frac{1}{8}h_k\boldsymbol{f}_{k+1} + \frac{1}{12}h_k\boldsymbol{f}_k - \frac{1}{6}h_k\boldsymbol{f}_{k+\frac{1}{2}} + \frac{1}{12}h_k\boldsymbol{f}_{k+1}, \tag{B.3.23}$$
$$\boldsymbol{x}_{k+\frac{1}{2}} = \boldsymbol{x}_k + \frac{5}{24}h_k\boldsymbol{f}_k + \frac{8}{24}h_k\boldsymbol{f}_{k+\frac{1}{2}} - \frac{1}{24}h_k\boldsymbol{f}_{k+1}, \tag{B.3.24}$$
$$\boldsymbol{x}_{k+\frac{1}{2}} = \boldsymbol{x}_k + \frac{h_k}{24}\left(5\boldsymbol{f}_k + 8\boldsymbol{f}_{k+\frac{1}{2}} - \boldsymbol{f}_{k+1}\right). \tag{B.3.25}$$

Now rewrite Eqn. B.3.20:

$$\frac{6}{h_k}\left(\boldsymbol{x}_{k+1}-\boldsymbol{x}_k\right)=\boldsymbol{f}_k+4\boldsymbol{f}_{k+\frac{1}{2}}+\boldsymbol{f}_{k+1}, \tag{B.3.26}$$

$$4\boldsymbol{f}_{k+\frac{1}{2}}=\frac{6}{h_k}\left(\boldsymbol{x}_{k+1}-\boldsymbol{x}_k\right)-\boldsymbol{f}_{k+1}-\boldsymbol{f}_k, \tag{B.3.27}$$

$$8\boldsymbol{f}_{k+\frac{1}{2}}=\frac{12}{h_k}\left(\boldsymbol{x}_{k+1}-\boldsymbol{x}_k\right)-2\left(\boldsymbol{f}_{k+1}+\boldsymbol{f}_k\right). \tag{B.3.28}$$

And substitute the result into Eqn. B.3.25:

$$\boldsymbol{x}_{k+\frac{1}{2}}=\boldsymbol{x}_k+\frac{h_k}{24}\left(5\boldsymbol{f}_k+\frac{12}{h_k}\left(\boldsymbol{x}_{k+1}-\boldsymbol{x}_k\right)-2\left(\boldsymbol{f}_{k+1}+\boldsymbol{f}_k\right)-\boldsymbol{f}_{k+1}\right), \tag{B.3.29}$$

$$\boldsymbol{x}_{k+\frac{1}{2}}=\boldsymbol{x}_k+\frac{5}{24}h_k\boldsymbol{f}_k+\frac{1}{2}\boldsymbol{x}_{k+1}-\frac{1}{2}\boldsymbol{x}_k-\frac{1}{12}h_k\boldsymbol{f}_{k+1}-\frac{1}{12}h_k\boldsymbol{f}_k-\frac{1}{24}h_k\boldsymbol{f}_{k+1}, \tag{B.3.30}$$

$$\boldsymbol{x}_{k+\frac{1}{2}}=\frac{1}{2}\boldsymbol{x}_k+\frac{1}{2}\boldsymbol{x}_{k+1}+\frac{1}{8}h_k\boldsymbol{f}_k-\frac{1}{8}h_k\boldsymbol{f}_{k+1}, \tag{B.3.31}$$

$$\boldsymbol{x}_{k+\frac{1}{2}}=\frac{1}{2}\left(\boldsymbol{x}_k+\boldsymbol{x}_{k+1}\right)+\frac{h_k}{8}\left(\boldsymbol{f}_k-\boldsymbol{f}_{k+1}\right). \tag{B.3.32}$$

It is possible to substitute the result into Eqn. B.3.20, which is called the compressed form:

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\frac{h_k}{6}\left(\boldsymbol{f}_k+4\boldsymbol{f}(\frac{1}{2}\left(\boldsymbol{x}_k+\boldsymbol{x}_{k+1}\right)+\frac{h_k}{8}\left(\boldsymbol{f}_k-\boldsymbol{f}_{k+1}\right),\boldsymbol{u}_{k+\frac{1}{2}},\boldsymbol{\lambda}_{k+\frac{1}{2}})+\boldsymbol{f}_{k+1}\right). \tag{B.3.33}$$

It is also possible to create an additional decision variable for the midpoint and use Eqn. B.3.32 as an additional constraint, which is called the separated form. The exact details about the performance differences between compressed and separated form are covered in [16]. According to [18] compressed form performs better when there are a lot of segments, and separated form performs better when there are few segments.

B.4 Trapezoidal Collocation (2nd Order)

All collocation methods discussed before rewrite the system dynamics as nonlinear first order state equations. In this case it could be described as a bad habit. Although it makes it easier to read and code, it prevents the construction of accurate constraints that capture the behaviour of second order differential equations. Martín et al. [22] demonstrated that this reduced the maximum dynamical error accumulated in a segment (the maximum of Eqn. 4.5.8) with 1 to 2 orders of magnitude.

In second order trapezoidal collocation, integration is approximated with a third order polynomial:

$$\boldsymbol{q}(\tau)=a+b\tau+c\tau^2+d\tau^3, \tag{B.4.1}$$

$$\dot{\boldsymbol{q}}(\tau)=b+2c\tau+3d\tau^2, \tag{B.4.2}$$

$$\ddot{\boldsymbol{q}}(\tau)=2c+6d\tau. \tag{B.4.3}$$

There are four unknowns (a, b, c and d), so find four knowns:

$$\boldsymbol{q}(0) = \boldsymbol{p}_k, \tag{B.4.4}$$
$$\dot{\boldsymbol{q}}(0) = \boldsymbol{v}_k, \tag{B.4.5}$$
$$\ddot{\boldsymbol{q}}(0) = \boldsymbol{g}_k, \tag{B.4.6}$$
$$\ddot{\boldsymbol{q}}(h_k) = \boldsymbol{g}_{k+1}. \tag{B.4.7}$$

Solve:

$$\boldsymbol{p}_k = a, \tag{B.4.8}$$
$$\boldsymbol{v}_k = b, \tag{B.4.9}$$
$$\boldsymbol{g}_k = 2c, \tag{B.4.10}$$
$$\boldsymbol{g}_{k+1} = 2c + 6h_k d. \tag{B.4.11}$$

Rewriting gives:

$$a = \boldsymbol{p}_k, \tag{B.4.12}$$
$$b = \boldsymbol{v}_k, \tag{B.4.13}$$
$$c = \frac{1}{2}\boldsymbol{g}_k, \tag{B.4.14}$$
$$d = \frac{1}{6h_k}(\boldsymbol{g}_{k+1} - \boldsymbol{g}_k). \tag{B.4.15}$$

Substituting these unknowns in B.4.1, B.4.2 and B.4.3 results in the equations that may be used to interpolate:

$$\boldsymbol{q}(\tau) = \boldsymbol{p}_k + \boldsymbol{v}_k\tau + \frac{\tau^2}{2}\boldsymbol{g}_k + \frac{\tau^3}{6h_k}(\boldsymbol{g}_{k+1} - \boldsymbol{g}_k), \tag{B.4.16}$$
$$\dot{\boldsymbol{q}}(\tau) = \boldsymbol{v}_k + \boldsymbol{g}_k\tau + \frac{\tau^2}{2h_k}(\boldsymbol{g}_{k+1} - \boldsymbol{g}_k), \tag{B.4.17}$$
$$\ddot{\boldsymbol{q}}(\tau) = \boldsymbol{g}_k + \frac{\tau}{h_k}(\boldsymbol{g}_{k+1} - \boldsymbol{g}_k). \tag{B.4.18}$$

To find the collocation constraints, evaluate $\boldsymbol{p}_{k+1} = \boldsymbol{q}(h_k)$ and $\boldsymbol{v}_{k+1} = \dot{\boldsymbol{q}}(h_k)$:

$$\boldsymbol{p}_{k+1} = \boldsymbol{p}_k + \boldsymbol{v}_k h_k + \frac{h_k^2}{2}\boldsymbol{g}_k + \frac{h_k^2}{6}(\boldsymbol{g}_{k+1} - \boldsymbol{g}_k), \tag{B.4.19}$$
$$\boldsymbol{p}_{k+1} = \boldsymbol{p}_k + \boldsymbol{v}_k h_k + \frac{h_k^2}{6}(\boldsymbol{g}_{k+1} + 2\boldsymbol{g}_k). \tag{B.4.20}$$

