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

Robert A. Roos BSc

Systems & Control - Robotics and Mechatronics

Faculty of Engineering Technology
Department of Biomechanical Engineering

Graduation Committee
Prof. Dr. Ir. H. van der Kooij
Dr. Ir. A. Q. L. Keemink
Dr. Ir. R. G. K. M. Aarts

Document number
BE-736

July, 2020

UNIVERSITEIT TWENTE.

Abstract

To increase the walking performance of active exoskeletons, and legged robots in general, model predictive techniques can provide more dynamic and robust control. Model predictive control (MPC) can optimize a trajectory and corresponding inputs for a given model, in a given setting. MPC has been used for legged locomotion before, though previous works tend to use simplified dynamics and therefore do not optimize the robot joint torques. In this work we set out to design an MPC framework for legged locomotion that includes whole-body dynamics, such that trajectories can be generated that are efficient in joint torque. MPC based on implicit contact would be preferable, but showed infeasible with the direct collocation method. Instead an approach with explicit contact, based on predefined stance and swing phases, was used. This is an extension of TOWR [1], with the addition of whole-body dynamics. MuJoCo was used to model the system dynamics and IPOPT as numerical optimizer. The framework proved effective in locomotion generation and versatile to robot models, terrains and gaits. For a wide variety of robot types gait trajectories are optimized. However, the optimized trajectories are ineffective when applied in an MPC simulation.

Acknowledgments

I would like to thank my graduation committee for working with me during this assignment. This project has been fascinating and almost as exciting as it was frustrating at times!
Specifically my daily supervisor Arvid Keemink deserves a second round of thanks. He once again revealed himself to be a skilled researcher and capable engineer. More importantly, Arvid is just a great person to be working with.

Contents

1 - Introduction

1.1 Context and Problem Description

Tens of thousands of people suffer from spinal cord injuries world-wide [2]. Injuries to the spinal cord often leave a person with a type of paralysis, for example paraplegia (lower-body paralysis). While such injuries have become more treatable, people suffering from it will often still be left unable to walk on their own after their rehabilitation. Being restricted to a wheelchair limits the self-reliance of a person greatly and therefore affects their quality of live.
Powered exoskeletons (see Fig. 1) are an upcoming technology that can make a difference. A paraplegic could use an exoskeleton to regain the ability to walk by means of the actuated joints, attached to their own legs.

Figure 1: An example of a powered exoskeleton, developed by the Symbitron+ team at the University of Twente. Image from [3].

Actuated exoskeletons are in some respects similar to a bipedal robot. When an active exoskeleton is used by a paraplegic, the motions of the joints in the legs have to be initiated autonomously, comparable to stand-alone robot. The combination of exoskeleton and human user will therefore be considered as a biped robot with disturbances.

These legged robots require a locomotive control scheme for walking. A controller needs to produce a trajectory for the joints resulting in a desired gait, i.e. a walking pattern. The most primitive approach to locomotive control is to simply play back reference joint trajectories recorded from healthy human beings.
Although this is a simple method, the drawbacks are clear: the trajectories will not be flexible to small deviations (terrain irregularities, steering, etc.), they will not be robust to perturbations or errors and trajectories need to be somehow scaled from the human to the robot. To expand on the latter point, a joints trajectory for a human is likely optimal for that human, however it might not be suitable to a robot that has different body lengths, mass distributions or even different degrees of freedom.

Hence there is a need for a more autonomous and universal method of legged locomotive control. Model predictive control (MPC) is a method that has been applied successfully in this context and still has more potential. MPC is a method of optimal control where a sequence of

inputs are computed for the future, while optimizing some objective function (see Sec. 2.2 *Model Predictive Control*). Through MPC walking gaits could be produced that are efficient and stable, tailored to the situation.

The aim of this work is to design a control framework based on model predictive control for locomotion of legged robots.

1.2 Requirements and Objectives

The goal of the framework is to enable control for legged locomotion. A rough abstraction of walking control shows two possible levels:

1. Finding footholds over time.
2. Control the body to step through the footholds.

The goal of the new framework is to solve the second step, which is steering the robot through footholds efficiently. However, it is preferable if the footholds are planned together with the body control, such that no explicit foothold planning is needed.

Another requirement for the new platform is efficiency in the gait with regard to the whole body. In the end it is only the joint-specific values (e.g. joint accelerations or joint torques) that matter, hence the optimization needs to regard those explicitly. Optimization with respect to e.g. only center of mass acceleration will not be sufficient for the framework. (Illustrated by Fig. 2.)

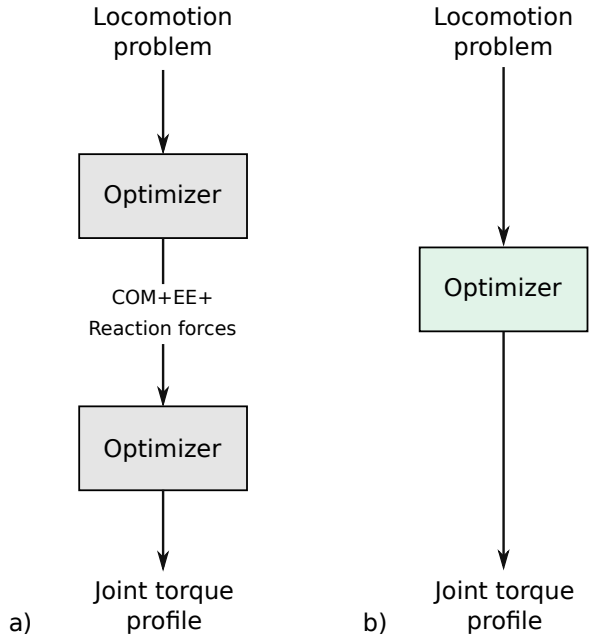

Figure 2: Optimization based on centroidal dynamics (a) requires a secondary optimization step to find joint torques, whereas they are immediately provided by a whole-body dynamics optimization (b). The joint torques found after the cascaded optimization (a) are not necessary optimal for the original problem.

Optimizations done with MPC methods are typically vulnerable to finding solutions in local minima, making them dependent on the initial guess. Hence an objective for the framework is to rely as little as possible on user input, such that a gait can be produced without a specific guess or constraint to steer the outcome.

Because there is a practical application to the framework, the speed of its solution is important. The faster it is, the more often it can be executed in a real system and the higher the quality of the trajectories. Achieving a rate of 10 to 100 Hz is desirable. Note that this is included as an objective, not a requirement. It is acceptable if the designed framework is a proof of concept and runs slower.

It is desired that convergence of the optimization is robust. It should be avoided that tedious tweaking of settings is needed to finish the optimization.

Lastly it would be useful if the framework was flexible to the kind of robot it optimizes for. This goes for the properties of the robot though also its structure, including for instance the number of limbs.

All requirements and objectives summarized together:

Requirements

1. Produce walking gaits with efficient actuation
2. Consider whole body in optimization

Objectives

1. Optimize footholds along with joint trajectory
2. Require little user input to produce a gait
3. Fast to compute
4. Robustness to optimization parameters
5. Variable number of legs
6. Easy to modify/replace robot model

1.3 Related Work

In this section a brief overview of relevant earlier works is given. Sec. 2 *Background* presents a more in-depth analysis of the components of these relevant works. The design concepts discussed in Sec. 3 *Design* will also often refer back to these. Table 1 provides a summary for each existing framework addressed here.

Opheusden [4] and Paas [5] are two projects which were performed earlier by this research group applying MPC for legged locomotion. Both optimized the walking trajectory of a five-link walker in the sagittal plane (see Fig. 3) and were in turn heavily based on an MPC manual written by Kelly [6].

The optimizations were based on analytical dynamics and were performed in MATLAB, though Paas [5] used an interface to IPOPT to speed up the optimization. Opheusden [4] focused on a practical example of obstacle crossing where Paas [5] focused on time and speed for real-time application. Paas [5] concluded that for the five-link biped real-time application should be possible and effective.

The largest shortcoming of these three works is the complete lack of dynamic ground contact. A walking trajectory that is optimized without forces from the ground will have little correlation to the real world. Moreover, only a single stance phase could be considered, double stance was ignored.

Hutter et al. [7, 8, 9] used trajectory optimization for their legged robot with wheels. Their optimization relies on predefined gaits and the acceleration profiles of the base and each end-effector are optimized. No system dynamics are included during optimization, allowing the optimization to be performed quickly. A whole-body controller then steers the robot according to the generated profiles.

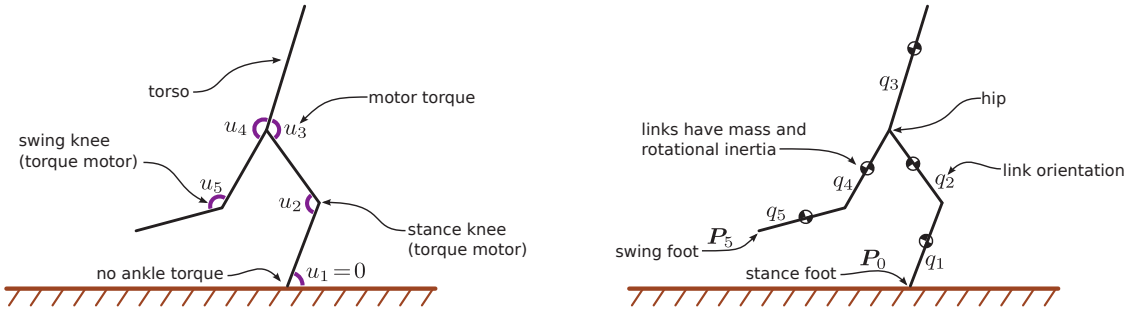

Figure 3: Biped five-link sagittal walker from Matthew Kelly [6]. The stance foot is hinged to the ground.

The optimization does not relate to torques or forces at all. Nonetheless, the results are good. The control scheme is tested on the actual robot and a series of challenges are passed successfully.

Sangbae et al. [10, 11] work on the control of a small quadruped robot called the Mini Cheetah. The optimization is based on only the dynamics of the base, the end-effectors and reaction forces. The 3D dynamics is linearized to result into a quadratic program that can be solved quickly. Ground contacts are explicit and defined by an input to the optimization.

Because only the body of the base is considered, no torque can be optimized. However, they mention the mass of the legs is only 10% of the total mass, so the assumed dynamics is reasonable. The results are good, the Cheetah has robust control and behaves dynamically.

Neunert et al. [12] aimed to apply a completely self-sufficient MPC approach to legged locomotion. Contacts are left implicit as part of the dynamics and dynamics of the complete system is considered. By considering the problem as unconstrained and by using a custom solver that is similar to iLQR the NLP can be solved at a speed that is unrivaled.

A large feat of this work is having no need for explicit contact. This allows the optimizer complete control over the motion, which is especially useful in light of disturbances.
There are a few drawbacks to this method although they are small. A brief time horizon is considered of only half a second, the constraints to the problem are soft (allowing the great speed) and, most importantly, the approach is highly complex and tailored. It is not straightforwardly adapted to other projects.

Tassa et al. [13] employed dynamic differential programming (specifically a variant of iLQG) to synthesize optimized trajectories[1]. Although they did not focus on legged locomotion, Tassa et al. do manage to optimize whole-body dynamics with implicit contact on a humanoid robot and even 'accidentally' produce a walking gait.

The success of their application of dynamic differential programming is somewhat surprising, because in theory DDP is not well suited to complex dynamic systems. Part of why it has worked well for them is probably the fact that no non-linear constraints were involved.

Posa et al. [14] focused on a method where no contact modes sequencing is required beforehand in the optimization. By adding a linear complementary problem (LCP) for contacts next to the optimization, contacts are solved implicitly. Sequential quadratic programming (SQP) is used to solve the problem. Posa et al. [14] test their framework on a biped sagittal walker, on which they find good results.

[1] A video of the results are posted here: `https://www.youtube.com/watch?v=anIsw2-Lbco`

No mention is made of the solve time of the optimization on the walker, nor is it clear how well this approach would scale to more complex systems or a system in 3D. Moreover it seems as only analytical models were used, limiting the range of systems that can be evaluated.

Winkler et al. [1, 15] have made an effective platform for legged locomotion called TOWR. It uses direct collocation, solved with IPOPT (wrapped in an interface library named IFOPT). Contacts are explicit and the key component of this platform is the phases are explicit for each end-effector. The duration of each phase is also an optimization variable, such that the initial guess does not completely fix the gait. Through varying the phases different gaits can still emerge. This is elaborated further in Sec. 2.5.4 *Timed Foothold Optimization*.

The base position, each end-effector positions and reaction forces are part of the optimization variables. These are constrained by centroidal dynamics, which is also the most significant shortcoming of this method. Although the platform is capable of optimizing a cost function, they choose to not use any costs. Only the set of constraints are solved.

	Dynamics	**Contact**	**Shortcomings**	**Platform**
Opheusden [4] and Paas [5]	Analytical, MATLAB generated	Hinged stance foot, no dynamic contact	Hybrid contacts	MATLAB & IPOPT
Hutter et al. [7, 8]	None	Explicit (only in position)	No dynamics	QuadProg++
Sangbae et al. [10, 11]	Centroidal	Explicit, optimized reaction forces	No whole-body dynamics	qpOASES
Neunert et al. [12]	Whole-body dynamics	Implicit (soft contacts in dynamics)	Short time horizon, unconstrained, complexity	Custom iLQR-NMPC solver
Tassa et al. [13]	Whole-body, using the MuJoCo simulator	Implicit (MuJoCo)	Not tested with locomotion	iLQG
Posa et al. [14]	Analytical	Implicit (contact parameters in LCP)	Limited dynamics description	SQP with additional LCP
Winkler et al. [1, 15]	Centroidal	Explicit (phase-based, optimizable)	No whole-body dynamics	IPOPT (TOWR)

Table 1: Brief overview of the evaluated previous works.

None of these existing frameworks suit the defined requirements and objectives directly. Most lack the use of whole-body dynamics and the ones that do include it do not have the desired flexibility in the model. If the work of Sangbae et al. or Winkler et al. were expanded to include whole-body dynamics, like in the platform from Tassa et al., they would fit the requirements.

2 - Background

In this section a brief overview of techniques for modelling dynamics systems is given first. Then the concept of model predictive control is explained. Numerical optimization in general is discussed as well as common approaches to MPC. Next, possible implementation levels of MPC are put forward. Finally, techniques used to speed up MPC to make it suitable for real-time application are summarized and different approaches to handling contact between bodies are evaluated.

2.1 Dynamic Modelling

The purpose of modelling dynamics is to predict the accelerations of a system in response to present torques and forces. The equation of motion is a system specific notation that correlates these forces to accelerations, velocities and positions:

$$\mathrm{M}(\mathbf{q}) + \mathrm{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathrm{N}(\mathbf{q}) = \boldsymbol{\tau} \tag{1}$$

Here $\mathbf{q}$ are the generalized coordinates, M the mass matrix, C the virtual forces, N the potential forces and $\boldsymbol{\tau}$ the generalized torque. In a general and brief notation the dynamics are described by the system state:

$$\mathbf{x} = \begin{pmatrix} \mathbf{q} \\ \dot{\mathbf{q}} \end{pmatrix} \tag{2}$$

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \tag{3}$$

The equations of motion can be found analytically by hand, through the Newton-Euler approach or Euler-Lagrange, though this is only feasible for simpler systems. The Euler-Lagrange approach was automated to avoid tedious derivations by hand. First the Lagrangian is defined as the difference between kinetic and potential energy:

$$\mathcal{L} = E_{kin} - E_{pot} \tag{4}$$

Then the equations of motion follow from:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_j}\right) - \frac{\partial \mathcal{L}}{\partial q_j} = \tau_j \tag{5}$$

Using the symbolics toolbox of MATLAB the Lagrangian is defined for a specific system. From this expression the equations of motion can then derived with Eq. (5) through analytical differentiation provided by the toolbox. The equations will be in the form of Eq. (1). If the system is small enough, i.e. if the analytical inverse of the mass matrix does not have too many terms, the dynamics can be expressed in closed form like Eq. (3). For more complex system, the analytical mass matrix is kept and only inverted numerically when used. From the symbolic expressions MATLAB or C++ code can be generated, which can then be used on different platforms.

2.1.1 Numerical Simulators

While such an analytical approach nets an accurate description that can be evaluated quickly, it is limited to systems with a low number of degrees-of-freedom, preferably with no kinematic coupling between those degrees-of-freedom. And using a pipeline that solves for the equations

of motion automatically pushes these limits back only a little bit. Specifically collisions between bodies are difficult to model in this way. Instead we can look at numerical methods to solve dynamic systems, which are not restricted by the dimension of the problem. Numerous physics simulators are available that solve such systems, such as MuJoCo, ODE and Bullet [16]. Such simulators work by composing a dynamics equation for all coordinates and subjecting it to constraints resulting from joints. Collisions can be included as constraints (hard contacts) or as reaction forces (soft contacts).
MuJoCo specifically is a good choice for a simulator when comparing it to alternatives because of its accuracy and speed [16, 17, 18]. It was created with model predictive applications in mind. In MuJoCo models are created in XML files, which provide a hierarchical structure. It can then compute forward and inverse dynamics, as well as step-wise simulation. The simulation is focused on generalized coordinates and states, velocities and forces can be easily set and read, as well as e.g. geometric jacobians and the mass matrix.

2.2 Model Predictive Control

Control techniques typically focus on defining the system inputs of the next discrete time steps based on current control error (feedback) or on current target (feedforward). Instead of working rather short-sighted we can try to optimize our control over a longer stretch of time. In a general sense, optimal control revolves around finding some input signal $u(t)$ in a window $0 \leq t \leq T,\ T \in (0, \infty)$ such that the output is as desired and a cost function (typically related to the system's control signal input) is minimized. Typically, optimal control results in an optimized feedback controller, tuned for a specific use-case. Model predictive control takes it a step further by using a model of the system to tailor a control sequence to the current situation, resulting in a set of feedforward controls that will bring the system to a target, assuming the physical system is identical to the modeled system. The final configuration at $t = T$ is referred to as the horizon. If the horizon is finite it can be at a fixed point in time (approaching horizon) or continuously in the future (fixed or rolling horizon).