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \boldsymbol{g}_k h_k + \frac{h_k}{2}(\boldsymbol{g}_{k+1} - \boldsymbol{g}_k), \tag{B.4.21}$$
$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \frac{h_k}{2}(\boldsymbol{g}_{k+1} + \boldsymbol{g}_k). \tag{B.4.22}$$

Appendix C

NLP Without Succinct Notation

$$
\begin{aligned}
\underset{\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{u},\boldsymbol{\lambda},\boldsymbol{\gamma}}{\text{minimize}} \quad & \sum_{k=1}^{N}\left(w_{\boldsymbol{u}}\boldsymbol{u}_k^{\top}\boldsymbol{u}_k + w_{\boldsymbol{\lambda}}\boldsymbol{\lambda}_k^{\top}\boldsymbol{\lambda}_k\right) && \text{(C.0.1a)}\\
\text{subject to} \quad & \boldsymbol{q}^{\mathrm{LB}} \le \boldsymbol{q}_k \le \boldsymbol{q}^{\mathrm{UB}}, && \text{(C.0.1b)}\\
& \dot{\boldsymbol{q}}^{\mathrm{LB}} \le \dot{\boldsymbol{q}}_k \le \dot{\boldsymbol{q}}^{\mathrm{UB}}, && \text{(C.0.1c)}\\
& \boldsymbol{u}^{\mathrm{LB}} \le \boldsymbol{u}_k \le \boldsymbol{u}^{\mathrm{UB}}, && \text{(C.0.1d)}\\
& \boldsymbol{q}_1^{\mathrm{LB}} \le \boldsymbol{q}_1 \le \boldsymbol{q}_1^{\mathrm{UB}}, && \text{(C.0.1e)}\\
& \boldsymbol{q}_N^{\mathrm{LB}} \le \boldsymbol{q}_N \le \boldsymbol{q}_N^{\mathrm{UB}}, && \text{(C.0.1f)}\\
& \dot{\boldsymbol{q}}_1^{\mathrm{LB}} \le \dot{\boldsymbol{q}}_1 \le \dot{\boldsymbol{q}}_1^{\mathrm{UB}}, && \text{(C.0.1g)}\\
& \dot{\boldsymbol{q}}_N^{\mathrm{LB}} \le \dot{\boldsymbol{q}}_N \le \dot{\boldsymbol{q}}_N^{\mathrm{UB}}, && \text{(C.0.1h)}\\
& \boldsymbol{0} \le \boldsymbol{\lambda}_k \le \boldsymbol{\infty}, && \text{(C.0.1i)}\\
& \boldsymbol{0} \le \boldsymbol{\gamma}_k \le \boldsymbol{\infty}, && \text{(C.0.1j)}\\
& \boldsymbol{q}_{k+1} - \boldsymbol{q}_k - \frac{h_k}{2}\left(\dot{\boldsymbol{q}}_k + \dot{\boldsymbol{q}}_{k+1}\right) = 0, && \text{(C.0.1k)}\\
& \dot{\boldsymbol{q}}_{k+1} - \dot{\boldsymbol{q}}_k - \frac{h_k}{2}\left(\mathbf{M}^{-1}(\boldsymbol{q}_k)\mathbf{F}(\boldsymbol{q}_k,\dot{\boldsymbol{q}}_k,\boldsymbol{u}_k,\boldsymbol{\lambda}_k) + \mathbf{M}^{-1}(\boldsymbol{q}_{k+1})\mathbf{F}(\boldsymbol{q}_{k+1},\dot{\boldsymbol{q}}_{k+1},\boldsymbol{u}_{k+1},\boldsymbol{\lambda}_{k+1})\right) = 0, && \text{(C.0.1l)}\\
& \boldsymbol{\phi}(\boldsymbol{q_k}) \ge \boldsymbol{0}, && \text{(C.0.1m)}\\
& \boldsymbol{\phi}^{\top}(\boldsymbol{q}_{k+1})\boldsymbol{\lambda}_k^{\perp} \le \boldsymbol{0}, && \text{(C.0.1n)}\\
& \boldsymbol{\phi}^{\top}(\boldsymbol{q}_N)\boldsymbol{\lambda}_N^{\perp} \le \boldsymbol{0}, && \text{(C.0.1o)}\\
& \mu\boldsymbol{\lambda}_k^{\perp} - \boldsymbol{\lambda}_k^{\parallel -} - \boldsymbol{\lambda}_k^{\parallel +} \ge \boldsymbol{0}, && \text{(C.0.1p)}\\
& \left[\mu\boldsymbol{\lambda}_k^{\perp} - \boldsymbol{\lambda}_k^{\parallel -} - \boldsymbol{\lambda}_k^{\parallel +}\right]^{\top}\boldsymbol{\gamma}_k \le \boldsymbol{0}, && \text{(C.0.1q)}\\
& \boldsymbol{\gamma}_k - \mathbf{J}^{\top}(\boldsymbol{q}_k)\dot{\boldsymbol{q}}_k \ge \boldsymbol{0}, && \text{(C.0.1r)}\\
& \left[\boldsymbol{\gamma}_k - \mathbf{J}^{\top}(\boldsymbol{q}_k)\dot{\boldsymbol{q}}_k\right]^{\top}\lambda_k^{\parallel +} \le \boldsymbol{0}, && \text{(C.0.1s)}\\
& \boldsymbol{\gamma}_k + \mathbf{J}^{\top}(\boldsymbol{q}_k)\dot{\boldsymbol{q}}_k \ge \boldsymbol{0}, && \text{(C.0.1t)}\\
& \left[\boldsymbol{\gamma}_k + \mathbf{J}^{\top}(\boldsymbol{q}_k)\dot{\boldsymbol{q}}_k\right]^{\top}\lambda_k^{\parallel -} \le \boldsymbol{0}. && \text{(C.0.1u)}
\end{aligned}
$$

Appendix D

Analytic Gradients for Dynamics

To reduce computational cost gradients — often also referred to as Jacobians — were added. Every state is dependent on the decision variables of k, constraints 4.3.1f and 4.3.1g also depend on $k+1$. Therefore $\frac{\partial J(\boldsymbol{z})}{\partial \boldsymbol{z}}$ (3.2.1a), $\frac{\partial \mathbf{f}(\boldsymbol{z})}{\partial \boldsymbol{z}}$ (3.2.1c), and $\frac{\partial \mathbf{g}(\boldsymbol{z})}{\partial \boldsymbol{z}}$ (3.2.1d) will mostly be sparse.

Most gradients are, although time consuming to implement, fairly trivial. However, the gradients for 4.3.1f require a bit more thought. The difficult part lies in taking the gradient of Eqn. 3.3.3, as it includes the analytical computation of $\mathbf{M}^{-1}$.

$$\frac{\partial \ddot{\boldsymbol{q}}_k}{\partial \boldsymbol{z}_k} = \frac{\partial}{\partial \boldsymbol{z}_k}\left(\mathbf{M}_k^{-1}\mathbf{F}_k\right). \tag{D.0.1}$$

But computing the analytical inverse for a large $\mathbf{M}$ may become problematic. This problem can be solved by using matrix calculation definitions [31]. If $\mathbf{M}$ is an invertible $m \times m$ matrix whose elements are functions of the scalar parameter $z_{k,n}$. Then

$$\frac{\partial \mathbf{M}^{-1}}{\partial z_{k,n}} = -\mathbf{M}^{-1}\frac{\partial \mathbf{M}}{\partial z_{k,n}}\mathbf{M}^{-1}. \tag{D.0.2}$$

As shown in [18], it is now possible to redefine the problem as:

$$\frac{\partial \ddot{\boldsymbol{q}}_k}{\partial z_{k,n}} = \left(-\mathbf{M}_k^{-1}\frac{\partial \mathbf{M}_k}{\partial z_{k,n}}\mathbf{M}_k^{-1}\right)\mathbf{F}_k + \mathbf{M}_k^{-1}\left(\frac{\partial \mathbf{F}_k}{\partial z_{k,n}}\right). \tag{D.0.3}$$

Substituting Eqn. 3.3.3 in results in:

$$\frac{\partial \ddot{\boldsymbol{q}}_k}{\partial z_{k,n}} = \mathbf{M}_k^{-1}\left(-\frac{\partial \mathbf{M}_k}{\partial z_{k,n}}\ddot{\boldsymbol{q}}_k + \frac{\partial \mathbf{F}_k}{\partial z_{k,n}}\right). \tag{D.0.4}$$

This process may then be repeated for every variable in $\boldsymbol{z}_k$. This reduces the complexity of the problem as $\ddot{\boldsymbol{q}}_k$, $\mathbf{M}_k^{-1}$, $\frac{\partial \mathbf{M}_k}{\partial z_{k,n}}$ and $\frac{\partial \mathbf{F}_k}{\partial z_{k,n}}$ are the only properties that have to be computed during runtime.

Appendix E

Additional Results

E.1 Gait

Experimental results showed that a goal of 2 meters in 2 seconds should be feasible and result in 2 or more steps. So with these parameters the number of knot points was increased, the results of which can be seen in Table E.1 and Figure 5.1a.

There is a visible positive trend for the solver time when the number of knot points increases. The collocation error has a steep drop for the first 30 knot points, but remains around the same values for higher numbers of knot points. There does not seem to be a clear correlation between the number of collocation points and the value of the objective function. The results satisfy the constraints better for 22 or more knot points.

What is notable from animations — but difficult to objectively quantify or display in still images — is that few (≤ 20) knot points results in 2 large steps, while more knot points result in multiple tiny steps.

Results of running the algorithm with 4 knot points and using the result as the initial guess for a second optimization with 49 knot points are shown in Figure E.1 and Figure 5.2 respectively.

The initial optimization using 4 knot points took 44 seconds. Constraint violations are clearly visible from the foot going through the ground in Figure E.1a. The collocation error is shown in Figure E.1b should ideally equal zero at all times, but it is only satisfied at the knot points — hence the name. The value of the objective function was $1.8{\cdot}10^6$ and the maximum deviation from the dynamics was $1.8{\cdot}10^2$.

Refining the result with 49 knot points took $2.3{\cdot}10^3$ seconds. It resulted in a more realistic looking gait, as can be seen in Figure 5.2a. The feet seem to slip, but in simulation it can be seen that it is merely repositioning its feet. In Figure 5.2b it can be seen that the average collocation error is much lower after the refined optimization. The variance of the collocation error is higher than with 4 knot points, the areas with increased collocation errors correspond with the timing of foot placement. The value of the objective function was $1.9{\cdot}10^6$ and the maximum deviation from the dynamics decreased to 3.7.

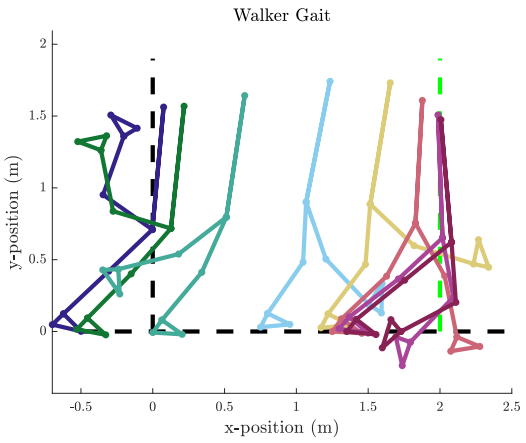
Positions, velocities, control and contact information of the generated gait can be seen in Figures E.2, E.3, E.4 and E.5 respectively.

E.2 Perturbation

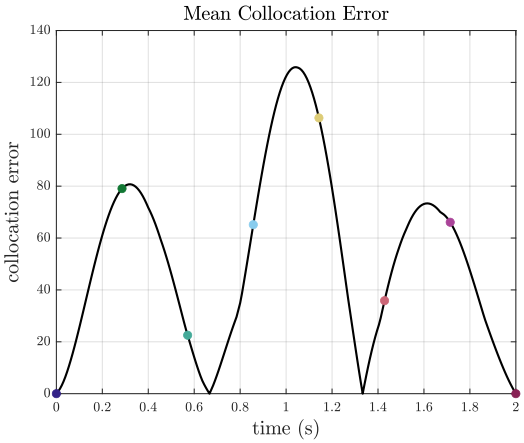
Positions, velocities, control and contact information of the generated trajectory after a perturbation can be seen in Figures E.6, E.7, E.8, E.9 and E.10a, and E.10b respectively.

Table E.1: Performance values for gait generation with varying number of knot points (N). Performance is expressed in solver time (t), objective value (J) and the maximum absolute error of the dynamics ($\boldsymbol{\eta}$). Restoration fails if the solver fails to find a feasible point that was acceptable to the filter line search for the original problem, which can happen when the problem does not satisfy the constraint qualification. Maximum iteration indicates that the set limit of $1{\cdot}10^4$ iterations was reached. Acceptable means that the largest constraint violation was on the order of $1{\cdot}10^{-3}$. Optimal means that the largest constraint violation was on the order of $1{\cdot}10^{-4}$ or smaller.

N	t (s)	J	$\max(\boldsymbol{\eta})$	**Exit code**
4	$8.0{\cdot}10^{1}$	$2.0{\cdot}10^{6}$	$1.1{\cdot}10^{2}$	restoration failed
6	$1.1{\cdot}10^{2}$	$2.1{\cdot}10^{6}$	$1.1{\cdot}10^{2}$	restoration failed
8	$1.0{\cdot}10^{2}$	$2.3{\cdot}10^{6}$	$4.2{\cdot}10^{1}$	restoration failed
10	$6.4{\cdot}10^{2}$	$2.0{\cdot}10^{6}$	$3.2{\cdot}10^{1}$	restoration failed
12	$7.2{\cdot}10^{2}$	$1.5{\cdot}10^{6}$	$1.5{\cdot}10^{1}$	restoration failed
14	$3.3{\cdot}10^{2}$	$2.8{\cdot}10^{6}$	$2.7{\cdot}10^{1}$	restoration failed
16	$7.1{\cdot}10^{2}$	$1.2{\cdot}10^{6}$	$7.7{\cdot}10^{0}$	restoration failed
18	$1.5{\cdot}10^{3}$	$1.3{\cdot}10^{6}$	$6.2{\cdot}10^{0}$	maximum iterations
20	$7.2{\cdot}10^{2}$	$2.7{\cdot}10^{6}$	$2.0{\cdot}10^{1}$	restoration failed
22	$1.4{\cdot}10^{3}$	$3.3{\cdot}10^{6}$	$1.8{\cdot}10^{1}$	acceptable
24	$8.5{\cdot}10^{2}$	$3.4{\cdot}10^{6}$	$9.6{\cdot}10^{0}$	restoration failed
26	$1.9{\cdot}10^{3}$	$2.7{\cdot}10^{6}$	$1.3{\cdot}10^{1}$	restoration failed
28	$4.4{\cdot}10^{2}$	$2.5{\cdot}10^{6}$	$7.4{\cdot}10^{0}$	optimal
30	$8.9{\cdot}10^{2}$	$1.5{\cdot}10^{6}$	$4.5{\cdot}10^{0}$	optimal
32	$6.7{\cdot}10^{2}$	$1.4{\cdot}10^{6}$	$4.0{\cdot}10^{0}$	acceptable
34	$1.6{\cdot}10^{3}$	$2.2{\cdot}10^{6}$	$5.5{\cdot}10^{0}$	acceptable
36	$1.4{\cdot}10^{3}$	$1.9{\cdot}10^{6}$	$2.0{\cdot}10^{0}$	acceptable
38	$2.8{\cdot}10^{3}$	$2.1{\cdot}10^{6}$	$4.1{\cdot}10^{0}$	restoration failed
40	$8.1{\cdot}10^{2}$	$2.8{\cdot}10^{6}$	$4.0{\cdot}10^{0}$	acceptable
42	$2.3{\cdot}10^{3}$	$1.4{\cdot}10^{6}$	$2.9{\cdot}10^{0}$	acceptable
44	$2.5{\cdot}10^{3}$	$1.8{\cdot}10^{6}$	$3.9{\cdot}10^{0}$	acceptable
46	$1.2{\cdot}10^{3}$	$2.4{\cdot}10^{6}$	$4.5{\cdot}10^{0}$	acceptable
48	$2.6{\cdot}10^{3}$	$1.5{\cdot}10^{6}$	$1.4{\cdot}10^{0}$	acceptable
50	$3.1{\cdot}10^{3}$	$2.4{\cdot}10^{6}$	$4.2{\cdot}10^{0}$	acceptable
52	$5.4{\cdot}10^{3}$	$1.7{\cdot}10^{6}$	$2.0{\cdot}10^{0}$	maximum iterations
54	$1.4{\cdot}10^{3}$	$1.7{\cdot}10^{6}$	$1.9{\cdot}10^{0}$	acceptable
56	$4.0{\cdot}10^{3}$	$2.2{\cdot}10^{6}$	$3.3{\cdot}10^{0}$	acceptable
58	$3.8{\cdot}10^{3}$	$2.2{\cdot}10^{6}$	$2.5{\cdot}10^{0}$	optimal
60	$2.1{\cdot}10^{3}$	$1.7{\cdot}10^{6}$	$1.9{\cdot}10^{0}$	acceptable



(a) Eight frames of the generated gait trajectory. The initial position of the CoM is indicated with the vertical black line (x=0) and the goal with the vertical green line (x=2). The floor is indicated with the horizontal line (y=0).