The model would have to be perfect and the plant without disturbances for the realized trajectory to be identical to the optimized trajectory, through only the optimized input. In practice this is almost never the case. Because of this, practical implementations of MPC use it online. That is to say the optimization is repeated such a new trajectory is created after the realized one has deviated. This is illustrated in Fig. 4. The factor that limits when the optimization is performed again is often the time it takes to perform the optimization. In practice this MPC rate is often at 10 to 100 Hz [5].

Additionally a feedback controller is typically added to steer the real trajectory to the optimized one (see Fig. 5). This feedback controller is not limited by computational power like the optimizer, so it can be run much more often (typically around 1000 Hz). The addition of the feedback controller makes the controlled plant much more robust to disturbances.

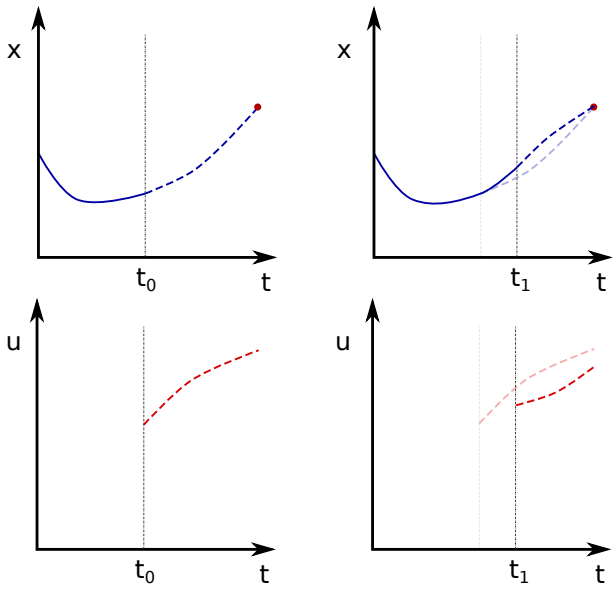

Figure 4: The past state (solid line) and predicted state and input (dashed lines) at two time instances. Because of a disturbance or modelling error the state trajectory evolved differently from the prediction, resulting in a new optimized trajectory.

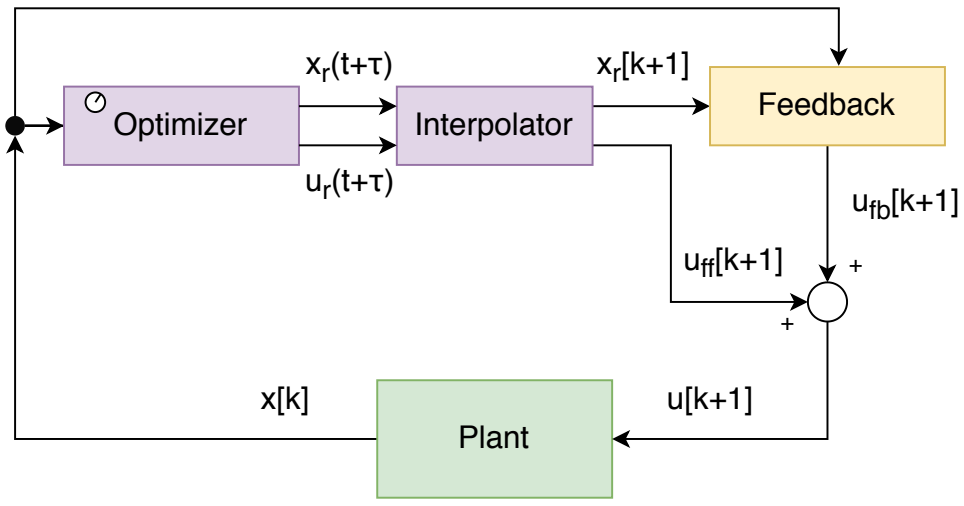

Figure 5: Overview of the online MPC control scheme. The optimizer produces a set of reference torques and states for the future and is updated at a slower rate than the control loop itself. Here $t+\tau$ represents the predicted time, last updated at t. The interpolater picks the reference state and feedforward torque from the reference set.

2.2.1 Optimization Problems

In a general sense, an optimization problem is formulated as:

$$\begin{aligned} \min_{\mathbf{z}} J(\mathbf{z}) \quad & \text{subject to :} \\ f(\mathbf{z}) &= \mathbf{0} \\ g(\mathbf{z}) &\leq \mathbf{0} \\ \mathbf{z}_l \leq & \mathbf{z} \leq \mathbf{z}_u \end{aligned} \tag{6}$$

Here $J(\mathbf{z})$ is the objective function, $\mathbf{z}$ the optimization variable, $f(\mathbf{z})$ an equality constraint, $g(\mathbf{z})$ an inequality constraint and $\mathbf{z}_l$ and $\mathbf{z}_u$ are the variable upper and lower bounds.

The collection of objective, variables, constraints and bounds is referred to a NLP (non-linear problem) in case the constraints are non-linear. There are also linear problems (LP) and quadratic problems (QP). NLPs are the hardest to solve and difficult to find a global optimum for.

A problem with optimizing a non-linear problem is that there is no guarantee a global solution is found. Instead the solver can return a local optimum (see Fig. 6) as it is not able to distinguish it as only a local optimum. Solvers typically follow the gradient of the objective function and return when this gradient is (close to) zero, indicating a minimum or maximum. This is also why the initial guess (also named starting point) of the problem plays an important role. The starting values for the optimization variables can determine which solution the solver will find. Having a starting point close the global minimum will make it much easier to find this optimum. In practice it not always achievable to have such a well-tailored starting point.

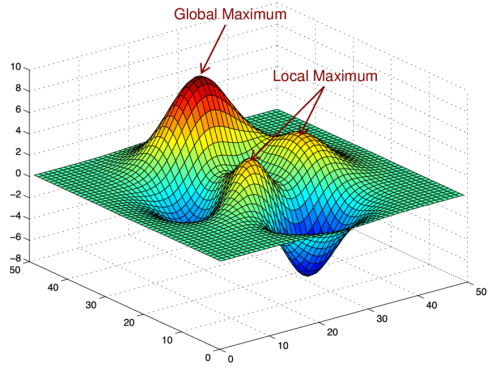

Figure 6: The objective function (vertical axis) plotted against two optimizations variables. An optimizer can find a local optimum instead of the global optimum. Image from [19].

The problem is solved when the optimization variables converge: the constraints are not violated and the objective function gradient is zero. When a problem converges easily, the solver finds a solution through a small number of productive steps. Convergence is not always easy, the objective function for example can be rough such that many steps are needed or the constraints too limiting to reach an optimum. When an optimizations does not converge, either the problem itself is unsolvable or the allotted time was too little. There is no sure way to know the difference.

There is a wide variety of solvers available that solve such optimization problems. Examples of solvers for NLPs are IPOPT [20] (interior-point optimization), SNOPT [21] (sparse non-linear optimizer) and function `fmincon` in MATLAB. Solvers for quadratic programs are for example Xpress [22] and CVXOPT [23] (and the code generator CVXGEN [24]).
All these tools have varying strengths and weaknesses and licenses. Although there is hardly

an ultimate champion among these solver, IPOPT [20] is particularly interesting because of its speed, open-source nature, active development, robustness and flexibility.

2.2.2 MPC Techniques

There is a distinction between two different approaches in MPC: there are the direct and indirect methods [25]. In a direct method the trajectory is first discretized to a finite set of points. For these points the constraints and objectives are then optimized. In other words, the problem is discretized and than optimized. With an indirect method the conditions for optimality need to first be composed analytically. Those are then discretized and solved analytically. So the problem is first optimized and then discretized [6].

Direct methods are typically more suitable to problems involving system dynamics, because they are less dependent on a strict initial guess and more robust, and they are easier to formulate and solve for complex constraint systems than indirect methods. On the other hand, indirect methods typically result in more accurate solutions and are easier to combine with path constraints [6].

There exist multiple methods of problem formulation for MPC. The most prominent ones are put forward.

Direct Collocation As the name implies, direct collocation is a direct method and therefore shapes both the control and state trajectories at the same time. It works by discretizing the optimization window into collocation points. This is shown in Fig. 7. Each state and input signal for each point is then an optimization variable, such that for N collocation points the total number of variables to be optimized is $N \cdot (n(\mathbf{u}) + n(\mathbf{x}))$. The state $\mathbf{x}$ is a combination of independent coordinates and velocities:

$$\mathbf{x} = \begin{pmatrix} \mathbf{q} \\ \mathbf{v} \end{pmatrix} \tag{7}$$

The complete optimization vector then looks like:

$$\mathbf{z}^T = (\mathbf{x}^T[1] \quad \mathbf{x}^T[2] \quad \dots \quad \mathbf{x}^T[N] \quad \mathbf{u}^T[1] \quad \mathbf{u}^T[2] \quad \dots \quad \mathbf{u}^T[N])^T \tag{8}$$

In order to keep the system dynamically correct, constraints are added to correlate accelerations, velocities, positions and torques of adjacent collocation points. A type of interpolation has to be chosen for integration between points. The simplest approach is a first order interpolation, resulting in trapezoidal integration. A more accurate solution could come from a second order interpolation, which is also called the Hermite-Simpson method [6]. Any order polynomial could be used to interpolate between points. The polynomial coefficients are then optimized in addition to or instead of the collocation points. Additional constraints might then be needed to keep the trajectories first order continuous to be physically feasible.

Differential Dynamic Programming Dynamic programming methods do not really belong to either the direct or indirect methods. The latter methods work by finding an optimal trajectory, whereas dynamic programming methods result in an optimal policy, which can then be applied to all points in the state space. The advantage is this policy can be directly applied to the real system. A second advantage is the global optimal will always be found (at least for sufficiently basic problems). A significant disadvantage of dynamic programming is that computing a solution for every point in the state space is expensive. And this computational intensity scales exponentially with the dimensions of the problem [6].

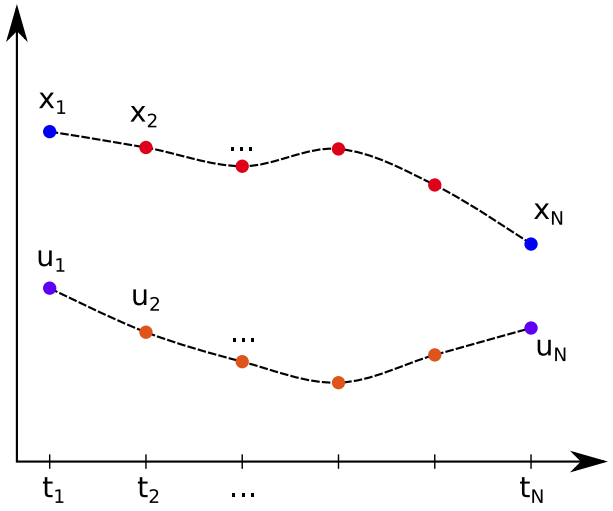

Figure 7: Diagram showing how the trajectory which is to be optimized is discretized into collocation points. The first and Nth points are fixed, all points in between are optimized.

Differential dynamic programming (DDP) is a type of dynamic programming. It relies on the optimality principle: a problem with an optimal sub-structure can be broken up into sub-problems, where each part is solved optimally, resulting in an optimal solution for the original problem.

We consider the dynamics of the system in consecutive timesteps:

$$\mathbf{x}_{i+1} = \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i) \tag{9}$$

And a set of all inputs for the time window:

$$\mathcal{U} = (\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{N-1}) \tag{10}$$

A cost function based on all inputs is constructed:

$$J_0(\mathbf{x}, \mathcal{U}) = \sum_{i=0}^{N-1} l(\mathbf{x}_i, \mathbf{u}_i) + l_f(\mathbf{x}_N) \tag{11}$$

We take a tail section of the control sequence and define the cost-to-go function (the cost to go from a certain point to the end):

$$\mathcal{U}_i = (\mathbf{u}_i, \mathbf{u}_{i+1}, \ldots, \mathbf{u}_{N-1}) \tag{12}$$

$$J_i(\mathbf{x}, \mathrm{U}_i) = \sum_{j=i}^{N-1} l(\mathbf{x}_j, \mathbf{u}_j) + l_f(\mathbf{x}_N) \tag{13}$$

Define the optimal cost function, which is the value of the cost-to-go function for a certain point:

$$V(\mathbf{x}, i) = \min_{\mathcal{U}_i} J_i(\mathbf{x}, \mathcal{U}_i) \tag{14}$$

From the optimality principle:

$$V(\mathbf{x}, i) = \min_{\mathbf{u}} \left[l(\mathbf{x}, \mathbf{u}) + V(\mathbf{x}_{i+1}, i+1) \right] \tag{15}$$

$$= \min_{\mathbf{u}} \left[l(\mathbf{x}, \mathbf{u}) + V(\mathbf{f}(\mathbf{x}_i, \mathbf{u}_i), i+1) \right] \tag{16}$$

We find a second order variation function of the arguments of the min function in V:

$$\begin{aligned} Q(\delta\mathbf{x}, \delta\mathbf{u}) &= l(\mathbf{x} + \delta\mathbf{x}, \mathbf{u} + \delta\mathbf{u}) + V(\mathbf{f}(\mathbf{x}_i + \delta\mathbf{x}_i, \mathbf{u}_i + \delta\mathbf{u}_i), i + 1) \\ &\quad - l(\mathbf{x}, \mathbf{u}) - V(\mathbf{f}(\mathbf{x}_i, \mathbf{u}_i), i + 1) \qquad (17) \\ &\approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix}^\mathsf{T} \begin{bmatrix} 0 & Q_\mathbf{x}^\mathsf{T} & Q_\mathbf{u}^\mathsf{T} \\ Q_\mathbf{x} & Q_{\mathbf{xx}} & Q_{\mathbf{xu}} \\ Q_\mathbf{u} & Q_{\mathbf{ux}} & Q_{\mathbf{uu}} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix} \qquad (18) \end{aligned}$$

This Q function is the discrete analog of the Hamiltonian.

Using Q we can iteratively optimize the problem:

1. Initialize problem with initial guess $\mathbf{x}_0$ and $\mathcal{U}_0$.
2. Perform a forward pass, i.e. a regular forward simulation of the system using $\mathcal{U}$.
3. Perform a backward pass, i.e. estimate the objective function and dynamics for each state and input.
4. Calculate an updated $\mathcal{U}$.
5. Decide on program termination or repeat with another forward pass.

Together with approximations to the Hessian of the dynamics in Eq. (18) with respect to the variations $\delta\mathbf{x}$ and $\delta\mathbf{u}$ this method is called iLQR or iLQG (iterative Linear Quadratic Regulator / Gaussian)

Direct Shooting Methods In direct shooting methods [6] a trajectory is optimized through simulation, from an initial state and controls along the way (shown in Fig. 8). The trajectory itself is not part of the optimization variables. In direct single shooting the entire trajectory in considered at once, whereas direct multiple shooting breaks the trajectory up in segments, making the optimization much more robust. Compared to collocation methods, whose problems are big and sparse, shooting methods build problems that are smaller and more dense.
Path constraints are difficult to implement on shooting methods because the trajectory is not directly a part of the NLP. Instead, the state at any time has to be propagated from the initial state.

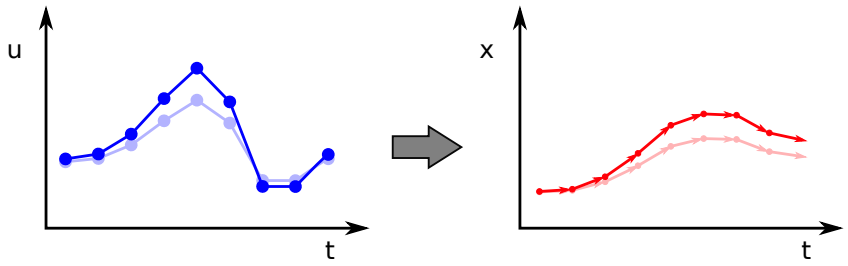

Figure 8: Diagram of an example of two iterations of direct single shooting. Only the input (left, blue nodes) are optimization variables. The resulting state trajectory (right, red arrows) has to be computed from the input.

2.3 Implementation Levels of Locomotive MPC

Model predictive control for locomotion can be applied on different stages and work together with different schemes that e.g. produce foothold locations. A few different levels of implementation are suggested below. See Fig. 9 for a schematic overview.