(b) The mean collocation error of the generated gait trajectory. The mean of variables with different units (meters and radians) was taken, therefore the mean collocation error has no meaningful unit. The coloured dots correspond with the frames of Figure E.1a.

Figure E.1: Generated gait with with 4 knot points and linear initialization. Time goes from **cobalt blue** to **raspberry red**.

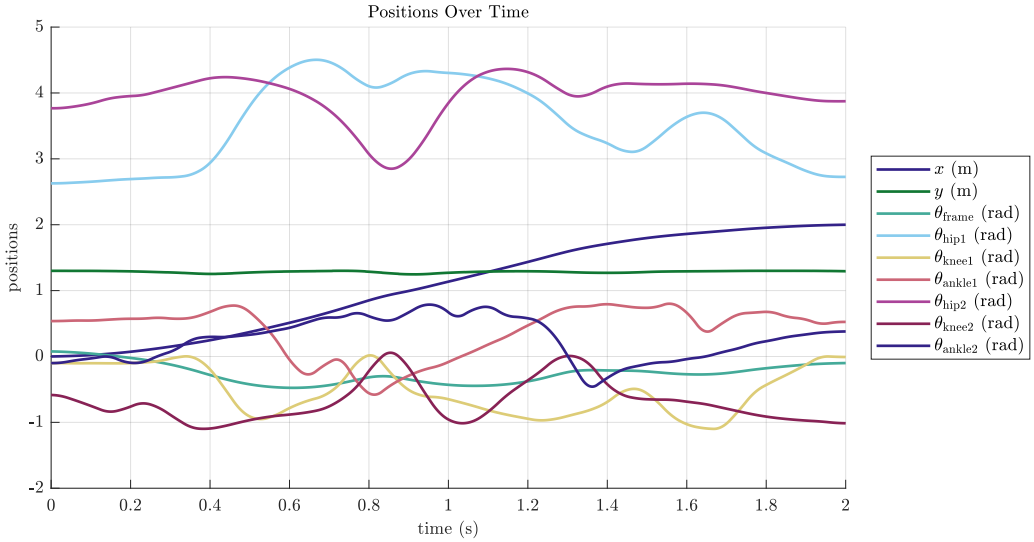


Figure E.2: Generalized coordinates over time of the generated gait.

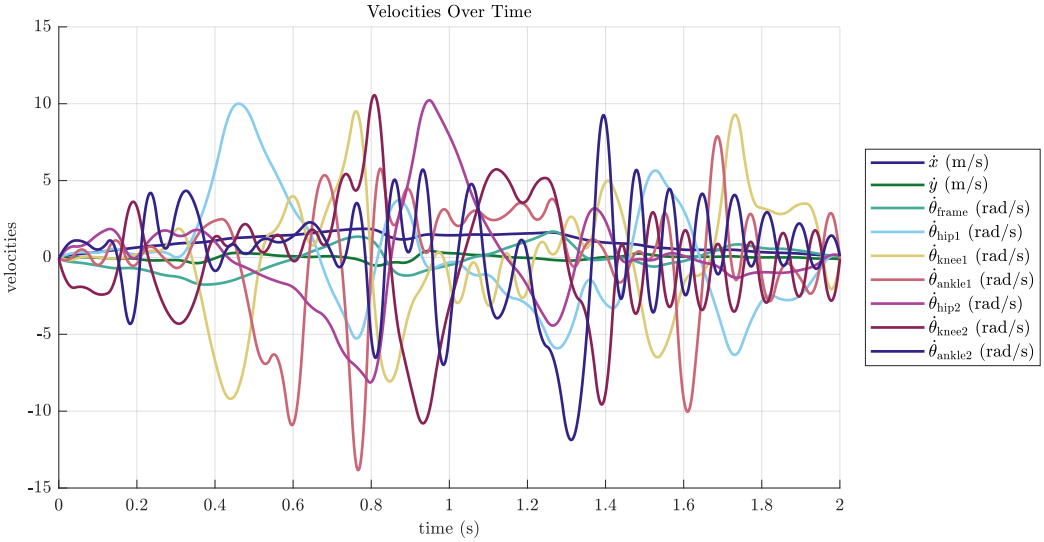


Figure E.3: Velocities of the generalized coordinates over time of the generated gait.

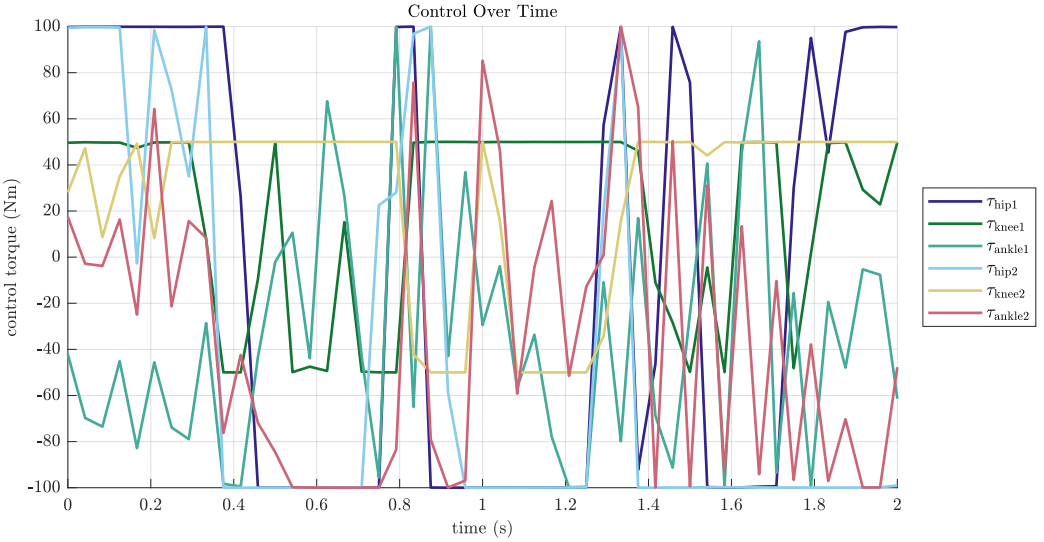


Figure E.4: Control over time of the generated gait.

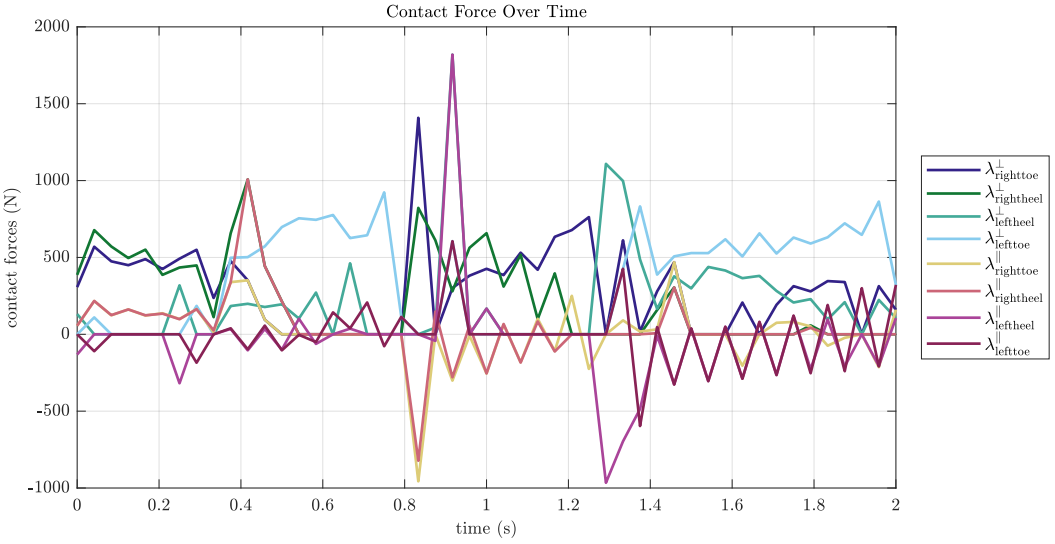


Figure E.5: Ground reaction forces over time of the generated gait.

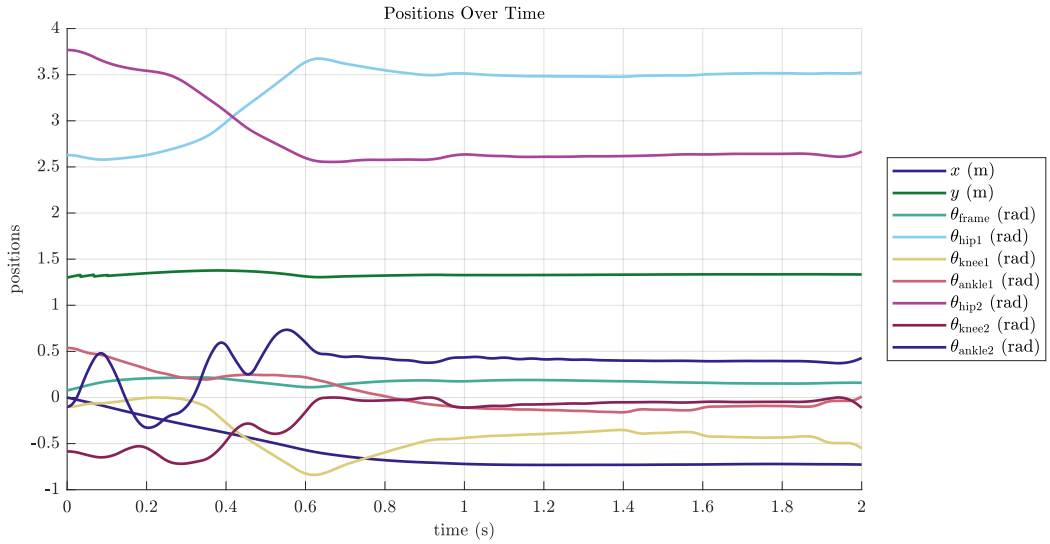


Figure E.6: Generalized coordinates over time of the generated trajectory to resist a perturbation.

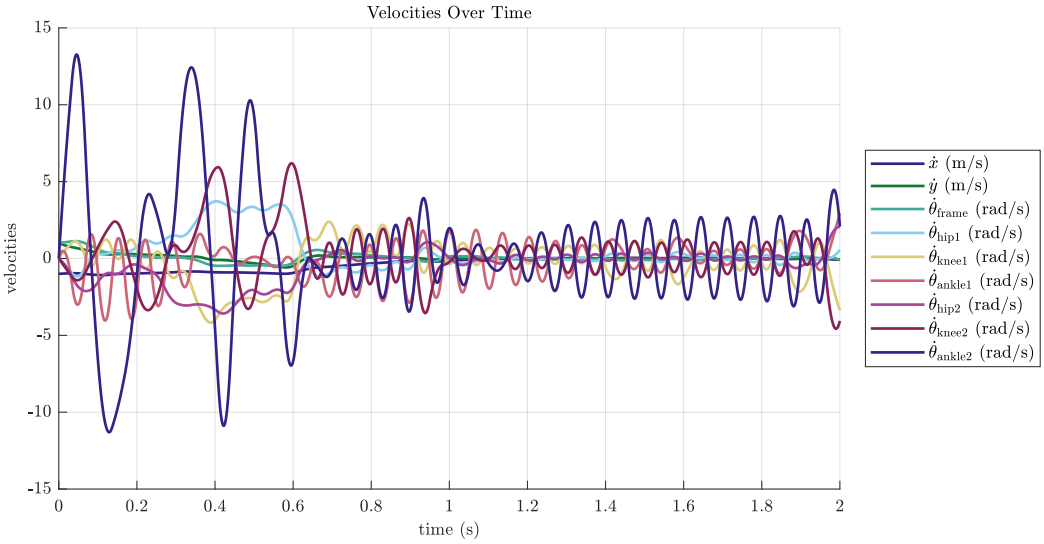


Figure E.7: Velocities of the generalized coordinates over time of the generated trajectory to resist a perturbation.

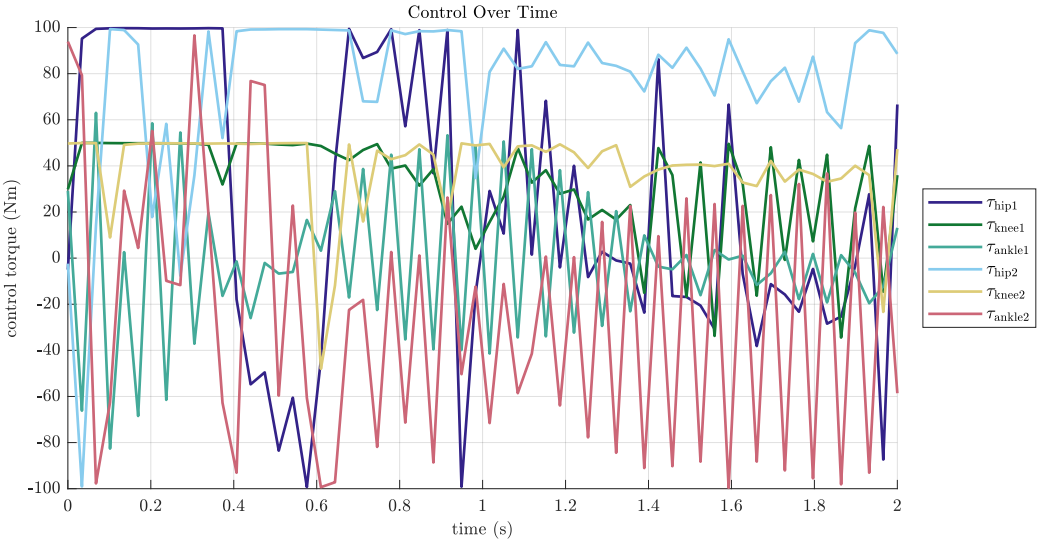


Figure E.8: Control over time of the generated trajectory to resist a perturbation.

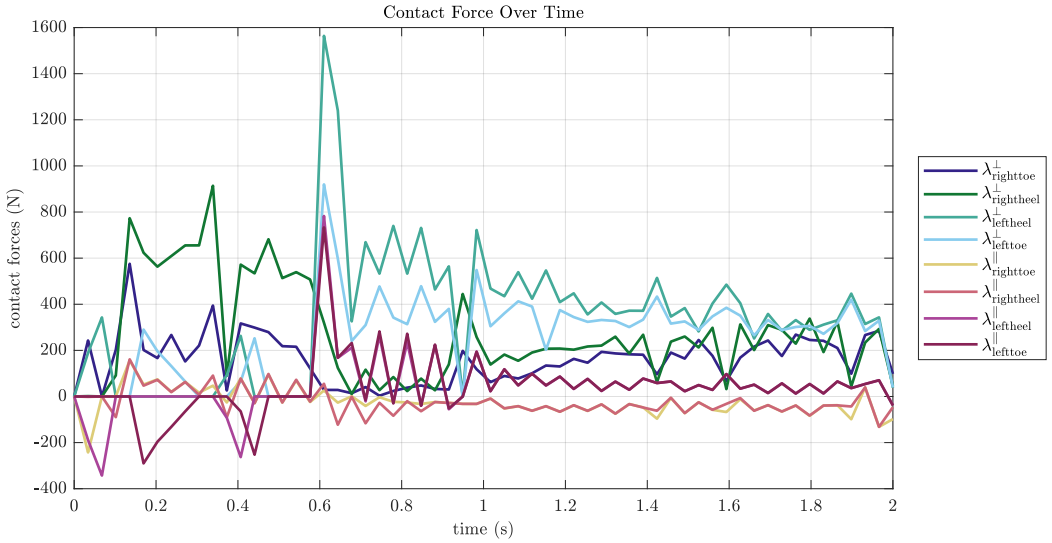
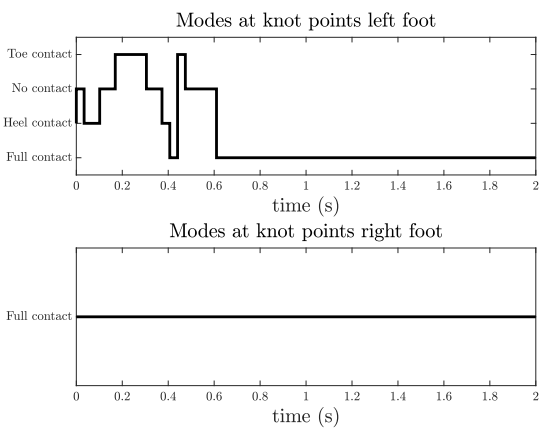
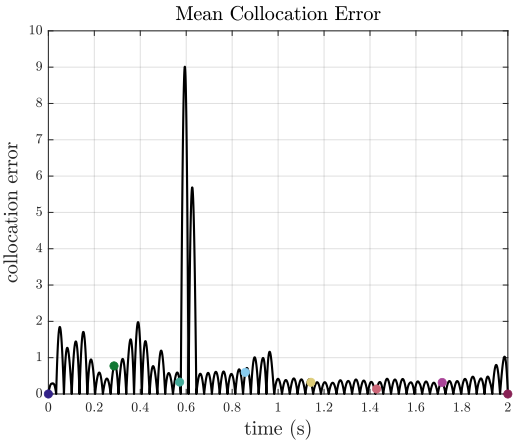


Figure E.9: Contact forces over time of the generated trajectory to resist a perturbation.