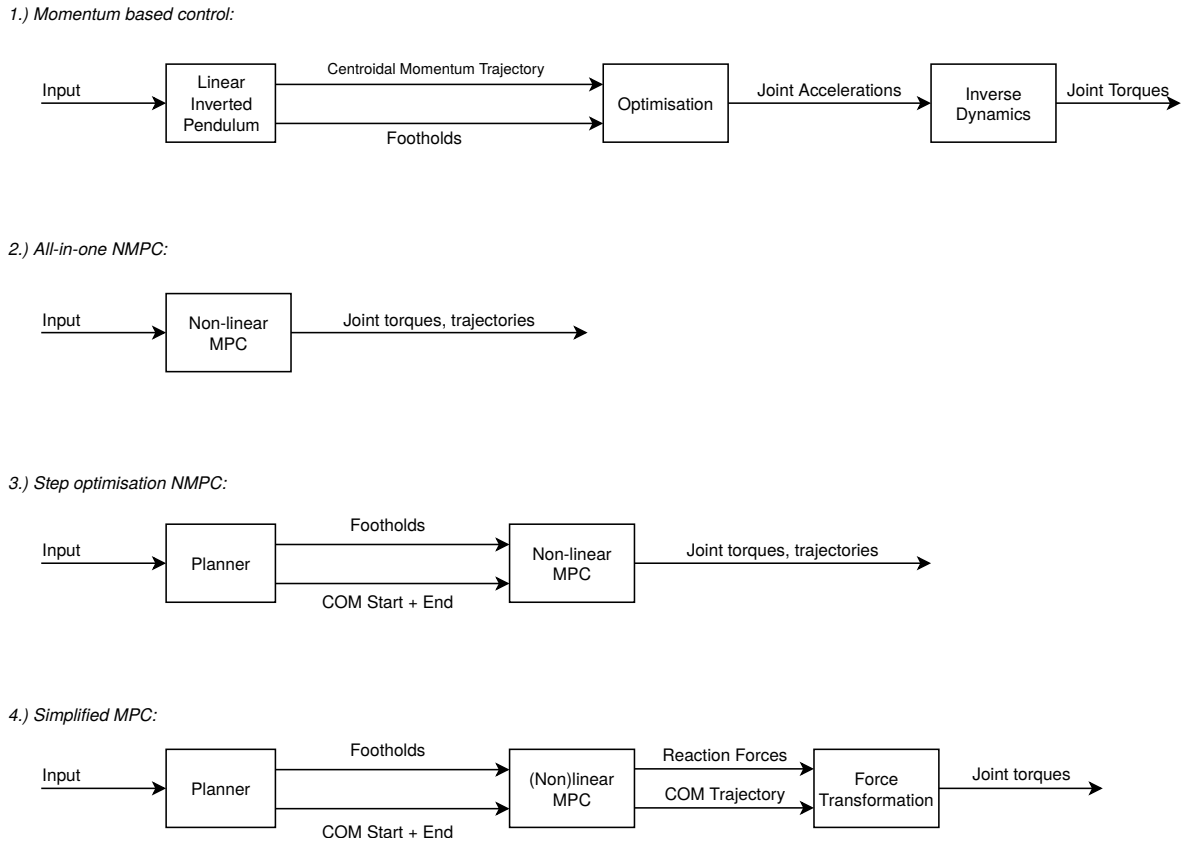

Figure 9: Overview of different levels to MPC-like locomotion control. The main variation is which steps are part of the locomotion optimizer. The most complete solution would be optimizing the entire movement from A to B (typically at least an entire stride), which is also the method that would be the most difficult to implement. A linear inverted pendulum (LIP) is a more basic and conventional model for legged locomotion. 'Input' represents a high-level control signal, such as a joystick.

All-in-one NMPC:

The amount of inputs and constraints is kept at a minimum. The optimizer should figure out where to place the footholds in order to move and keep stability.
Getting a complete NMPC scheme to converge quickly enough is challenging. We are still unsure if this method is practical at all. It is likely that optimization has to be done over an entire stride (i.e. two steps) in order to achieve proper walking.
In theory this is the preferred method, because both trajectory and stability are optimized at the same time (e.g. balance can be kept after a disturbance by changing the next foothold).

Step optimization NMPC:

The footholds are produced by another scheme and the NMPC steps through them. This could be either per step or per stride (two steps). The latter might produce a more optimal and less rigid motion since intermediate states are less restricted. This might be easier to converge than the all-in-one NMPC, though it might also be less resilient in terms of balance. Balance robustness will depend on how strict the foothold constraints are and how intricate the foothold planner is. If the footholds are only described by a range (or maybe even only through the objective function), side-stepping to maintain balance could still occur.

Simplified (L)MPC:

The complete system dynamics come down to:

$$\mathrm{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathrm{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathrm{N}(\mathbf{q}) = \boldsymbol{\tau} \tag{19}$$

Instead we only consider the main body, as a lumped mass with only reaction forces from the ground. Then:

$$m\ddot{\mathbf{r}}(t) = \sum_i \mathbf{f}_i(t) - m\mathbf{g} \tag{20}$$

$$\mathrm{I}\dot{\boldsymbol{\omega}}(t) + \boldsymbol{\omega}(t) \times \mathrm{I}\boldsymbol{\omega}(t) = \sum_i \mathbf{f}_i \times (\mathbf{r}(t) - \mathbf{p}_i(t)) \tag{21}$$

Having the footholds from an external planner, we can optimize the reaction forces from those footholds and the center of mass trajectory (see Fig. 10). Because of rotations in 3D this simple system is still non-linear. However, if the angular velocities remain small the fictitious forces can be neglected ($\boldsymbol{\omega}(t) \times \mathrm{I}\boldsymbol{\omega}(t) \approx 0$). A quadratic program can be solved orders of magnitudes faster than a non-linear equivalent, so this could be advantageous for real-time applications. From the desired reaction forces the joint torques can be computed using kineostatics:

$$\boldsymbol{\tau} = \mathrm{J}^T(\mathbf{q}) \sum_i \mathbf{f}_i \tag{22}$$

Note that finding optimal joint torques which correspond to ground reaction forces is an optimization problem in itself, because the number of actuated joints is typically larger than the combined dimensions of the reaction forces. The pseudo-inverse mentioned above is the easiest solution, though another optimizer could be used to get a more efficient outcome.

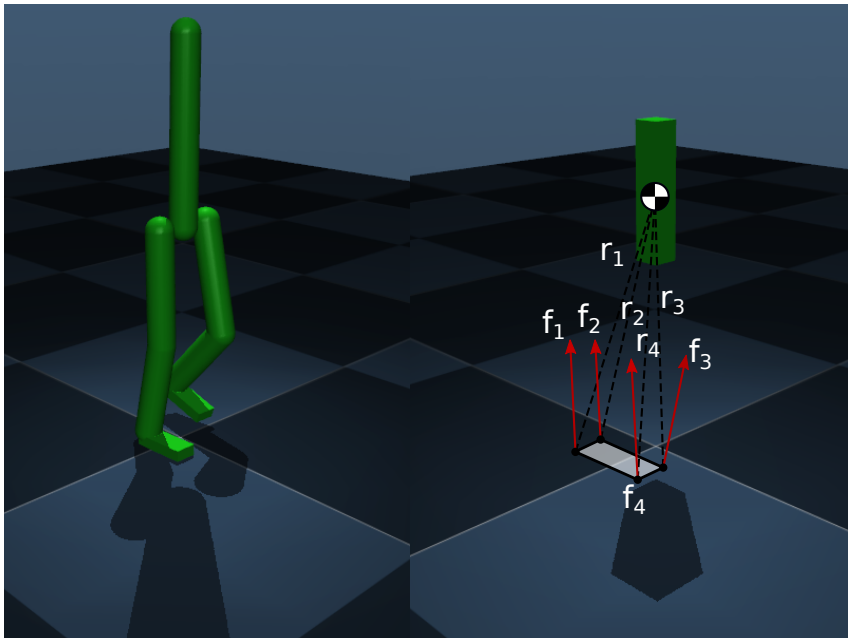

Figure 10: The system is simplified to a single body on which reaction forces from the ground act. These reaction forces span the contact surface.

2.4 MPC Real-Time Implementation

Perfect non-linear model predictive control, which would be control for an entire body over a large number of collocation points for a significant horizon executed at a high control rate, is difficult and currently impossible. Conventional control loops run at a speed in the order of 1 ms (1000 times per second). Solving a difficult NLP in such a short time is not possible on even modern personal computers. Techniques have been used in previous work in attempts to overcome this speed limitation. The most promising techniques will be discussed with their advantages and disadvantages.

Simplification: Avoid System Dynamics

NMPC is computationally intensive to solve because of the non-linear nature. A linear or quadratic optimization problem can typically be solved quickly enough to be used in real-time. The non-linear aspect in NMPC comes from constraints which are not linear. The system dynamics constraints for instance are not linear to all optimization variables for any system in 3D or containing even a single rotation joint. By simply excluding the system kinetics and only considering the kinematics, the problem can be made a quadratic program (QP), making it fairly easy to solve. This is what Hutter et al. do in [7] for the quadruped hybrid ANYmal robot. They use MPC to find a smooth trajectory for the wheels at the tips of the legs, without taking torques into account. As a result they can optimize each wheel trajectory in under 1 ms.

The significant downside of this is the resulting trajectory is optimal only in its own shape. Momentum of all bodies or joint torques cannot be considered, which are arguably the properties that matter most for optimality.

Simplification: Lumped Rigid Body

In [10] MPC is used in combination with whole-body control (WBC) on the quadruped Cheetah robot. For the system dynamics only the base body is taken into account, which is modelled as a single rigid body. Additionally the Newton-Euler equations are simplified such that the EOM in 3D are linear. Instead of optimizing joint torques, the reaction forces at the ground for each leg are optimized (see Fig. 10). The optimization problem is quadratic, they call it convex model predictive control (cMPC) and it can be quickly solved. The online WBC then computes the required joint torques to realize the computed reaction forces. Sangbae et al. advice to put aside the idea of optimizing joint trajectories and instead only focus on the reaction forces. This would work well for a quadruped robot, which is frequently entirely in the air.
Although this cMPC method results in a QP, Sanbgae et al. only run the prediction at 30 Hz. It should be possible to run the optimization more often if needed. Still, their results are impressively dynamic.

Similar work was done before by [26]. Here they optimize the dynamics of only the center of mass, which they call the centroidal dynamics, of a biped robot. And also in [1], where Winkler et al. optimize centroidal dynamics for robots with any number of legs in a impressive software platform, named TOWR.

Optimization: Complete Dynamics

A more head-on approach is done in [12]. Hutter et al. include the entire system dynamics and go through great lengths to tailor a custom optimization sequence (instead of an off-the-shelf optimizer) that can optimize their control torques at a rate up to 190 Hz for a time window of 0.5 seconds.

Hutter et al. made the software platform open-source, it is released as the Control Toolbox.

Offline optimization: Neural Network

Another solution would be to optimize state and control trajectories offline. This would result into a large look-up table of states and torques. This table could then be used for online control instead of running the optimization online. A potentially more effective method could be training a neural network on the generated table and using the trained network for online control. To the best of our knowledge such an intermediate neural network has not been tested on walking robots before.

2.5 Modelling Contact in MPC

Dynamic optimization problems based on a set of continuous equations of motion are reasonably straightforward. The problem description can be defined clearly, gradients are defined continuously and convergence is relatively easy to achieve. This was done earlier by this research group [4, 5] for a sagittal walker, hinged at the stance foot. However, the resulting generated gait has no ground interaction whatsoever, making the double stance phase infinitely short. The end-effectors never switch between stance or swing during the horizon. The result would likely not be directly useful for a real robot, the interaction with the ground through contact is vital in legged locomotion.

To find a more useful solution the ground interaction has to be taken into account. This has two purposes. For one, contact is needed such that all walking phases such as foot landing, double stance and foot push-off can be included. And secondly the floor contacts significantly influence the system dynamics. For example, a high speed impact might be unfavorable because of the resulting torques to stabilize. Such aspects are ignored when no floor contact is included.

Modelling contact with the floor is not trivial, largely because it is inherently discontinuous. There are various approaches, which are discussed below. Figure 11 summarizes the methods.

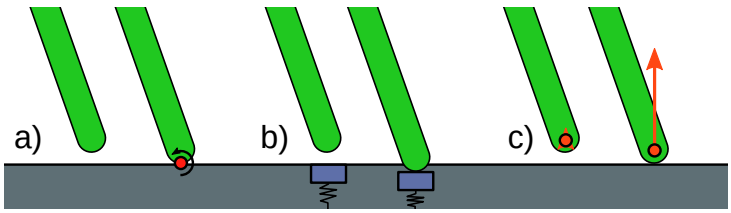

Figure 11: Three different methods of handling contacts: a) Hybrid dynamics (switch to a joint), b) non-linear soft contacts and c) reaction force optimization.

2.5.1 Implicit Contacts

It could be easiest to include the contact dynamics in the equations of motion $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$. This way no specific attention to contact is required. The dynamics will be accurate, no scheduled contact information is needed and contacts will be automatically discovered.
One way to accomplish this is by employing soft contacts, i.e. modelling contact through spring-dampers. Hutter et al. [12] use exponential spring-dampers to keep reaction forces (close to) zero in the air and to a varying non-zero value based on penetration:

$$\boldsymbol{\lambda} = -k \exp\left(\alpha_k \mathbf{p}_z(\mathbf{q})\right) - d \operatorname{sign}\left(\alpha_d \mathbf{p}_z(\mathbf{q})\right) \dot{\mathbf{p}}(\mathbf{q}, \dot{\mathbf{q}}) \tag{23}$$

Downsides of such soft contacts is that they are relatively inaccurate compared to the hard contacts of the real world. If the spring-dampers are made more stiff the accuracy becomes larger, however as a result a higher frequency in simulation (i.e. density of collocation points) will be needed to model it properly. The model will be more robust for softer springs, at the cost of accuracy compared to the real world.

Another problem is the spring-dampers require careful tuning to be non-sticking, i.e. it should not take additional force to pull a limb out of the penetration into the floor. The Hunt-Crossley spring-damper [27] is suitable, it is designed to be non-sticking for contact simulation. Its definition is somewhat complex and therefor the dynamics derivatives will be even more complex.

A method that still seems to have been explored little in combination with collocation techniques so far is using dedicated simulators that can handle rigid body mechanics and contact dynamics. A simulator like MuJoCo [28] could be used to compute the dynamics function during optimization, also making the contacts entirely implicit. Such simulators are based on soft contacts, though through numerical methods the stiffness of contacts can be much greater than an analytical model with soft contact.
A limitation of numerical simulators is the required gradients. Forward dynamics can be easy and fast to compute, but the NLP will inevitably require gradients in order to be solved. So besides the system dynamics itself:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \tag{24}$$

all gradients are needed too:

$$\frac{\partial \dot{\mathbf{x}}}{\partial \mathbf{x}}, \frac{\partial \dot{\mathbf{x}}}{\partial \mathbf{u}} \tag{25}$$

Such gradients will have to be approximated through finite differences, e.g.:

$$\frac{\partial \dot{\mathbf{x}}}{\partial x_i} \approx \frac{f(\mathbf{x} + \boldsymbol{\epsilon}_i, \mathbf{u}) - f(\mathbf{x}, \mathbf{u})}{||\boldsymbol{\epsilon}_i||} \tag{26}$$

Meaning that for every element of $\mathbf{x}$ dynamics have to be computed again, increasing the computation time of the optimization proportional to the size of $\mathbf{x}$.

2.5.2 Hybrid Model

Arguably the most common method is using a hybrid model [29, 30, 31], i.e. a model that switches behavior based on the current mode. The previous works of Opheusden [4] and Paas [5] are arguably also hybrid models, although their optimizations simply stop before the transition to a next phase. Examples of such modes could be double stance, swing or flight (completely free from the ground, relevant for running gaits). The modes are then connected through extra constraints, to make sure the states at the boundaries correspond. Modes typically switch at fixed schedules, the time for each mode is then pre-programmed.

This type of model is not ideal, most noticeably for more complex systems. Moreover, scheduled mode switches limit the extend to which the problem can be optimized. The hybrid model is criticized further in [14] and the suggested alternative is mentioned below.

2.5.3 Contact Optimization

An alternative to the hybrid model is suggested in [14] is optimizing the contact explicitly. Numerical simulators typically solve contacts between rigid bodies using an optimization problem. The result from this optimization are the contact forces. Instead of considering the contact forces as part of the dynamics function (and therefore part of the constraints), it is suggested to include the contact forces in the optimization variables, effectively solving the dynamic contacts together with the rest of the problem. This does increase the number of variables to be solved, though this is acceptable considering the ease with which the dynamics can now be calculated.

Posa et al. [14] describe the forward dynamics of a rigid body with floating base as:

$$\begin{aligned} \text{find} \quad & \ddot{\mathbf{q}}, \boldsymbol{\lambda} && (27) \\ \text{subject to} \quad & \mathrm{M}(\mathbf{q}) + \mathrm{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathrm{N}(\mathbf{q}) = \mathrm{J}_a(\mathbf{q})\mathbf{u} + \mathrm{J}^T(\mathbf{q})\boldsymbol{\lambda} \\ & \phi(\mathbf{q}) = 0 \\ & \boldsymbol{\lambda} \geq \mathbf{0} && (28) \\ & \forall i, \ \phi_i(\mathbf{q})\lambda_i = 0 && (29) \end{aligned}$$

Where $\boldsymbol{\lambda}$ represent the constraint forces, $\phi(\mathbf{q}) \geq 0$ the non-penetration constraint and $\mathrm{J}^T(\mathbf{q})$ the jacobian projecting constraint forces to generalized forces.

2.5.4 Timed Foothold Optimization

A large downside of hybrid dynamics methods is that footholds are typically entirely planned beforehand, i.e. timing of each gait phase is fixed. Ideally the gait itself is optimized as well, instead of having to rely on the suggestion that is given. An improvement to this is suggested by Winkler et al. [1]: each leg has consecutive contact times $T_{i,contact}$ and swing times $T_{i,free}$, which are optimized too. The number of points that react to the ground is discretized to a set of end-effectors. Winkler et al. use a single end-effector per leg, modelling pin-feet. This could be extended further to include for instance heels and toes on a leg.
The optimizer is free to naturally select a gait pattern. In practice the final gait is largely dependent on the initial guess, so the gait can be designed through it while still being optimized. The number of phases for each end-effector is fixed. However, because the duration of each individual phase is variable, there is still large flexibility in the final gait selection.

The gait phases are realized through reaction forces at the end-effectors. These forces are zero during swing and non-zero during stance. The forces are included in the optimization variables, comparable to the contact method mentioned above.

Figure 12 shows how the phase durations result in different gaits. Figure 13 shows the complete non-linear problem description.