(a) Contact modes of both feet over time.



(b) Mean collocation error over time.

Figure E.10: Contact modes and collocation error of the generated trajectory to resist a perturbation.

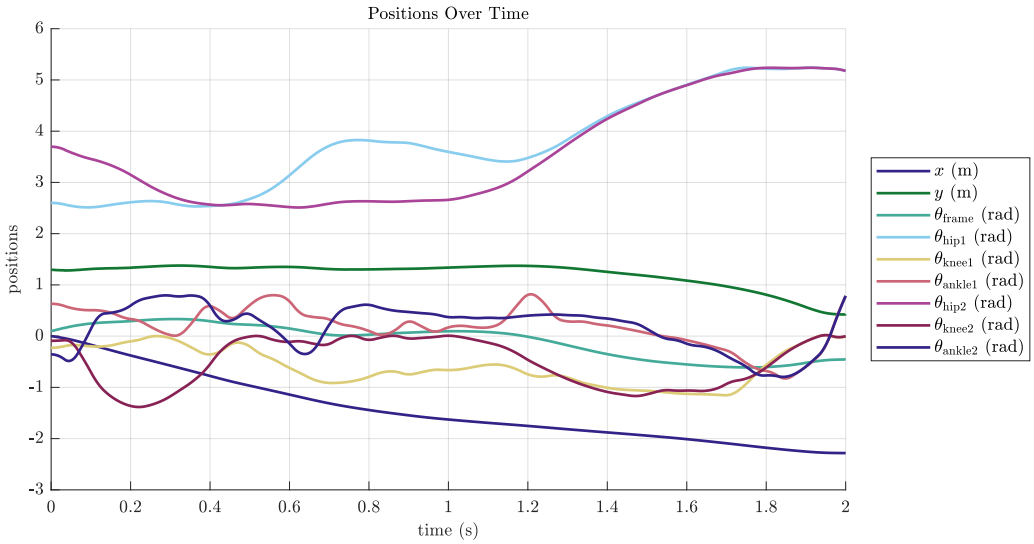


Figure E.11: Generalized coordinates over time of the generated falling trajectory.

E.3 Falling

Positions, velocities, control and contact information of the generated optimal falling trajectory can be seen in Figures E.11, E.12, E.13, E.14, and E.15 respectively.

E.3.1 Experimental Results Optimal Falling

Experimental results showed that an initial condition with a backwards rotation on the center of mass of around 14 degrees and rotational velocity of 36 degrees per second resulted in falling behaviour with acceptable feasibility.

Generating trajectories of falling backwards (Figure E.16) took 23 minutes and resulted in an objective value of $3.5{\cdot}10^{6}$ and maximum dynamic error of $4.5{\cdot}10^{-1}$. It is notable that it seems like the walker is sitting down. And the dynamic error is relatively low compared to the gait trajectories.

By coincidence, the algorithm found a way to fall forward (Figure E.17) with the conditions meant for falling backwards. It took 38 minutes, and resulted in an objective value of $7.3{\cdot}10^{6}$ and maximum dynamic error of $7.9{\cdot}10^{-1}$. It is notable how it tries to break its fall will its knees. The objective function also indicates that this is a worse solution to the problem than falling backwards. It is also still in motion, if the simulation time span was larger it would surely have an impact with its head.

The results shown in Figures E.16 and E.17 have nonzero velocities at the final point in time.

While searching for appropriate initial conditions for falling, it was notable how well the algorithm was able to stay on its feet or stumble backwards before falling over. Examples are shown in Figure E.18.

E.4 Recovery

Positions, velocities, control and error information of the generated trajectory to recover from a kneeled to neutral stance can be seen in Figures E.19, E.20, E.21 and E.22 respectively.

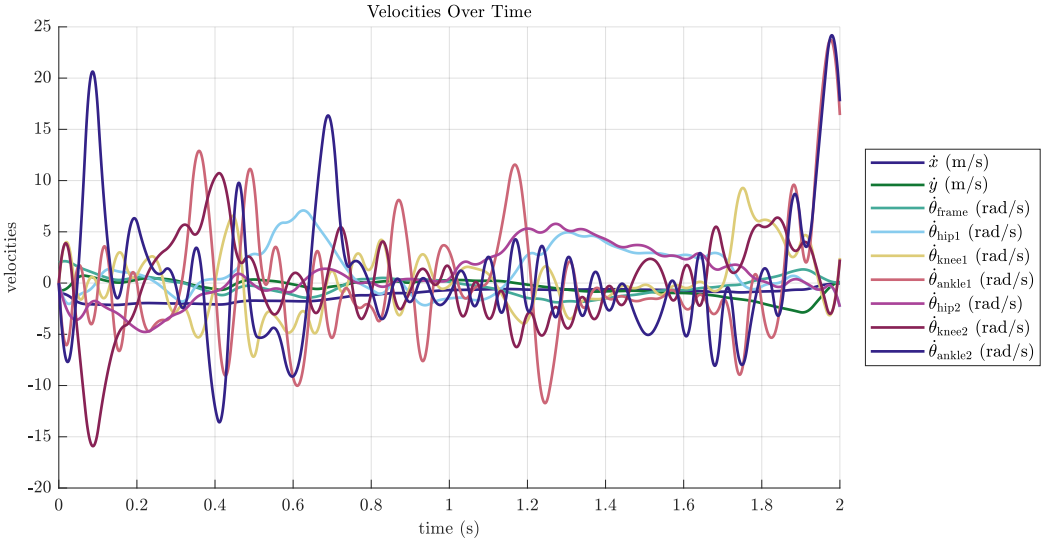


Figure E.12: Velocities of the generalized coordinates over time of the generated falling trajectory.

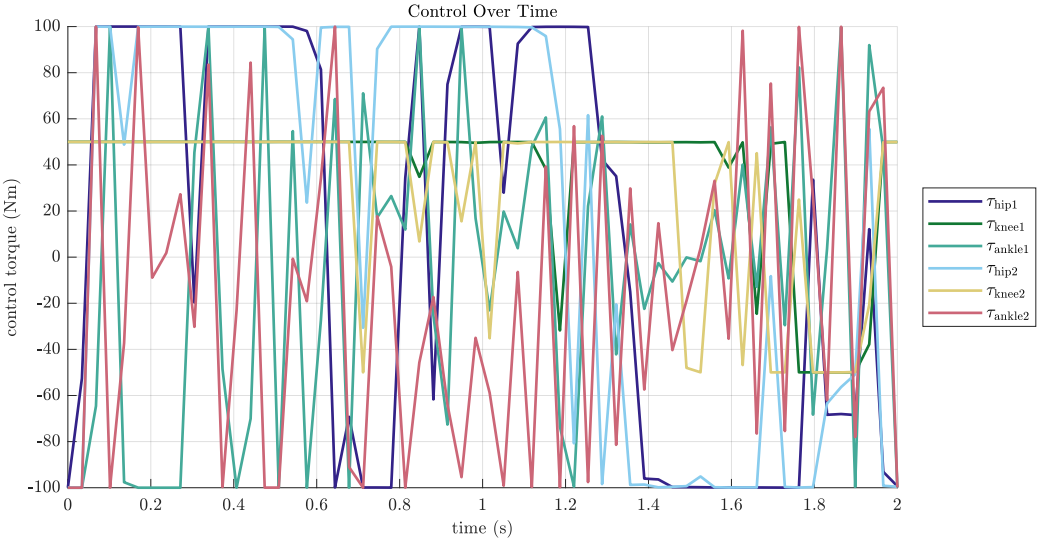


Figure E.13: Controls over time of the generated falling trajectory.

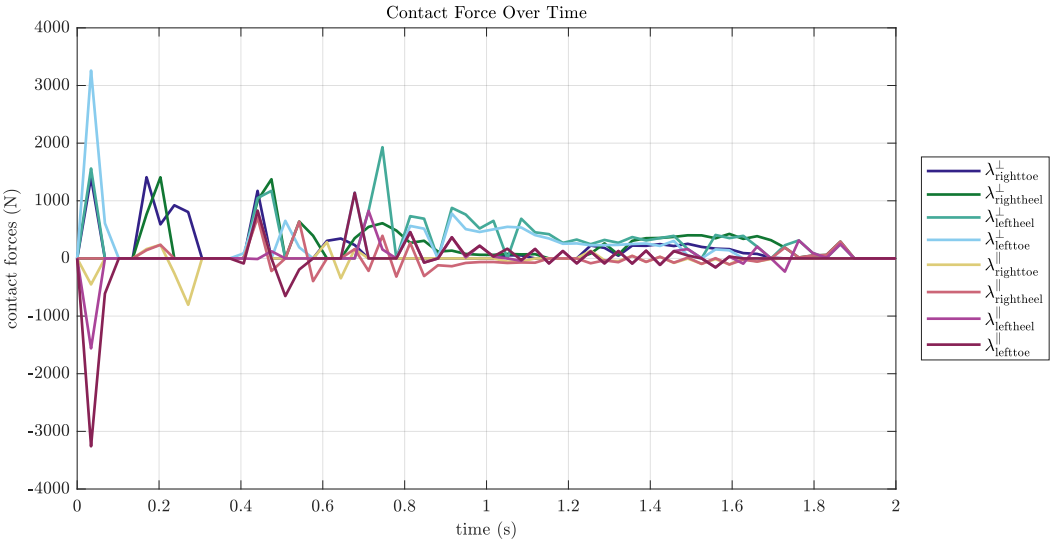


Figure E.14: Contact forces on the feet over time of the generated falling trajectory.

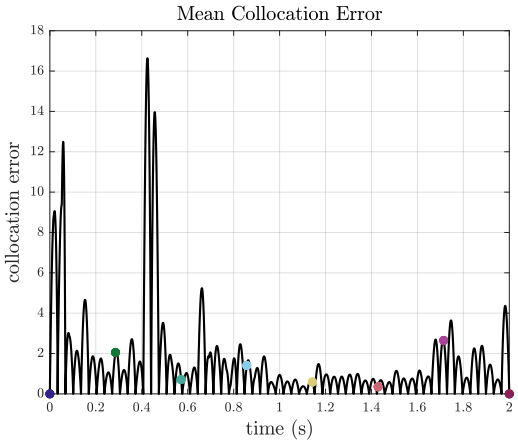
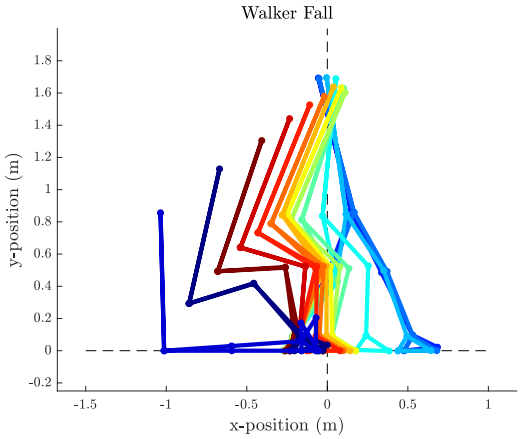
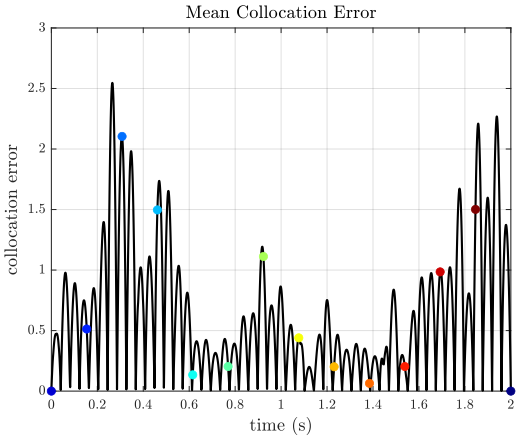


Figure E.15: Mean collocation error over time of the generated optimal falling trajectory.

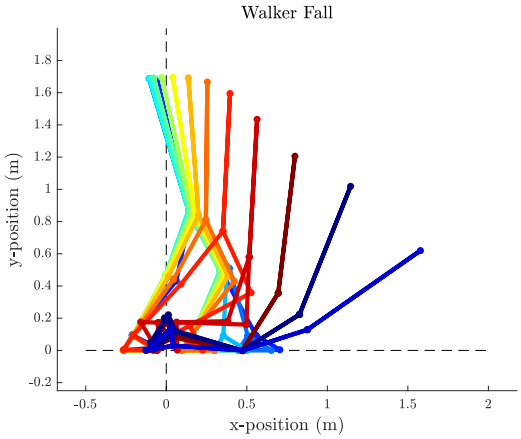


(a) Fourteen frames of the generated falling trajectory. The initial position of the CoM is indicated with the vertical line (x=0). The floor is indicated with the horizontal line (y=0).

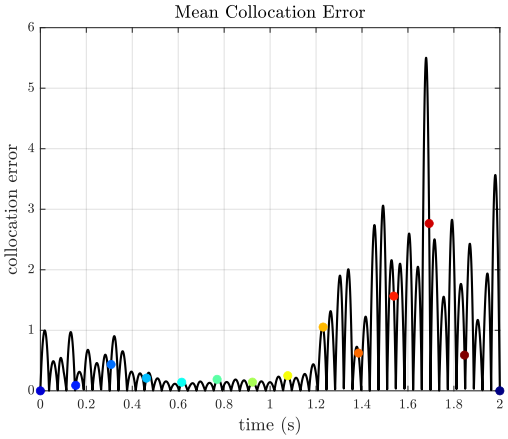


(b) The mean collocation error of the generated falling trajectory. The mean of variables with different units (meters and radians) was taken, therefore the mean collocation error has no meaningful unit. The coloured dots correspond with the frames of Figure E.16a.

Figure E.16: Optimization of falling backwards with 50 knot points.

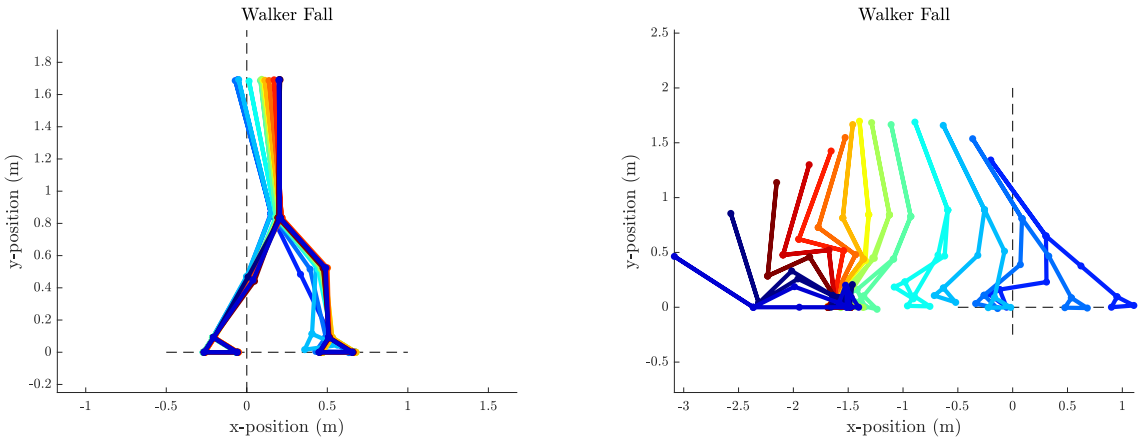


(a) Fourteen frames of the generated falling trajectory. The initial position of the CoM is indicated with the vertical line (x=0). The floor is indicated with the horizontal line (y=0).



(b) The mean collocation error of the generated falling trajectory. The mean of variables with different units (meters and radians) was taken, therefore the mean collocation error has no meaningful unit. The coloured dots correspond with the frames of Figure E.17a.