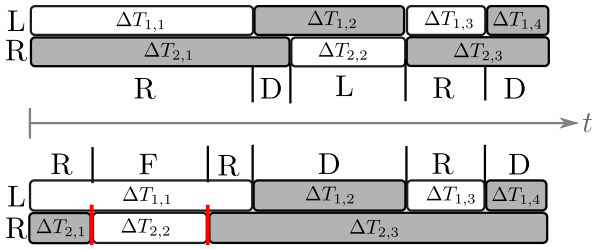

Figure 12: Two different biped gaits (top and bottom), only varied by changing the phase durations of individual legs. [1]

$$
\begin{aligned}
\text{find} \quad & \mathbf{r}(t) \in \mathbb{R}^3 && \text{(CoM linear position)} \\
& \boldsymbol{\theta}(t) \in \mathbb{R}^3 && \text{(base euler angles)} \\
& \text{for every foot } i: \\
& \quad \Delta T_{i,1} \ldots, \Delta T_{i,2n_{s,i}} \in \mathbb{R} && \text{(phase durations)} \\
& \quad \mathbf{p}_i(t, \Delta T_{i,1}, \ldots) \in \mathbb{R}^3 && \text{(foot position)} \\
& \quad \mathbf{f}_i(t, \Delta T_{i,1}, \ldots) \in \mathbb{R}^3 && \text{(force at foot)} \\
\text{s.t.} \quad & [\mathbf{r}, \boldsymbol{\theta}](t{=}0) = [\mathbf{r}_0, \boldsymbol{\theta}_0] && \text{(initial state)} \\
& \mathbf{r}(t{=}T) = \mathbf{r}_g && \text{(desired goal)} \\
& [\ddot{\mathbf{r}}, \dot{\boldsymbol{\omega}}]^T = \mathbf{f}_d(\mathbf{r}, \mathbf{p}_1, \ldots, \mathbf{f}_1, \ldots) && \text{(dynamic model)} \\
& \text{for every foot } i: \\
& \quad \mathbf{p}_i(t) \in \mathcal{R}_i(\mathbf{r}, \boldsymbol{\theta}), && \text{(kinematic model)} \\
& \quad \text{if foot } i \text{ in contact}: \\
& \quad\quad \dot{\mathbf{p}}_i(t \in \mathcal{C}_i) = \mathbf{0} && \text{(no slip)} \\
& \quad\quad p_i^z(t \in \mathcal{C}_i) = h_{terrain}(\mathbf{p}_i^{xy}) && \text{(terrain height)} \\
& \quad\quad \mathbf{f}_i(t \in \mathcal{C}_i) \cdot \mathbf{n}(\mathbf{p}_i^{xy}) \geq 0 && \text{(pushing force)} \\
& \quad\quad \mathbf{f}_i(t \in \mathcal{C}_i) \in \mathcal{F}(\mu, \mathbf{n}, \mathbf{p}_i^{xy}) && \text{(friction cone)} \\
& \quad \text{if foot } i \text{ in air}: \\
& \quad\quad \mathbf{f}_i(t \notin \mathcal{C}_i) = \mathbf{0} && \text{(no force in air)} \\
& \quad \textstyle\sum_{j=1}^{2n_{s,i}} \Delta T_{i,j} = T && \text{(total duration)}
\end{aligned}
$$

Figure 13: The complete problem description of time-parameterized, multi-legged locomotion. [1]

3 - Design

In this chapter the design of the MPC framework is discussed. This is split into different sections, each addressing a specific aspect and the decisions involved in that aspect. These aspects are contacts, system dynamics and NLP formulation. Additionally some light is shed on how the MPC is used in an online fashion and how the software implementation was done.

The foundation was formed by the works by Opheusden [4] and Paas [5], whereas this design builds upon this.

3.1 Contacts

3.1.1 Implicit Contact

The analytical dynamics model of the sagittal five link biped from Kelly [6] (see Fig. 3) is hinged to the ground at the stance foot. By removing this hinge and instead connecting both feet to non-linear spring-dampers (Eq. (23)) soft contact could be simulated, as was put forward in Sec. 2.5.1 *Implicit Contacts*. Because of the analytical nature of the model the needed gradients could be defined exactly, all extracted from MATLAB as outlined in Sec. 2.1 *Dynamic Modelling*. With no need to pre-define gait information the optimization might discover a gait by itself, which is why this method of implicit contact is desirable.
However, no sensible gait can be generated when the model is extended with spring-damper contacts. Only when the springs are made weak the optimization can converge, though the result is an unnatural bouncing from the initial to the final configuration.

As an alternative the analytical model was replaced by a model in the MuJoCo physics simulator. The same five link biped in sagittal plane was recreated as a MuJoCo model. When the stance foot is hinged to the world and the swing foot left free to interact with the ground, good single-step trajectories can be produced. The optimization is quick and robust to the initial guess.
When both feet are made free, again no sensible gait can be produced. No optimizations would converge and the unconverged results look like unnatural and erratic jumping that is not close to physically correct and could not be used on a real system. Starting with a reasonable initial guess does not improve the results. Figure 14 shows the objective function and constraints violation over the course of the iterations. There seems to be no steady decline in constraint violation, so it is unlikely it is a matter of patience. To put this in more perspective, the optimization of the biped when hinged at the stance foot converged typically within 20 iterations.

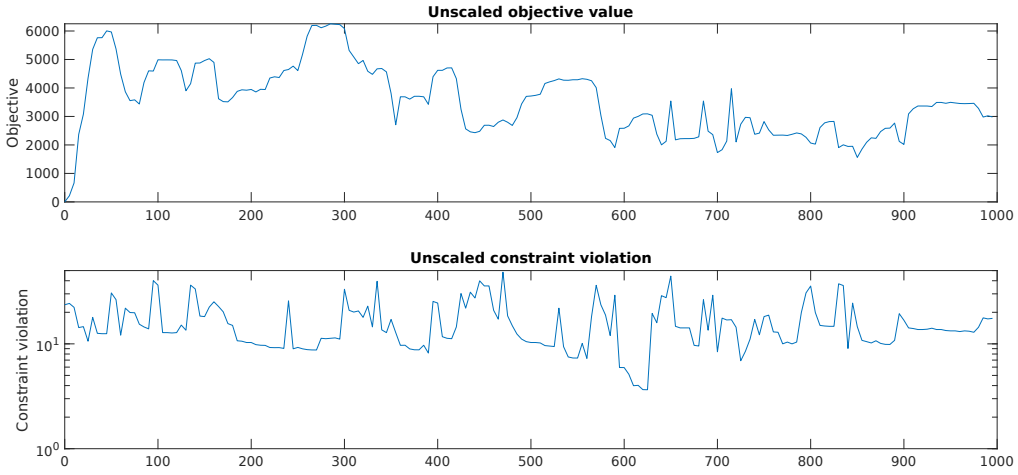

Figure 14: Objective function and constraint violation over iterations during optimization of biped in MuJoCo with implicit contacts. An unscaled constraint violation of about 10^{-5} is considered acceptable.

From these findings it seems that implicit contact through soft-contacts is simply not feasible for optimization of system trajectories. Moreover, the design requires not just feasibility but also robustness in the optimization and flexibility to the initial guess (Sec. 1.2 *Requirements and Objectives*).

A hypothesis to why this could be is outlined in Fig. 15. The optimizer works by following gradients, it has no understanding of the situation. For example, from the gradient of the dynamics function it can 'deduce' that increasing a specific torque will steer a state out of constraint violation. The ground reaction forces have a gradient that is flat when not in contact and become steep when penetration starts. For hard contacts (and the real world) this gradient would be almost infinitely steep. This means that before a limb is close to contact the optimizer is not 'aware' of it. When contact is made the reaction force is sensitive and its gradient non-linear. Thus making it difficult to find a constant penetration depth and rather making the limbs bounce on the surface.

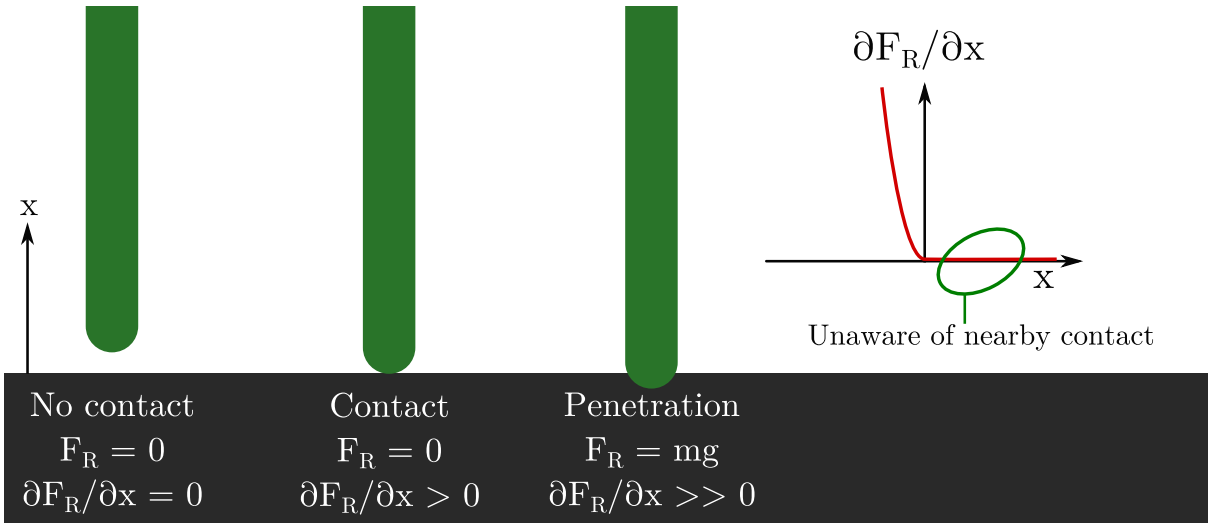

Figure 15: Diagram showing how the ground reaction force (F_R) and its gradient change when contact is being made in a soft-contact simulation. The optimizer looks only at gradients and from the gradient alone a contact cannot be foreseen in time.

3.1.2 Explicit Contact

As the framework concepts based on implicit contacts do not seem feasible, a logical next concept involved explicit contacts. The method of timed footholds of Winkler [1] forms the basis for this, as explained in Sec. 2.5.4 *Timed Foothold Optimization*.
A fork of Winkler's TOWR framework was made, using whole-body dynamics while still relying on explicitly defined contact phases. This proved to be effective, trajectories for arbitrary robots could be produced even with the most basic initial guess. The solver can comfortably deal with the constraints created by the contact, solving them through reaction forces and keeping the system dynamics valid.

There is one major difference between the phase-based optimization in TOWR and this framework: the gait phases are not part of the optimization variables and therefore fixed to the predefined phases. This is a consequence from the universal nodes, which is explained in the next section.

3.2 System Dynamics and Kinematics

3.2.1 MuJoCo

Analytical dynamics produced by MATLAB are a relatively easy solution to modelling dynamics. However it quickly showed it is not practical for realistic systems with more degrees of freedom. Instead it was decided to use a numerical simulator to compute forward dynamics. MuJoCo [28] was chosen as it designed for purposes such as these, as outlined in Sec. 2.1 *Dynamic Modelling*.

Besides dynamics MuJoCo can also provide the solver with kinematic constraints. By naming the end-effector bodies in the XML model file their positions can be found as a function of the generalized coordinates (see Fig. 16). Geometric jacobians for each body are already computed internally that can be used for the constraints jacobian.

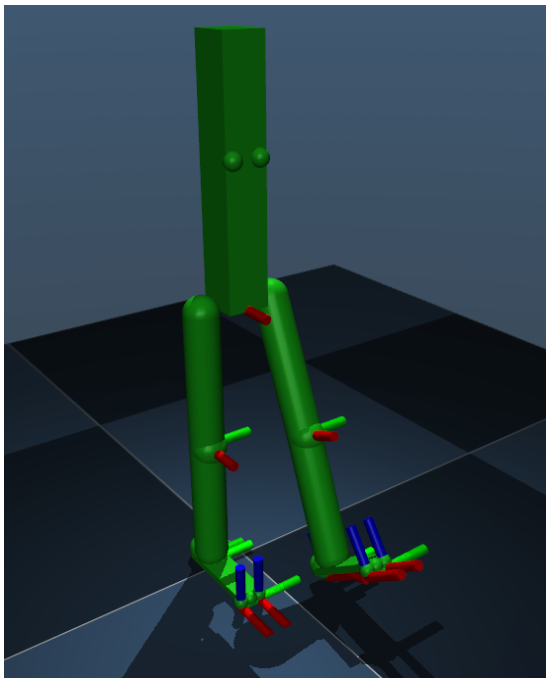

Figure 16: Rendering of a MuJoCo model for a 3d biped walker, showing the body frames. Body frames can be marked as end-effectors by the optimization framework. In this example the heels and inner and outer toes are selected as end-effectors.

A large advantage of using MuJoCo is that models are easily modified and replaced, since almost nothing of the robotic system needs to be embedded in the code. Adding a new robot to produce gaits for is done with the following procedure:

1. Making a new XML model file, with a name for each end-effector body.
2. Pointing the optimizer to this file and listing the names of the end-effector bodies.
3. Describing basic joint positions which will make up the initial guess.
4. Optionally adding joint bounds to limit the solutions.

Although MuJoCo is a suitable platform, it could easily be replaced. The code of the framework is written in an object oriented fashion such the abstract robot class can be extended by any other class relying on a different platform. Such a platform must be able to compute forward dynamics without integration and should include direct state manipulation and forward kinematics.

3.2.2 Node Interpolation

Opheusden [4] and Paas [5] used direct collocation with universal nodes fixed in time, and between the nodes linear interpolation was used. Universal nodes means all variables are defined with a fixed time interval between them, like frames of an animation. Winkler's TOWR framework [1] works with cubic splines instead, where the start and end of each spline are restricted to positions and their derivatives. And the nodes connecting these splines are not all universal, nor are they all fixed in time. This is illustrated in Fig. 17. This works well with the timed phases approach. In TOWR each phase consists of three cubic splines, where start and end node of these phases are timed and can be shifted by the optimizer. Because the nodes are shifted, the full continuous range of time is available. The nodes for variables specific to an end-effector are independently timed to other end-effectors. The nodes for base position and orientation are fixed in time, independent on gait phases.

When collocation nodes are universal, constraints can be applied to a set of nodes that completely define the system at that frame, for instance to keep the dynamics correct. When the nodes are variable in time, constraints have to be applied to the interpolation between them (see Fig. 18) because the nodes do not align in time. This is done in the TOWR framework, at fixed intervals the dynamics and kinematics constraints are applied. Velocities are determined through the derivatives of the splines. The dynamics are constrained by matching the second derivatives of the base splines the applied forces.

When the same principle of phase-based nodes is extended with another spline representing joint torques and the centroidal dynamics constraint replaced with a whole-body constraint, the solver can no longer find a solution that satisfied all constraints. This is likely because of the irregularity in the constraints relative to nodes. Figure 18 shows this: the end-effector position spline has three nodes between two constraints on the far left, whereas there is only one node between two constraints right next it. In order words, parts of the problem are over-constrained while other parts are under-constrained. For almost linear centroidal dynamics, as applied in TOWR, this is not a problem because a linear relation between functions can always be satisfied when those functions are sets of cubic splines. However, a highly non-linear relation like whole-body dynamics cannot always be satisfied by polynomial functions.

Hence it is necessary to use universal nodes, constrained by integrated dynamics (Fig. 19 illustrates this). This proved successful, universal nodes in combination with trapezoidal dynamics integration result in converging optimizations.

The downside of universal nodes compared to phase-based nodes is that the order of gait phases cannot be changed. The reason for this is that the bounds of the inequality constraints

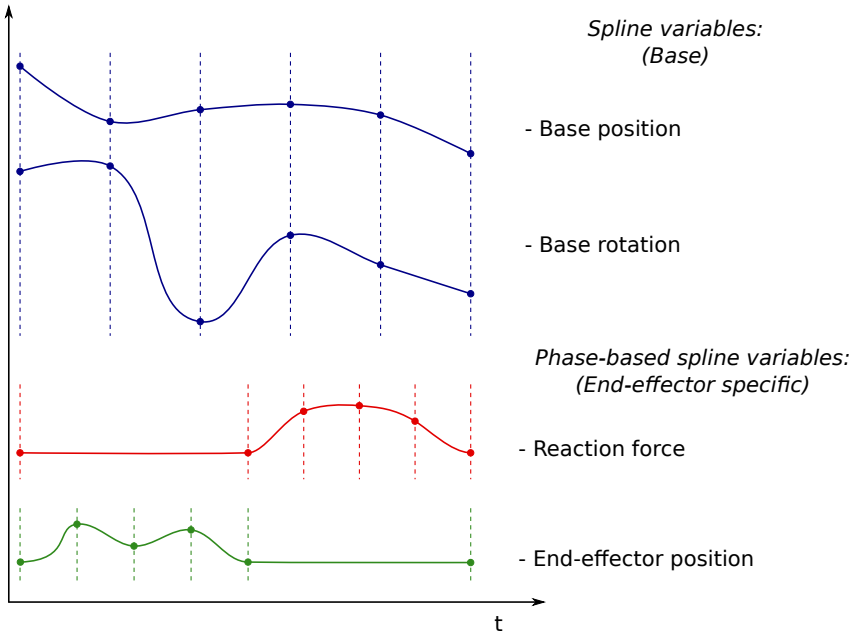

Figure 17: Two different types of spline variables used in the phase-based NLP. For an abstract visualization only a single dimension is drawn per variables and the image is not to a physical scale. The base motion (linear and angular) and the new torque variables are splines with nodes at constant intervals. The variables specific to an end-effectors (reaction forces, end-effector positions) are splines where the node spacing is not constant, in order to realize the variable gait phases. Here the values are either constant or a spline where the start and end time is defined by the phase duration. Inside a phase the nodes are evenly spaced.

must be static and cannot depend on optimization variables. E.g. the limits of an end-effector constraint cannot be toggled from $(0, 0)$ for a contact phase to $(0, \infty)$ for a swing phase. It is still possible to alter the time of nodes and therefore the phases, although these are for all phases at that time. Nonetheless the universal nodes were implemented in the framework to achieve functional dynamics constraints.