Figure E.17: Optimization of falling forwards with 54 knot points.



(a) Resisting perturbation with 44 knot points.

(b) Tumbling backwards with 38 knot points.

Figure E.18: Examples of other results from perturbations.

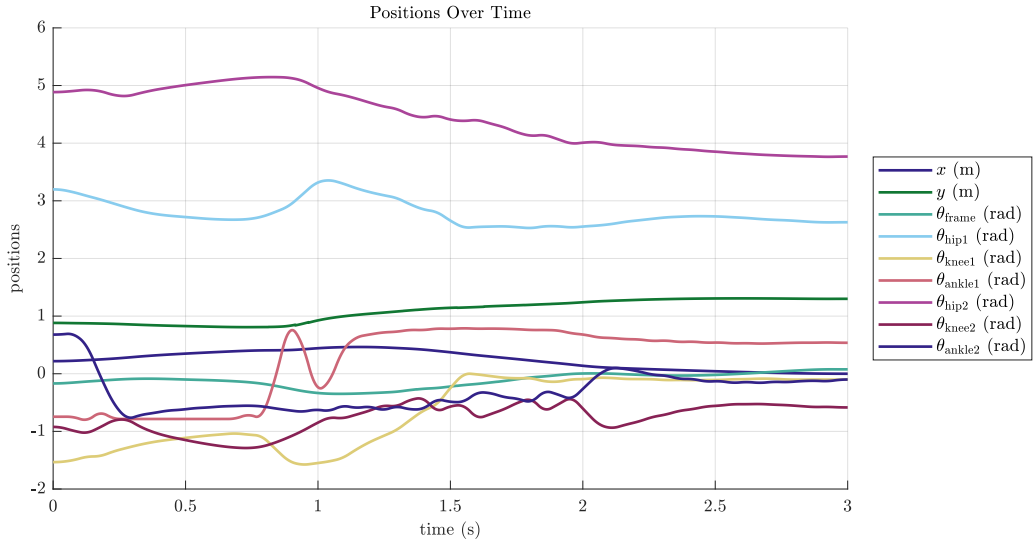


Figure E.19: Positions of the generalized coordinates over time of the generated recovery trajectory.

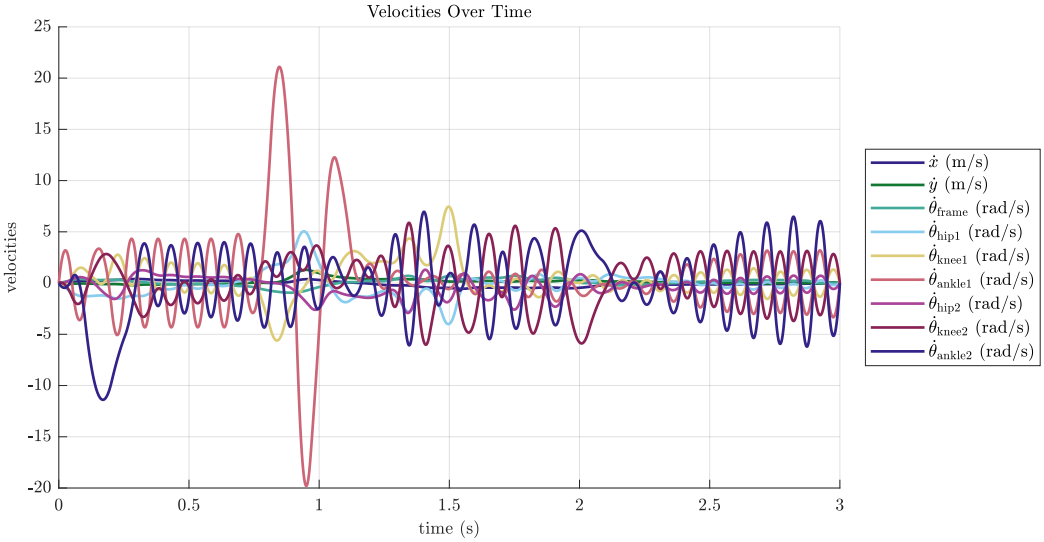


Figure E.20: Velocities of the generalized coordinates over time of the generated recovery trajectory.

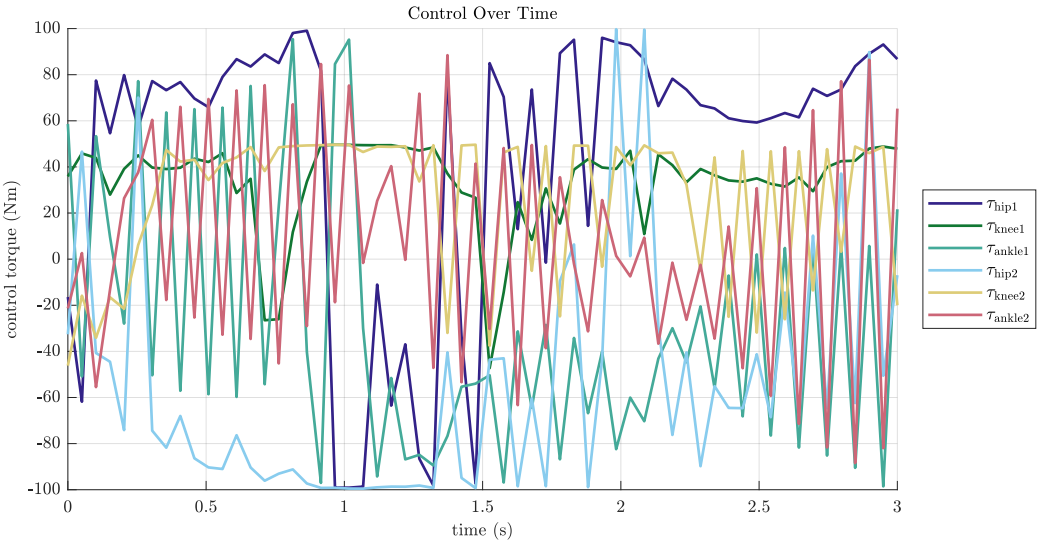


Figure E.21: Control over time of the generated recovery trajectory.

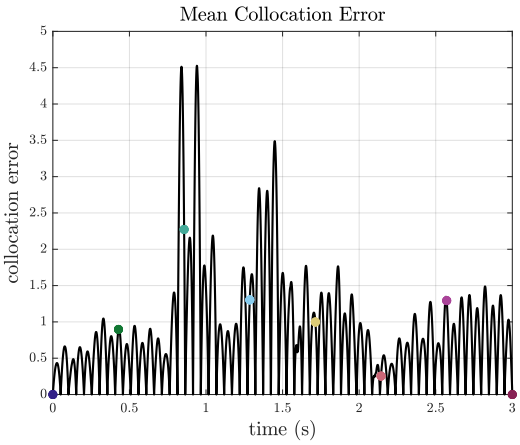


Figure E.22: Mean collocation error over time of the generated recovery trajectory.

Appendix F

Linear Combination for Ground Reaction Forces

If the angle of the floor can be described with a consistent function $s(\boldsymbol{q})$, it would enable the formulation of contact points with just two decision variables (d_1 and d_2):

$$\theta = s(\boldsymbol{q}) \tag{F.0.1}$$

$$\begin{bmatrix} \lambda^{\parallel} \\ \lambda^{\perp} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \left(d_1 \begin{bmatrix} -\mu \\ 1 \end{bmatrix} + d_2 \begin{bmatrix} \mu \\ 1 \end{bmatrix} \right) \tag{F.0.2}$$

This eliminates one decision variable per contact point for each knot point, as $\boldsymbol{\lambda}^{\parallel}$ no longer has to be split in a negative and positive direction. This new formulation also implicitly enforces a friction cone, removing constraints. The slip and friction can still be formulated in a similar fashion:

$$0 \leq \boldsymbol{\gamma} - \left[\mathbf{J}^{\parallel}(\boldsymbol{q})\right]^{\top} \dot{\boldsymbol{q}} \perp d_1 \geq 0, \tag{F.0.3}$$

$$0 \leq \boldsymbol{\gamma} + \left[\mathbf{J}^{\parallel}(\boldsymbol{q})\right]^{\top} \dot{\boldsymbol{q}} \perp d_2 \geq 0. \tag{F.0.4}$$

If this were to be implemented, it is important to normalize the decision variables before using them in the objective function. The current formulation would cause inconsistencies in penalizing the same GRF for different friction coefficients.