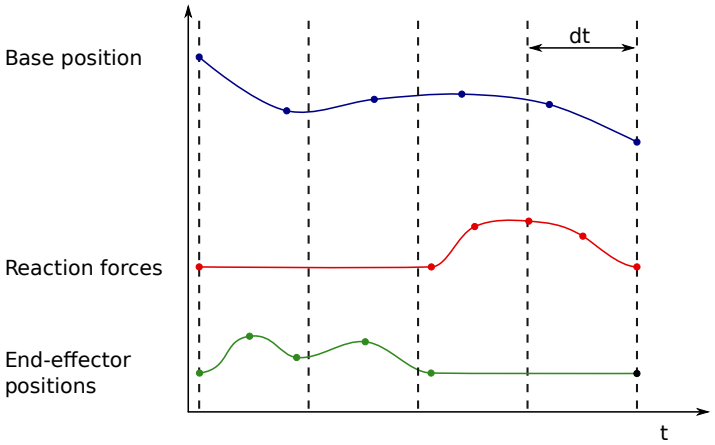

Figure 18: Constraints (dashed lines) are applied at fixed intervals, not directly on the nodes but on the spline interpolation between them.

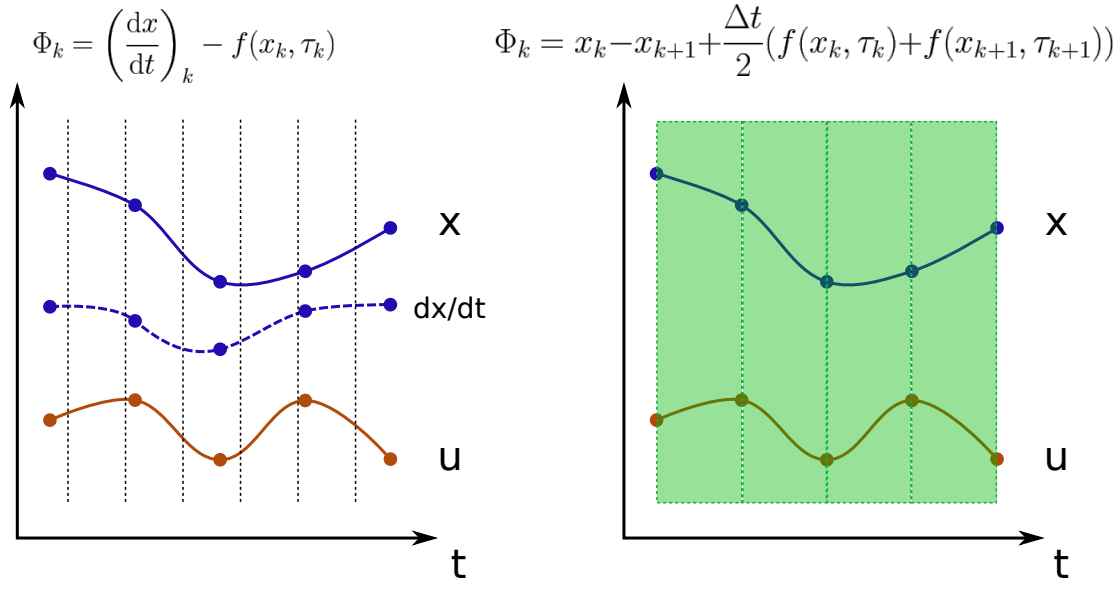

Figure 19: Dynamics constraints can be applied at fixed times (left) for node splines. The acceleration at time instance can be differentiated from the position splines. Alternatively the constraint can be applied by comparing adjacent nodes (right), applying trapezoidal integration of the acceleration. In the latter case no differentiation is needed.

3.3 Non-Linear Problem Formulation

Figure 20 gives an abstract but complete overview of the non-linear problem in the framework. The next sections will elaborate more on the choices made.

Optimize:

- $J = \sum \boldsymbol{\tau}^2$

For:

- $\mathbf{q}(t) \in \mathbb{R}^{n_q}$ Joint coordinates
- $\mathbf{v}(t) \in \mathbb{R}^{n_v}$ Joint velocities
- $\boldsymbol{\tau}(t) \in \mathbb{R}^{n_u}$ Joint torques

For every foot i:

- $\mathbf{F}_i(t, \Delta T_{i,1}, ...) \in \mathbb{R}^3$ Reaction forces

With the constraints:

- $\dot{\mathbf{v}} - f(\mathbf{q}, \mathbf{v}, \boldsymbol{\tau}, \mathbf{F}) = 0$ System dynamics
- $\dot{\mathbf{q}} - \frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} = 0$ System states

For every foot i:

During stance:

- $p_{i,z}(\mathbf{q}) = 0,\ \mathrm{J}_{ee,i}(\mathbf{q})\mathbf{v} = 0$ End-effector on the terrain
- $\sqrt{{F_{i,t1}}^2 + {F_{i,t2}}^2} \leq \mu F_{i,n}$ Limiting to friction pyramid

During swing:

- $p_{i,z}(\mathbf{q}) > 0$ End-effector above the terrain
- $F_i = 0$ No reaction forces

Figure 20: A simplified representation of the non-linear program of the new framework. The phase durations are leading variables but are not optimized over.

3.3.1 Variables

The biped walker optimizations from previous work contain the states (joint positions and joint velocities) and torques as optimization variables. The TOWR platform uses the base position and orientation, end-effector positions and reaction forces. The velocities of the base are not made explicit, instead these are determined from the derivative of the position splines.
With the addition of whole-body dynamics, the joint torques will have to be included as optimization variables. There is however still freedom in how the state is described.
Because the end-effector positions will be needed in the constraints, due to the explicit contact method, it is tempting to include the end-effector positions in the optimization vector, like in the formulation in TOWR. For a simple model the end-effectors can provide an exact coordinate system, see for example Fig. 21. For more complex models a preferable inverse kinematics solution would needed to be picked (e.g. a knee swivel angle) to make for an exact representation. Another downside is the dynamics description become rather difficult (see appendix C).

Including only the joint positions and velocities would make the dynamics description easier. Geometric jacobians would then be needed to constrain the end-effector positions.
A third option is then to include both joint and end-effector states. It would make both the

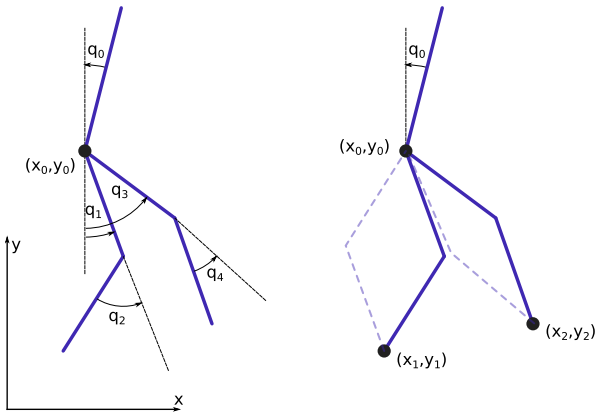

Figure 21: Two examples of exact coordinate sets: joint space (left) and end-effector positions (right). The number of independent variables should be 7, regardless of which set is chosen.

dynamics and terrain constraints easiest, only it would require another set of constraints to make sure the end-effectors match with the joint positions.
It was decided to only include joint states. This limits the size of the optimization vector and the number of constraints. More over, of all notations the one involving only joint states converges most easily.

So the optimization variables will consist on the joint positions $\mathbf{q}$, joint velocities $\mathbf{v}$, joint torques $\boldsymbol{\tau}$ and reaction forces $\mathbf{F}$. Each of these forms a set of variables, which are defined as a set by the help of the IFOPT interface. Each variable set has the variables for that type for all collocation points. We will denote this with the bar symbol:

$$\mathbf{q}[k] = \begin{pmatrix} q_1[k] & q_2[k] & ... \end{pmatrix}^T \tag{30}$$

$$\bar{\mathbf{q}} = \begin{pmatrix} \mathbf{q}[1]^T & \mathbf{q}[2]^T & ... \end{pmatrix}^T \tag{31}$$

$$\tag{32}$$

The same goes for $\bar{\mathbf{v}}$, $\bar{\boldsymbol{\tau}}$ and $\bar{\mathbf{F}}$. Then

$$\mathbf{z}^T = \begin{pmatrix} \bar{\mathbf{q}}^T & \bar{\mathbf{v}}^T & \bar{\boldsymbol{\tau}}^T & \bar{\mathbf{F}}^T \end{pmatrix} \tag{33}$$

3.3.2 Costs

IFOPT makes it easy to define multiple costs that will be added together. The gradient of this can then be defined per cost and per variable set.
The main cost is the torque squared:

$$J(\mathbf{z}) = \sum_{k=0}^{N} \sum_{i=0}^{n(\boldsymbol{\tau})} {\tau_i}^2[k] \tag{34}$$

Joint torque is chosen for minimization because it is the best representation of an efficient gait. The sum is squared, as is common in optimization, such that the objective function itself and its derivative are smooth, a global minimum exists and such that no distinction is made between negative and positive values.

A secondary cost that was experimented with is the angular velocity of the main body:

$$J(\mathbf{z}) = \sum_{k=0}^{N} \sum_{i=3}^{5} {v_i}^2[k] \tag{35}$$

Where v_3 to v_5 is the angular velocity of the base. The addition of this cost limits twisting in the upper body.

3.3.3 Constraints

See appendix D for complete definitions of the constraints and jacobians.

A number of constraints were needed or helpful to produce the gaits. The IFOPT allows us to define constraints in smaller sets. And instead of needing to define the jacobian for these constraints to the entire optimization vector, IFOPT lets us define jacobians per constraint, per variable set:

$$\mathbf{g}^T = \begin{pmatrix} \mathbf{\Phi}_1{}^T & \mathbf{\Phi}_2{}^T & \cdots \end{pmatrix} \tag{36}$$

$$\frac{\partial \mathbf{g}}{\partial \mathbf{z}} = \begin{bmatrix} \frac{\partial \mathbf{\Phi}_1{}^T}{\partial \bar{\mathbf{q}}} & \frac{\partial \mathbf{\Phi}_1{}^T}{\partial \bar{\mathbf{v}}} & \cdots \\ \frac{\partial \mathbf{\Phi}_2{}^T}{\partial \bar{\mathbf{q}}} & \frac{\partial \mathbf{\Phi}_2{}^T}{\partial \bar{\mathbf{v}}} & \cdots \\ \vdots & & \ddots \end{bmatrix} \tag{37}$$

These constraints $\mathbf{\Phi}$ all apply to each collocation point:

$$\mathbf{\Phi}_1 = \begin{pmatrix} \mathbf{\Phi}_1[0] \\ \mathbf{\Phi}_1[1] \\ \vdots \\ \mathbf{\Phi}_1[N-1] \end{pmatrix} \tag{38}$$

The constraint jacobians need to be defined to entire variable sets. To make these definitions easier we introduce node jacobians:

$$\frac{\partial \mathbf{\Phi}[k]}{\partial \bar{\mathbf{q}}} = \frac{\partial \mathbf{\Phi}[k]}{\partial \mathbf{q}[k]} \frac{\partial \mathbf{q}[k]}{\partial \bar{\mathbf{q}}} \tag{39}$$

$$\frac{\partial \mathbf{q}[k]}{\partial \bar{\mathbf{q}}} = \begin{bmatrix} 0 & \dots & 0 & I_{n\times n} & 0 & \dots & 0 \end{bmatrix} \tag{40}$$

$$n = \mathrm{n}(\mathbf{q}[k]) \tag{41}$$

So we only need to define the jacobians to a general point k:

$$\frac{\partial \Phi[k]}{\partial \mathbf{q}[k]}, \ \frac{\partial \Phi[k]}{\partial \mathbf{u}[k]}, \ \frac{\partial \Phi[k]}{\partial \mathbf{F}[k]}, \ \text{etc.} \tag{42}$$

For brevity we will omit the notation $[k]$ further on when possible.

Integration constraints are added to constrain that joint velocities add up to joint positions. If states $\mathbf{x}$ were used as optimization variables this would be implicit in a dynamics constraint: $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$. However, to make integration with a simulator easier the positions and velocities were separated. Therefore also the dynamics constraints and integration constraints are independent.
The constraint is:

$$\mathbf{\Phi}[k] = \mathbf{q}[k] - \mathbf{q}[k+1] + \frac{\Delta t}{2}(\dot{\mathbf{q}}[k] + \dot{\mathbf{q}}[k+1]) \tag{43}$$

Note that the joint rates $\dot{\mathbf{q}}$ are integrated. These can be determined from the joint velocities:

$$\mathbf{\Phi}[k] = \mathbf{q}[k] - \mathbf{q}[k+1] + \frac{\Delta t}{2}\left(\mathrm{B}_k \mathbf{v}[k] + \mathrm{B}_{k+1}\mathbf{v}[k+1]\right) \tag{44}$$

Here $\mathrm{B}(\mathbf{q})$ relates the joint velocities to joint rates (see appendix A.2 *Quaternions in joint position*):

$$\dot{\mathbf{q}} = \mathrm{B}(\mathbf{q})\mathbf{v} \tag{45}$$

Note that last collocation point k is kept free. The bounds of this constraint set are zero.

Dynamics constraints keep the velocity, torques and forces physically correct. We define it as:

$$\mathbf{\Phi}[k] = \mathbf{v}[k] - \mathbf{v}[k+1] + \frac{\Delta t}{2}(\dot{\mathbf{v}}[k] + \dot{\mathbf{v}}[k+1]) \tag{46}$$

$\dot{v}$ is not an explicit variable, so it is computed by the dynamics function:

$$\mathbf{\Phi}[k] = \mathbf{v}[k] - \mathbf{v}[k+1] + \frac{\Delta t}{2}(f_k + f_{k+1}) \tag{47}$$

Where

$$f_k = f(\mathbf{q}[k], \mathbf{v}[k], \boldsymbol{\tau}[k], \mathbf{F}[k]) \tag{48}$$

which is determined numerically by MuJoCo in this implementation. It could just as well be defined analytically or by a different simulation tool.

Note that last collocation point k is kept free. The bounds of this constraint set are zero.

Quaternions constraints make sure the norm of the quaternions embedded in the joint positions are kept normalized:

$$\mathbf{\Phi}[k] = \begin{pmatrix} ||\left(\mathbf{q}_i[k] \quad \mathbf{q}_{i+1}[k] \quad \mathbf{q}_{i+2}[k] \quad \mathbf{q}_{i+3}[k]\right)|| - 1 \\ \vdots \end{pmatrix} \tag{49}$$

Here i is the index corresponding to the first component of a quaternion.
Each quaternion present in the model results in a row in the constraint for each collocation point. Note that most conventional models will only include a single quaternion for the free base.

Terrain constraints are needed to keep the end-effectors on the surface of the terrain without slip:

$$\Phi[k] = \begin{pmatrix} z_{ee}[k] - z_{terrain}[k] \\ x_{ee}[k] - x_{ee}[k-1] \\ y_{ee}[k] - y_{ee}[k-1] \end{pmatrix} \tag{50}$$

The first row keeps the vertical distance between end-effector and terrain zero during stance. The second and third rows define the difference in horizontal location between adjacent collocation points and also should be zero during stance, with the exception of the first point of the stance phase (then $k-1$ is still part of a swing phase).
Instead of these differences in position the horizontal velocities could be constraint to zero. This would be accomplished through multiplication of the end-effector jacobians and the joint velocities. The constraint jacobian would then involve $\frac{\partial J_{ee}(\mathbf{q})\mathbf{v}}{\partial \mathbf{v}}$, which contains the kinematic Hessian and is not easily retrieved from MuJoCo.

Force constraints are used to keep the reaction forces zero during swing phases and limit the forces to a friction cone during stance.
The friction cone is simplified to a pyramid shape to make the equations easier (Fig. 22). This shape corresponds to three limits:

$$0 \leq \mathbf{F} \cdot \mathbf{n} \leq F_{n,max} \tag{51}$$
$$|\mathbf{F} \cdot \mathbf{t}_1| \leq \mu(\mathbf{F} \cdot \mathbf{n}) \tag{52}$$
$$|\mathbf{F} \cdot \mathbf{t}_2| \leq \mu(\mathbf{F} \cdot \mathbf{n}) \tag{53}$$

This means that the reaction force must be on the positive side of the surface and limited to some maximum and each tangential projection of the reaction force must be limited by a friction coefficient μ.

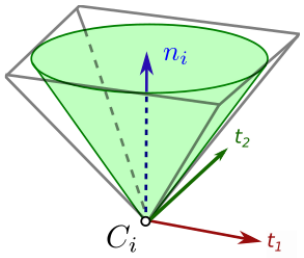

Figure 22: Friction cone and simplified friction pyramid for a contact force. n_i is the surface normal and t_1 and t_2 the surface tangents. Image is from [32] (modified).

This results in 5 constraints in our NLP, because the absolute value operation should be avoided due to corresponding discontinuity in the gradient. These are [1]:

$$\Phi = \begin{pmatrix} \mathbf{F} \cdot \mathbf{n} \\ \mathbf{F} \cdot (\mathbf{t}_1 - \mu\mathbf{n}) \\ \mathbf{F} \cdot (\mathbf{t}_1 + \mu\mathbf{n}) \\ \mathbf{F} \cdot (\mathbf{t}_2 - \mu\mathbf{n}) \\ \mathbf{F} \cdot (\mathbf{t}_2 + \mu\mathbf{n}) \end{pmatrix} \tag{54}$$

These constraints must be zero during flight: $\mathbf{\Phi} = 0$. During stance the inequality constraint is:

$$\begin{pmatrix} 0 \\ -\infty \\ 0 \\ -\infty \\ 0 \end{pmatrix} \leq \mathbf{\Phi} \leq \begin{pmatrix} F_{n,max} \\ 0 \\ \infty \\ 0 \\ \infty \end{pmatrix} \tag{55}$$

Note that these have to be applied for each end-effector individually.

3.4 Online MPC Loop

The purpose the locomotive trajectory generation is online MPC on a real system, hence the designed optimization framework should be testing in an online MPC fashion.
Figure 5 shows an overview of how such an MPC scheme looks like. The optimizer finds a trajectory until the horizon, which is applied as feedforward to the system. A feedback controller is used with the optimizer output as reference. The feedback loop can run at a high rate (order of 1000 Hz), whereas the optimizer takes much more computation time and will therefor update at a slower rate (1 - 100 Hz). The feedback controller and the fact that the optimizer updates can

make the system robust against disturbances, while keeping the optimal nature of the control.

Because a legged robot is not fixed the ground, it is underactuated:

$$\mathrm{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathrm{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathrm{N}(\mathbf{q}) = \mathrm{S}\mathbf{u} + \mathrm{J}_{\boldsymbol{\lambda}}(\mathbf{q})\boldsymbol{\lambda} \tag{56}$$

$$\mathrm{n}(\dot{\mathbf{q}}) > \mathrm{n}(\mathbf{u}) \tag{57}$$

Here S is a selection matrix mapping the actuator torques to system states and $\mathrm{J}_{\boldsymbol{\lambda}}(\mathbf{q})\boldsymbol{\lambda}$ the reaction forces on the end-effectors. All actuators work on joints which are included in the generalized coordinates, hence S consists of rows with only a single 1, the rest being zero.
For the feedback controller torques $\mathbf{u}$ have to be found to reduce an error in the states: $\mathbf{q}_{ref} - \mathbf{q}$. Because the system is underactuated there is no solution that will make the error go to zero for all time instances. It is chosen to simply only include the actuated generalized coordinates in the feedback controller:

$$\mathbf{u}(\mathbf{q}_{u,error}) = \mathbf{u}(\mathrm{S}^T\mathbf{q}_{error}) \tag{58}$$

For a PD controller, where the derivative action is based on the state velocity:

$$\mathbf{u} = \mathrm{S}^T\left(K_p(\mathbf{q}_r - \mathbf{q}) + K_d(\dot{\mathbf{q}}_r - \dot{\mathbf{q}})\right) \tag{59}$$

3.5 Software Implementation

The framework is written in C++, chosen because of its speed and because both IPOPT and MuJoCo are available in C++. The implementation is very similar of the TOWR platform made by Winkler [1]. Like TOWR, IFOPT is used as an interface to the IPOPT solver. IFOPT greatly eases the creation of a non-linear problem by combining variable sets, cost sets and constraint sets, each as individual objects. All of these are classes that extend an application base class, which in turn extends an general IFOPT base class. Appendix E shows UML diagrams showing the structure of parts of the code.

The program starts with a helper class to compose the NLP (`NLPFormulation`). This class creates the variable, cost and constraint sets. Note that all these objects are created as shared pointers such they can be easily shared. The pointers of the variable sets are grouped together in a struct, `NodesHolder`. By passing this struct to a constraint it has easy access to all the variables. When the optimization is finished the nodes holder can also be used to extract the solutions.

The NLP formulation class also contains two helper objects: a terrain and a robot model. The terrain is a height-map instance providing information of the ground, i.e. the height, normal vector and the corresponding derivatives. The robot model is an object containing methods for the system dynamics and kinematics (and derivatives). Pointers to these two objects are passed to constraints where needed, e.g. the robot model is passed to the dynamics constraint and the terrain object to the forces constraint.
The robot model is used for forward dynamics, not for a step-wise simulation. So each constraint that uses it will update the model with the state for a collocation point, read the output and update it again for the next point.

Integration with MuJoCo is accomplished through the `MuJoCoModel` class. MuJoCo itself is deliberately written in C instead of C++ for performance, and is therefore completely without object orientation. The `MuJoCoModel` class is an object oriented wrapper for MuJoCo making it much easier to use and the code more organized. It is unlikely this is at the cost of computation

speed for a C++ compiler with optimizations enabled.

The predefined gaits play an important role. To ease the gait definitions the same method from TOWR is used. For each type of legged robot (e.g. biped, quadruped, biped with toes, etc.) an extension of `GaitGenerator` is made. In each generator all possible modes are defined, which are combinations of each end-effector being in contact and in the air. The generator then has a number of strides, such as stance, flight or taking a step. Each stride consists of a set of modes with for each mode a duration. An entire sequence can then be easily composed by combining strides. An example is shown in Fig. 23.

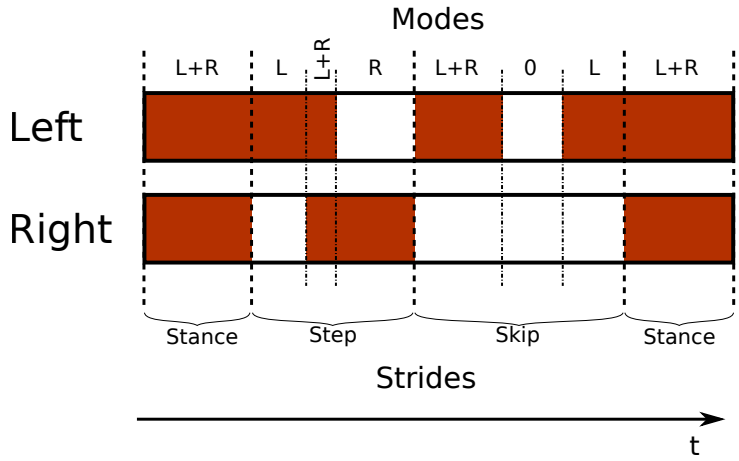

Figure 23: Illustration of the gait generator for a biped. The four different modes and the set of strides have to be defined only once. The strides can then be easily combined to form walking sequences.

It is significant to compile the program with the g++ `-Ofast` option, as it can improve the computation speed as much as five times compared to `-O0`. `-Ofast` enables all standard compile optimizations also used in `-O3`, as well as some optimizations that are outside the standard.

The complete source be found here: `https://bitbucket.org/ctw-bw/gambol/`.

4 - Results

In this section the results of the new framework are shown. Examples of generated motions for different models, gaits and terrains are shown, as well as the influences on the computation time and the performance in an online MPC setting.

4.1 Gait Generation

The framework is able to produce a wide variety of gaits for different types of robots on different terrains. Figure 24 shows an example of an optimized sequence for a biped in 3D. This example shows a full gait including single and double support, with heel strike and toe lift. Figure 25 shows the corresponding trajectory. This is only the suggested trajectory by the optimizer, not a simulated result.

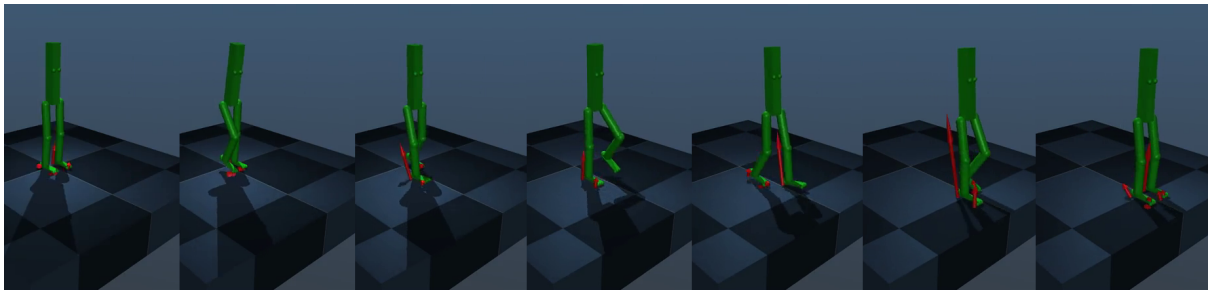

Figure 24: Resulting gait from a biped in 3D with heels and toes, consisting of 18 degrees-of-freedom. It has six defined end-effectors: the heel, inner toe and outer toe of each foot. This sequence consists of 25 collocation points. The red arrow represent the reaction forces. It starts and ends with zero velocity.

Models both in 3D and in only the sagittal plane can be optimized, the latter being significantly faster. The following robot models have been created and tried in gait optimization:

- Monoped (single end-effector, 3 DOF)
- Biped (two pin-foot end-effectors, 14 DOF, hip ball-joint)
- Biped with feet (six end-effectors (one for each heel and two for the toes), 18 DOF, with movable ankle)
- Quadruped (four pin-foot end-effectors, 18 DOF)
- Caterpillar (four-link chain with three end-effectors, without actual legs, in 2D)

All have been used successfully. Figures 26 to 28 show snapshots of examples of generated trajectories of these robots. Videos of the optimization results can be found here: `https://robert-roos.nl/category/masters-assignment`.

Most of the generated motions are intuitive and seem natural, with the exception for the monoped robot. It makes an unexpected twisting motion around the x and z axes. When an additional cost is added to the angular velocity of the base this twisting does not occur. However, the total squared sum of torques is larger for this straight motion, explaining the behavior. Similarly, bipeds optimized in 3D tend to place their feet awkwardly far apart. This likely another artifact because of the torque squared minimization. It could be the sum squared torque is less when the load is distributed over multiple joints, although this is not confirmed.

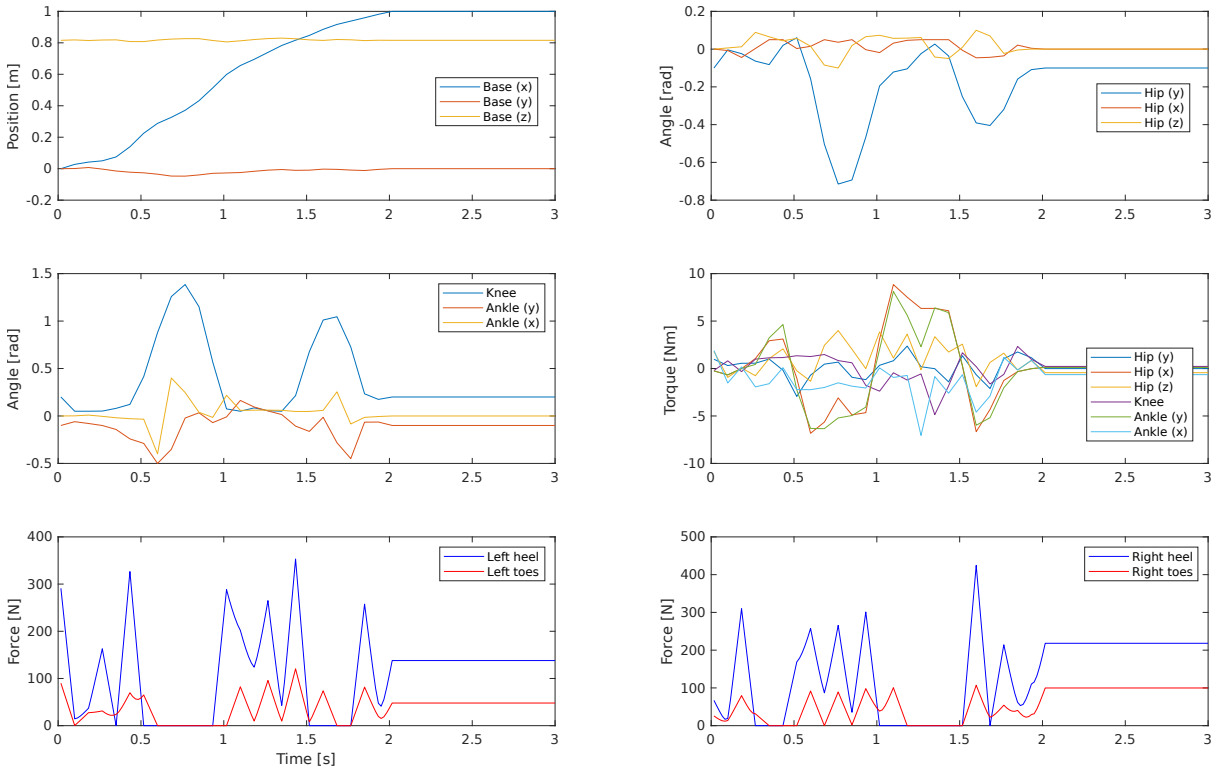

Figure 25: Graphs of the generated trajectory for the biped in 3D, corresponding to Fig. 24. The joints and torques for the right leg have been omitted.

Although not a practical robot, the caterpillar robot emphasizes the versatility of the platform. Any body can be used as an end-effector and as long as the gaits are defined a trajectory can be optimized.

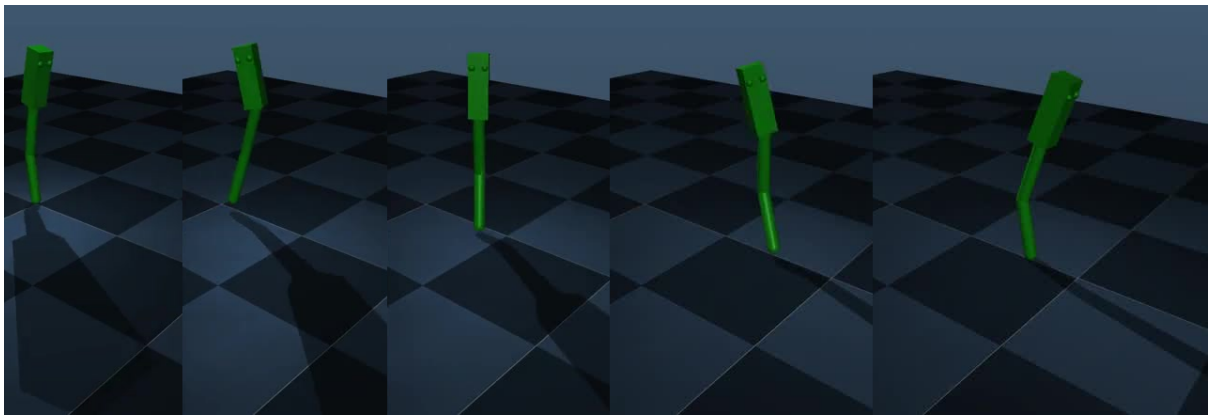

Figure 26: Gait sequence of a monoped.

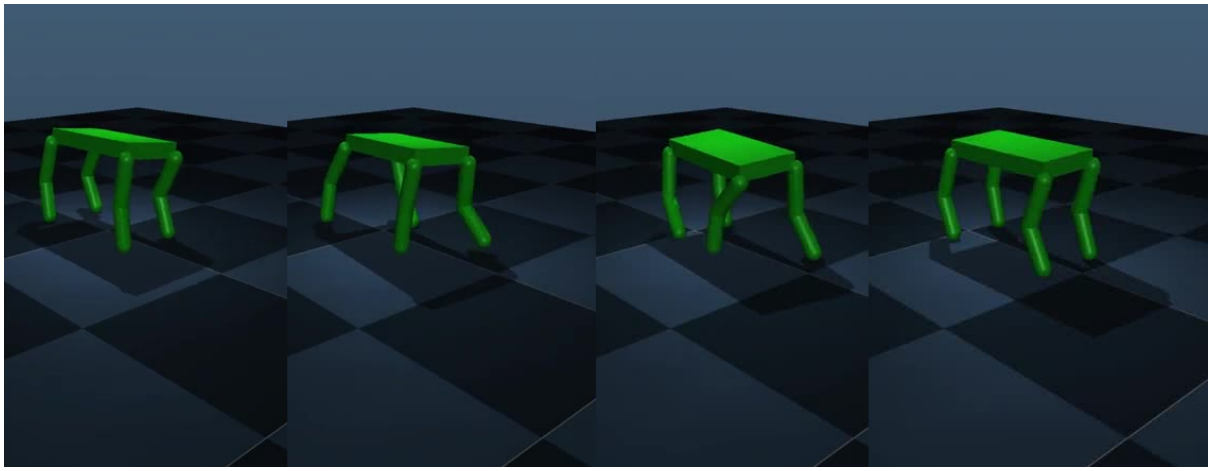

Figure 27: Gait sequence of a quadruped.

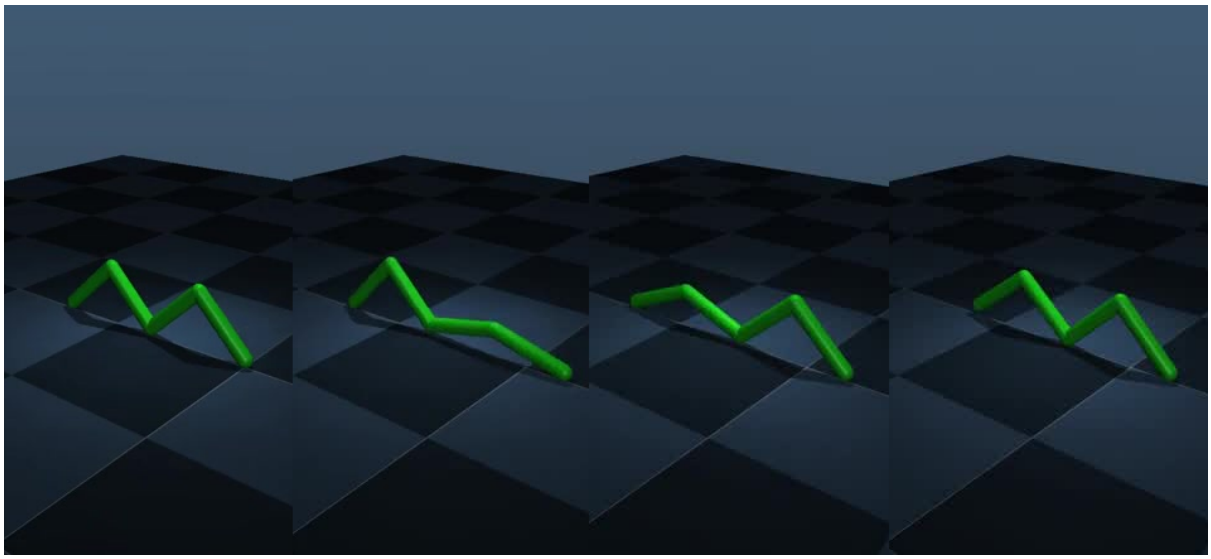

Figure 28: Gait sequence of a snake robot.

By changing the gait phases a different walking pattern emerges. Figures 29 and 30 show the same model with the same NLP starting point but with different gait phases. The emerging motions are specific the gait.

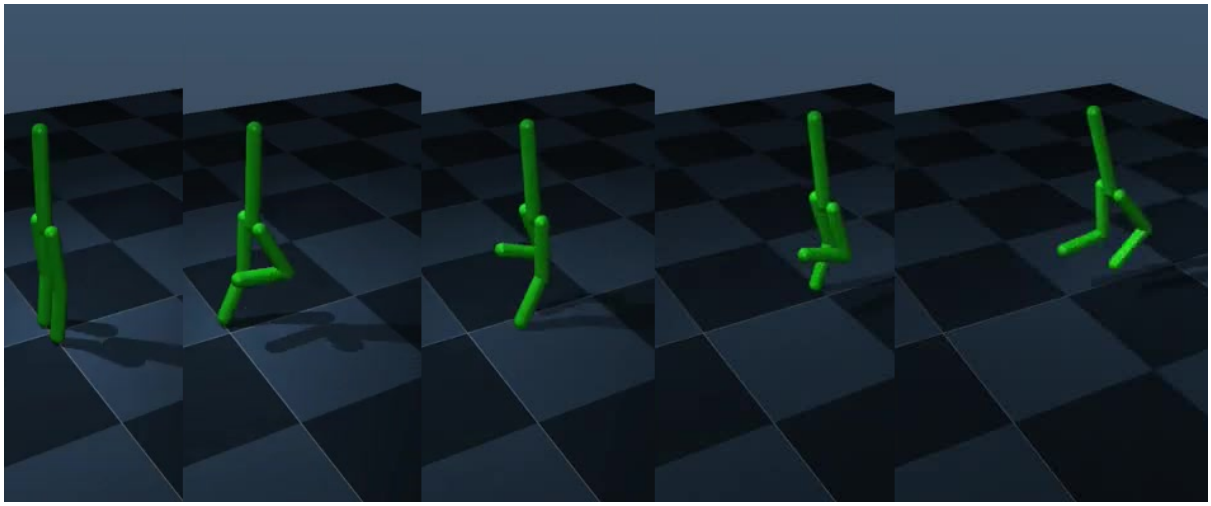

Figure 29: Biped model with a running gait.

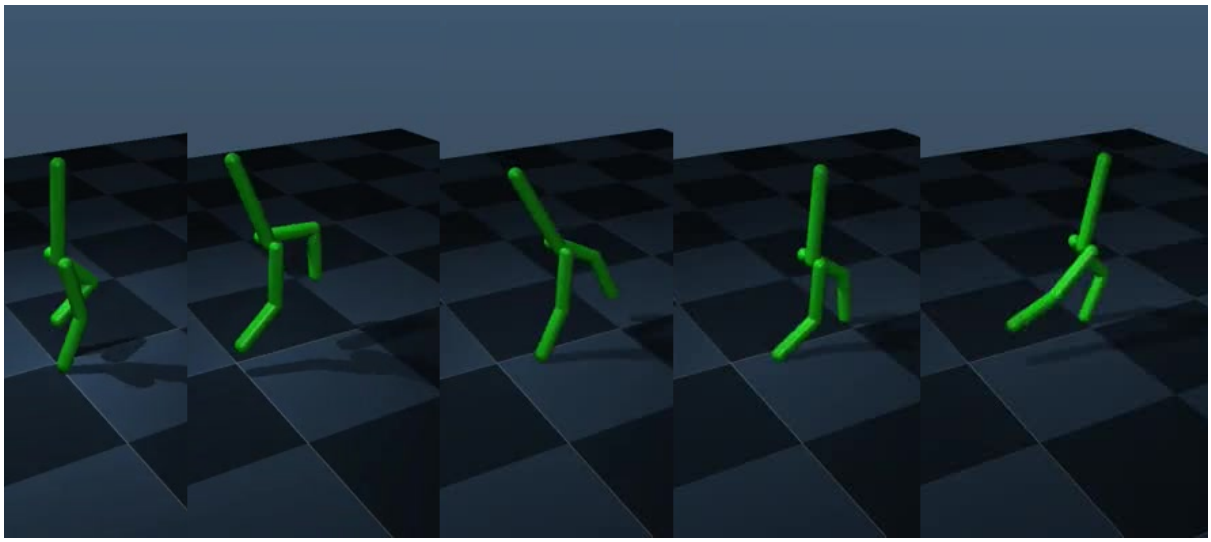

Figure 30: Biped model with a skipping gait.

Figure 31 shows the optimization with examples of terrains that are not flat, produced by simple height-maps.

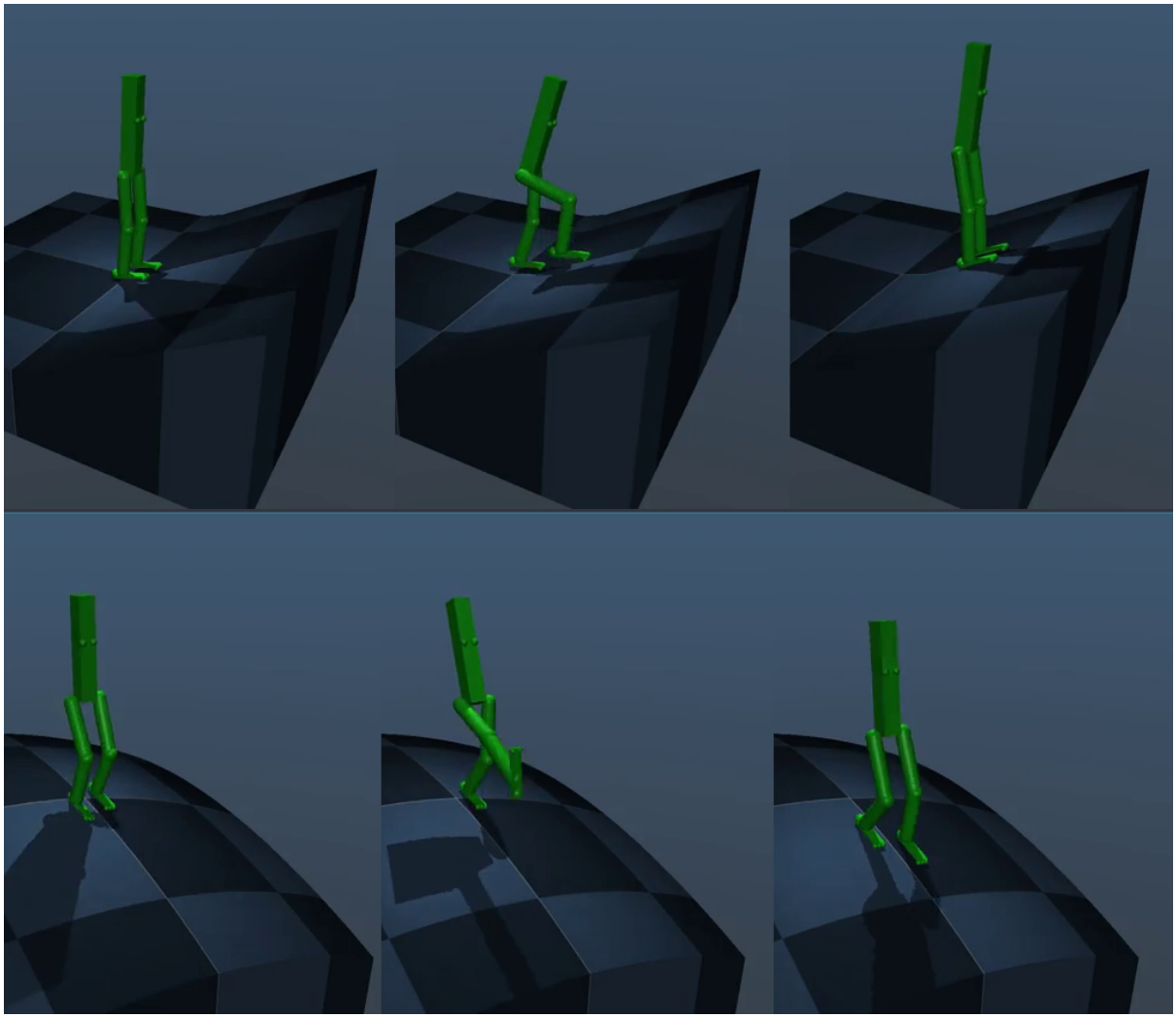

Figure 31: Biped model with terrain that is not flat.

4.2 Symbitron Exoskeleton

The framework was also tested with a model for the Symbitron exoskeleton (depicted in Fig. 1). The Symbitron exoskeleton is a powered lower-limb exoskeleton, intended for paraplegics. It has two actuated joints in each hip, one in the knee and one in the ankle. It also has a passive degree-of-freedom for ankle inversion and eversion, connected to a spring. The hip also has passive internal and external rotation but is connected to a very stiff spring. Figure 32 shows all degrees-of-freedom of the exoskeleton.

The kinematics and actuation of the systems is modelled in MuJoCo, without attention to precise dimensions or inertias. The goal is to test whether an exoskeleton with a structure like the Symbitron can balance on one foot and walk normally, despite the single actuator in the ankle.

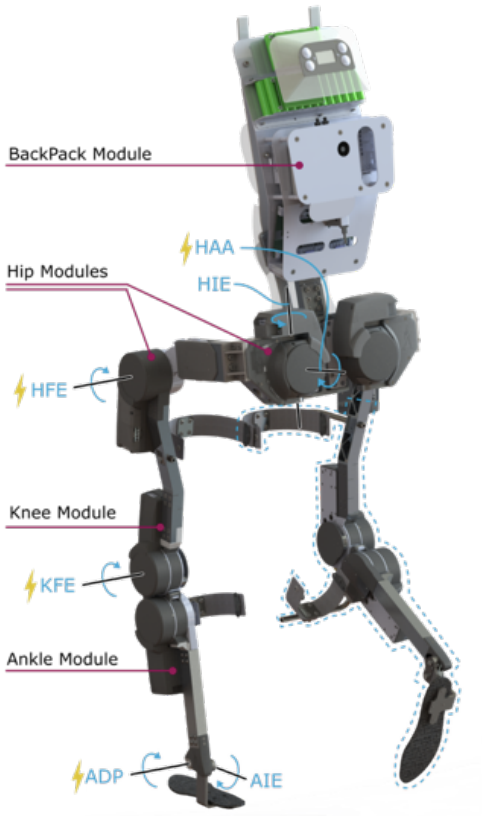

Figure 32: The Symbitron exoskeleton. The 6 degrees-of-freedom per leg are shown, with four of them actuated. The AIE axis (ankle in- and eversion) is passive and the HIE axis (hip internal and external rotation) is practically locked.

When the ankle is modelled in MuJoCo as a reasonably stiff spring, the optimization cannot converge. Only when the stiffness is insignificant the optimization can converge. This likely due to how the spring time constant compares to the time between collocation points. If this time constant is approximately equal to or smaller than the time between collocation points, no meaningful dynamics propagation can be done. This is comparable to a simulation where the timestep is large compared to a time constant in the simulated system.
Therefore the ankle is modelled as completely passive instead. The hip twist is considered as locked.

Figure 33 shows snapshots of the optimized trajectory for the exoskeleton balancing on a single leg. The Symbitron model can easily balance on one foot with zero initial velocity. Also when the base has an initial velocity, as if disturbed, the robot can regain balance. An initial velocity of up to 1.5 m/s can be compensated by the robot.

Snapshots of an optimized walking trajectory for the Symbitron exoskeleton can be seen in Fig. 34. Noteworthy is how the exoskeleton makes the first step without moving forward. This is perhaps needed to gain a forward velocity, as it starts completely stationary.

The optimized gaits for the exoskeleton differ only slightly from optimized results for a fully actuated biped, as in Fig. 24. The Symbitron exoskeleton has more sway in the upper body. However, the robot seems capable of walking and maintaining balance even with an actuated axis missing in both the hip and ankle.

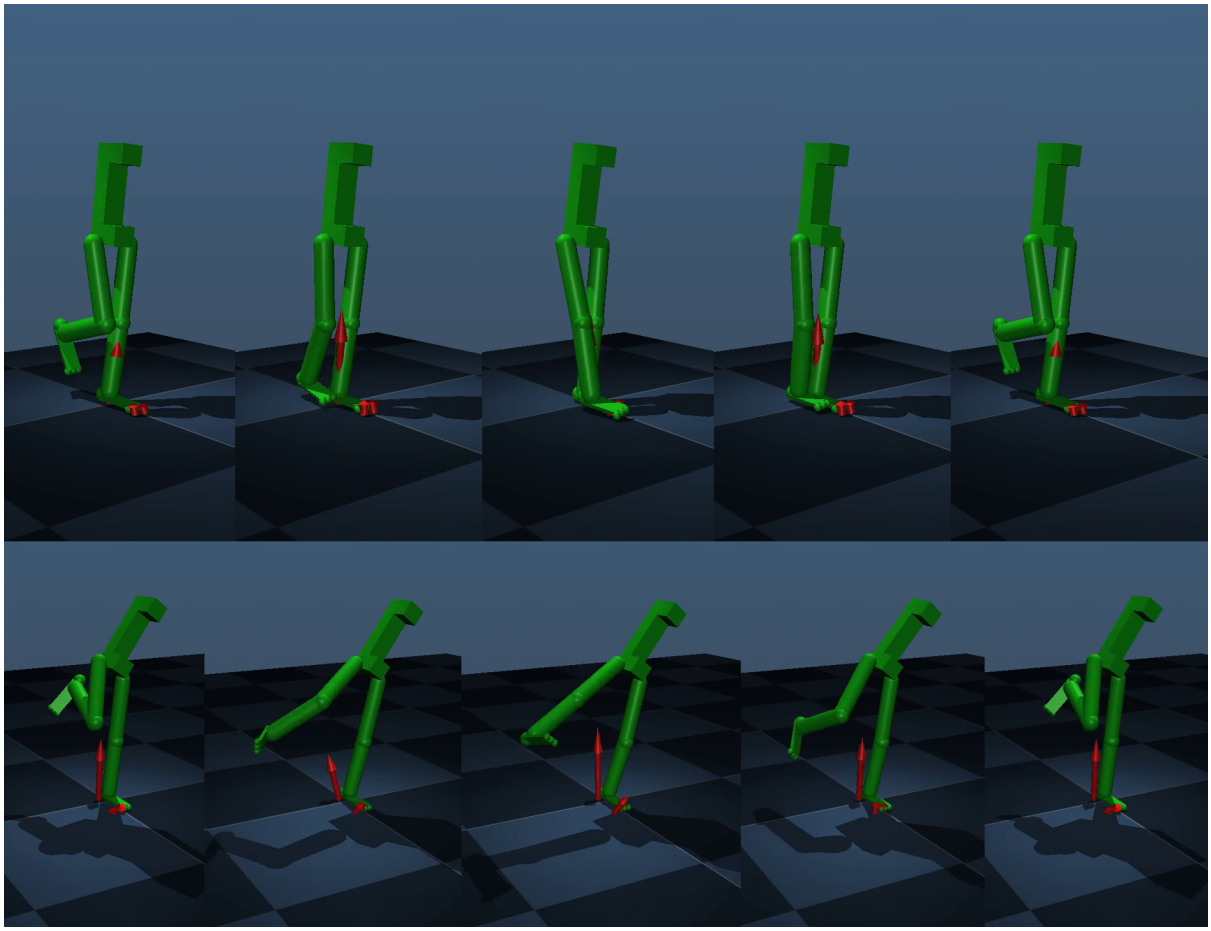

Figure 33: Snapshots of the optimization result for the exoskeleton balancing on one leg. The top sequence has no initial velocity, whereas in the bottom sequence the base has an initial velocity of 1.5 m/s to its left.

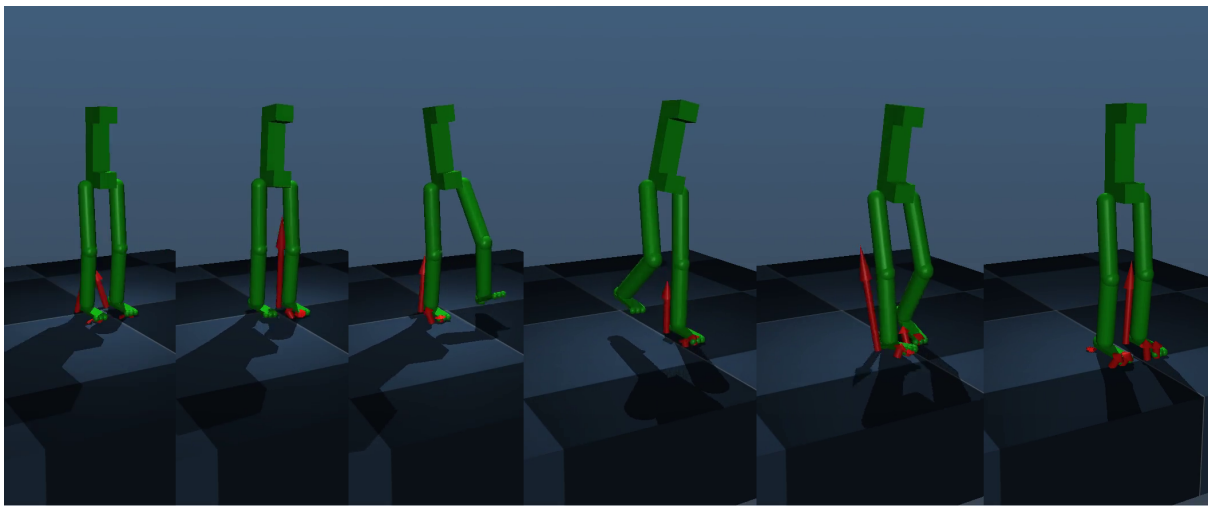

Figure 34: Snapshots of an optimized gait trajectory for the exoskeleton.

4.3 Influence of Collocation Density

Central in the technique of direct collocation is the number of collocation points N. Increasing the number of points will make the optimization more realistic, at the cost of computation time. The number of collocation points should be considered along with the duration of the optimized sequence. We describe the number of collocation points per second as the collocation density.

To investigate the influence of the collocation density the same sequence is optimized with an increasing value for N. The hypothesis is the optimized trajectory will start to converge to a 'real' optimal trajectory, at $N \to \infty$. In other words, for high values for N the optimized trajectory should no longer change.
The result of this experiment is shown in Fig. 35. A sequence of 2 seconds for the biped with heels and toes in 3D was optimized for 25, 35, 50 and 100 collocation points.
From the graph it can be seen the four trajectories are different, although they produce roughly the same overall motion. The angles at $t = 0.8$ seems to converge as the density increases. However, at $t = 1.7$ the trajectory for the angle of the knee first move away from the trajectory of $N = 100$. The torque profile mostly seems to become more noisy as the collocation density increases.

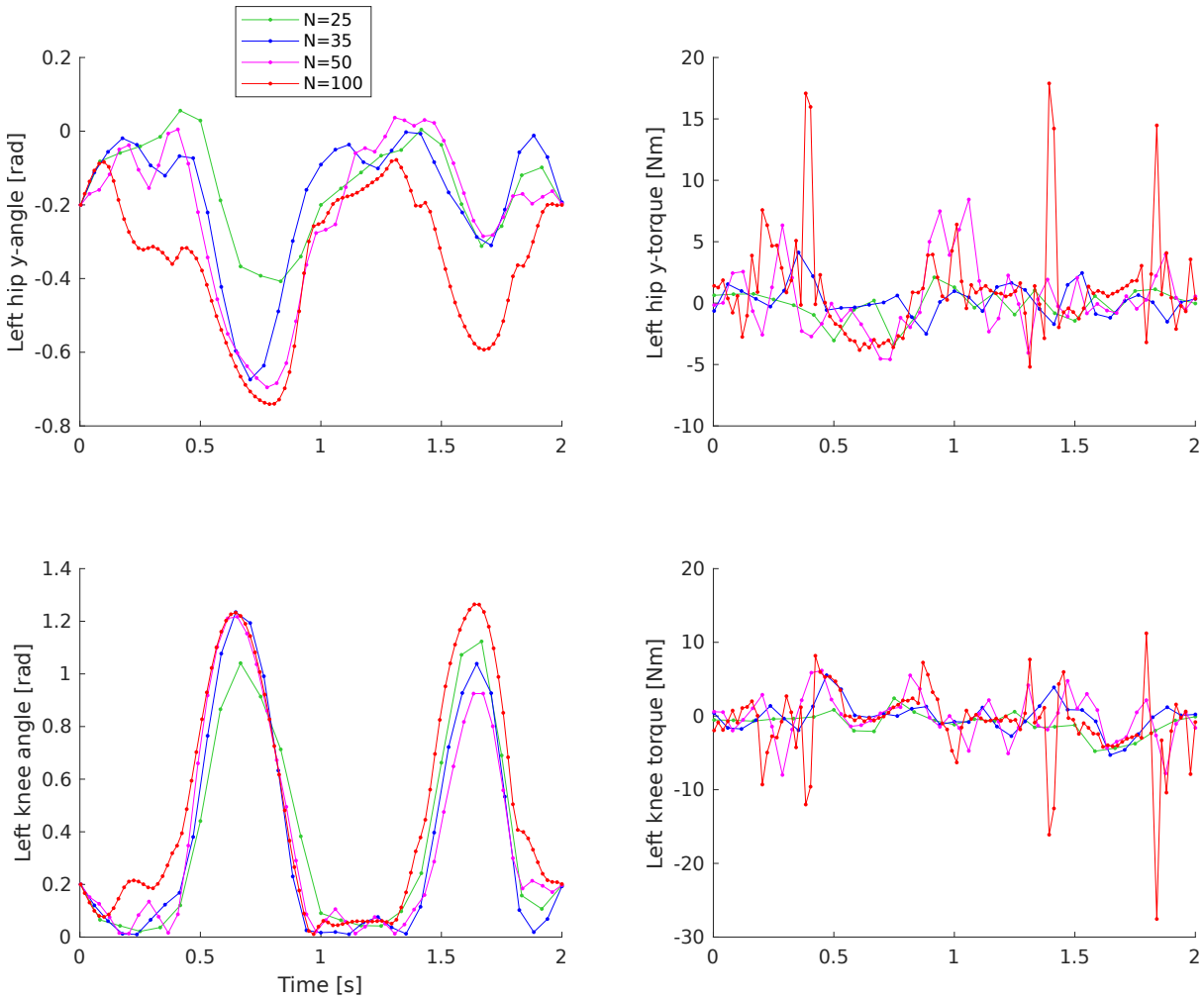

Figure 35: Angles and torques of the left hip flexion/extension and left knee, of the same optimization problem with an increasing number of collocation points N.

In conclusion, the trajectories themselves are dependent on the choice of N. Increasing N does not necessary converge to a final solution, at least not within the practical range for N. A useful collocation density seems to be around 15 points per second or higher, as determined qualitatively. A lower number can result in unnatural motions.

The optimization does not converge when there are more than 100 collocation points, IPOPT then throws a 'Converged to local infeasibility' error. This means IPOPT is no longer capable of satisfying the constraints from that current point. The error typically arises from problems that have no solution. Why this occurs in this situation is not clear. Since the less accurate formulations do converge, it seems unlikely the problem with higher density is unsolvable. There is no theoretical limit to the density of collocation points. Perhaps problems arise when the duration of nodes become closer to the time-steps in the finite-difference method. Note that the problem with $N = 200$ is also very sizable: it has 13,364 optimization variables, 17,163 constraints and 746,984 non-zero elements in the constraint jacobian.

4.4 Computation Time

The number of collocation points greatly influences the optimization time, though also the accuracy of the result. Figure 36 shows the average time per optimization iteration for different models. Increasing the size of the NLP through the number of collocation points increases both dimensions of the constraint jacobians, hence a quadratic relation between iteration time and collocation points is expected. However, the figure shows a roughly linear relation. This is likely because of the sparse nature of the constraint jacobians. The jacobians of the node constraints are only non-zero around the diagonal. The number of iterations needed per optimization is shown in Fig. 37. More iterations are needed for a larger number of collocation points. And a more complex model does not always require more iterations to be solved. Figure 38 shows the total optimization time needed for different models. It appears the optimization time is not exponential to the number of collocation points, it could be approximately quadratic. Finally Fig. 39 displays the final value for the objective function for different numbers of collocation points. It shows the final cost decreases with a higher number of collocation points. However, this decrease almost stagnates after about 20 points. The monoped in 2D does not quite adhere to this trend, the final objective value is more sporadic. This is likely due to the fact the monoped has to jump as a walking gait, making the trajectory more sensitive to dynamic incorrectness for fewer collocation points.

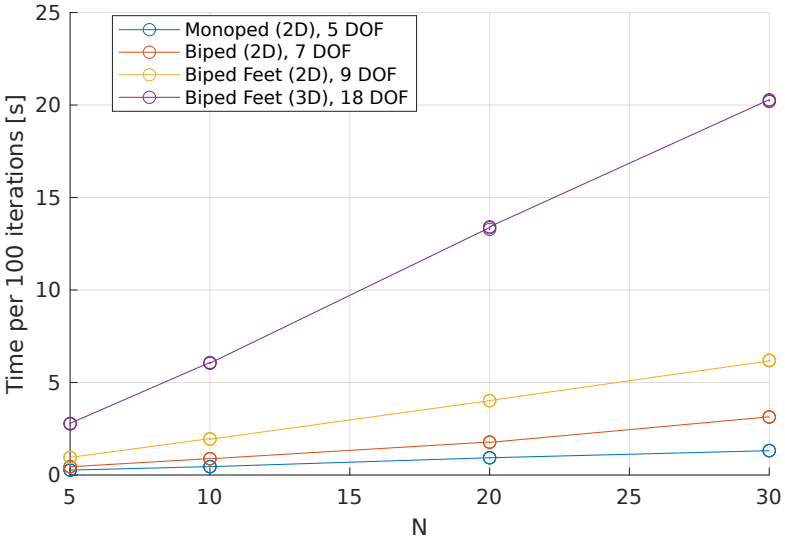

Figure 36: Optimization time per 100 iterations for different models and number of collocation points.

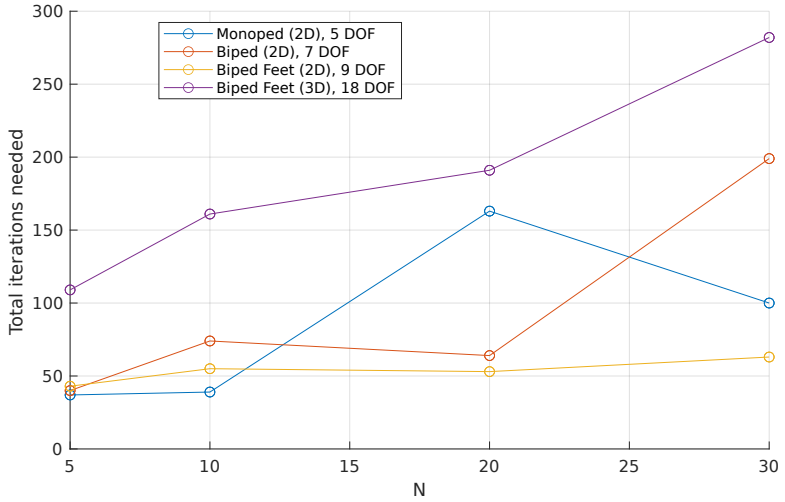

Figure 37: Number of iterations needed for optimization, for different models and number of collocation points. Note that the 9 DOF model requires fewer iterations than the 7 DOF model.

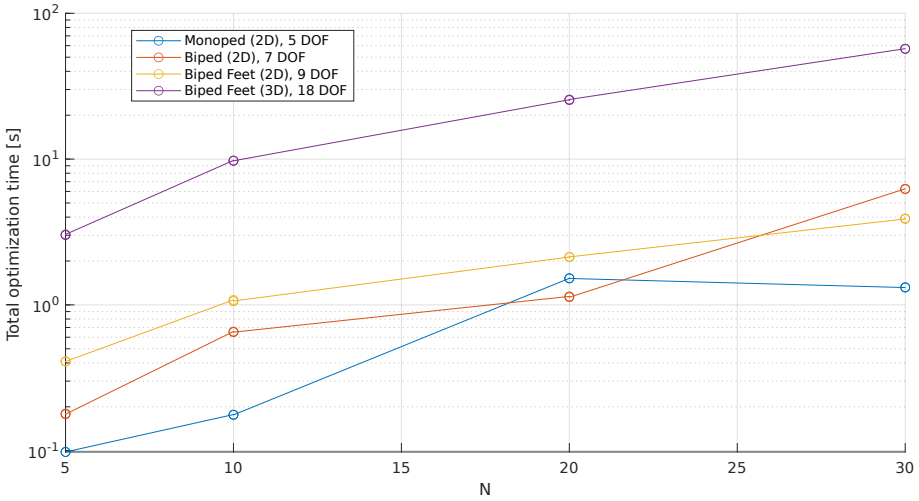

Figure 38: Total time needed for optimization of different models and number of collocation points on logarithmic scale.

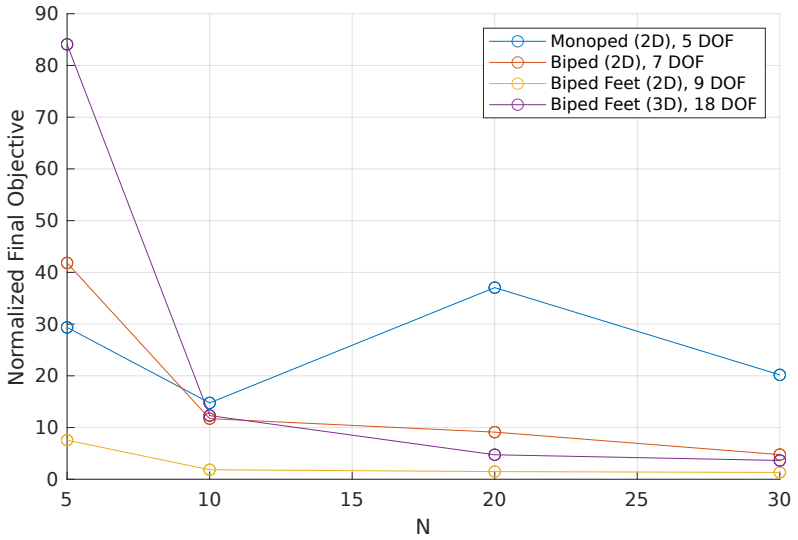

Figure 39: The final objective value (torque squared, normalized per collocation point) for different collocation densities, for different models.

Figure 40 shows the progress of solving the NLP over the course of iterations. It shows the objective to be minimized quickly, most of the iterations are spent satisfying the constraints. It is interesting that the constraint violation does no longer steadily decline after 70 iterations, but instead changes almost step-wise.

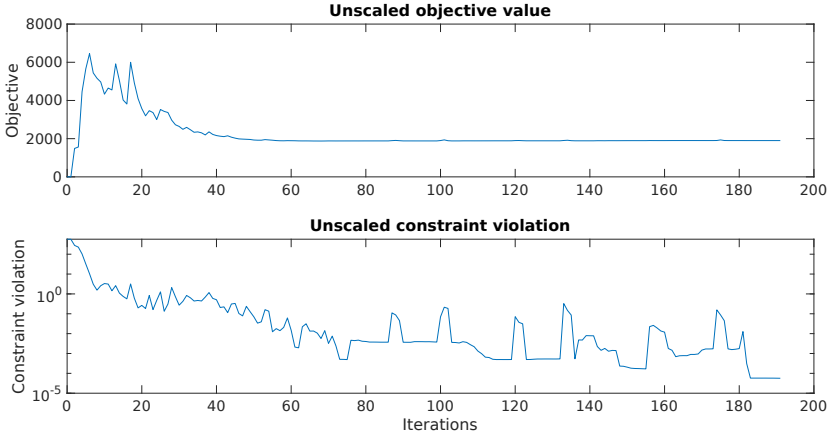

Figure 40: The objective function and constraint violation for the biped in 3D, $N = 20$.

The implementation of arbitrarily definable terrains is useful, though it adds to the computation time. With the addition of non-flat terrain, the terrain constraint needs to look up the height, height gradients, normals and normal jacobians for each end-effector, for each point. The force gradient also needs to take these extra steps. And it now needs to include the robot model too, to compute the end-effector location at a certain point. This results in about 15% more computation time per iteration, even for flat terrains.

4.5 MPC Setting

Multiple models were tried in a simulated setting. A MuJoCo simulation of a model was run, with input torques suggested by the optimizer and a feedback controller applied to the trajectory, as put forward in Sec. 3.4 *Online MPC Loop*. The optimizations were repeated with the current state as initial configuration at an interval ΔT. This was not performed in real-time, during the optimization the simulation was simply paused. Although unrealistic, this does allow us to test the influence of ΔT on the online MPC performance.

The simulations show that when only the optimized torques are applied to the system, the joint trajectories diverge from the optimized trajectories. When the optimization is run only once without being repeated, as the only input, the robots fall over almost instantly. This divergence indicates ΔT should be small for a feasible control loop. The PD-controller is calibrated by running an optimization of the robot standing still: the parameters are chosen such the robot stays upright.

Experiments with the framework show most models cannot be made to go through a stable gait, regardless of values for ΔT and the feedback controller. The biped with heels and toes in sagittal plane is a simpler model that does work in simulation. Figures 41 and 42 show snapshots of this trajectory and graphs of the trajectory. For this simulation $\Delta T = 0.1$ seconds, which is a realistic value. The simulated gait is not as smooth as the generated one, the body twitches during simulation. This is particularly noticeable in the legs during swing phase. The graph shows the reference is provided by the optimizer is being tracked well. However the ground

reaction forces do not align closely.

To investigate the divergence between repeated optimizations, each optimized trajectory is saved and plotted together. This is done in Fig. 43. As expected, the repeated optimizations are not all identical. It seems that the actuated joints (like the hip angle) diverge little, whereas un-actuated degrees-of-freedom (like the base angle to the ground) diverge significantly. In the case of the 2D biped the repeated optimization compensates this properly.

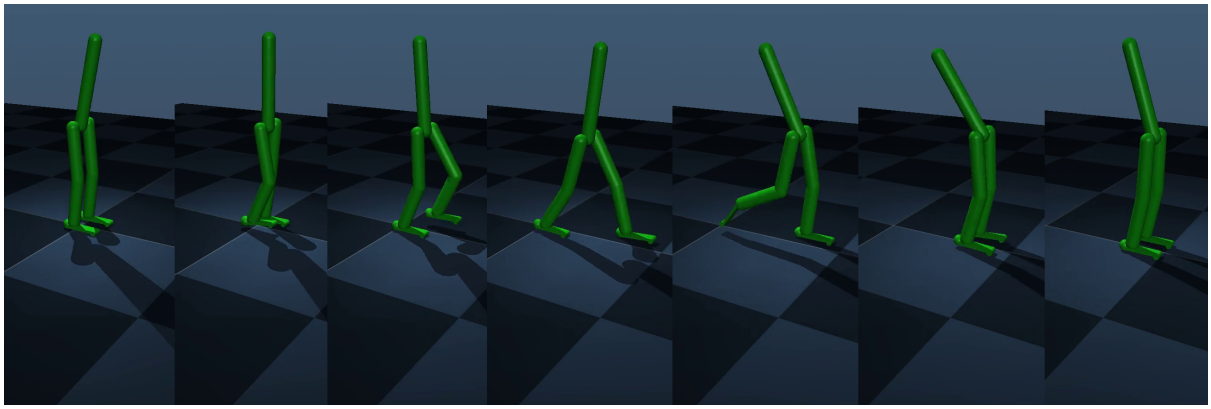

Figure 41: Applied MPC with feedback on a biped with heels and toes in the sagittal plane. The MPC rate is 0.1 seconds and feedback settings $K_p = 20$, $K_d = 2$. The initial and final velocities are zero. The robot twitches noticeably during the simulation, especially in the ankles during swing.

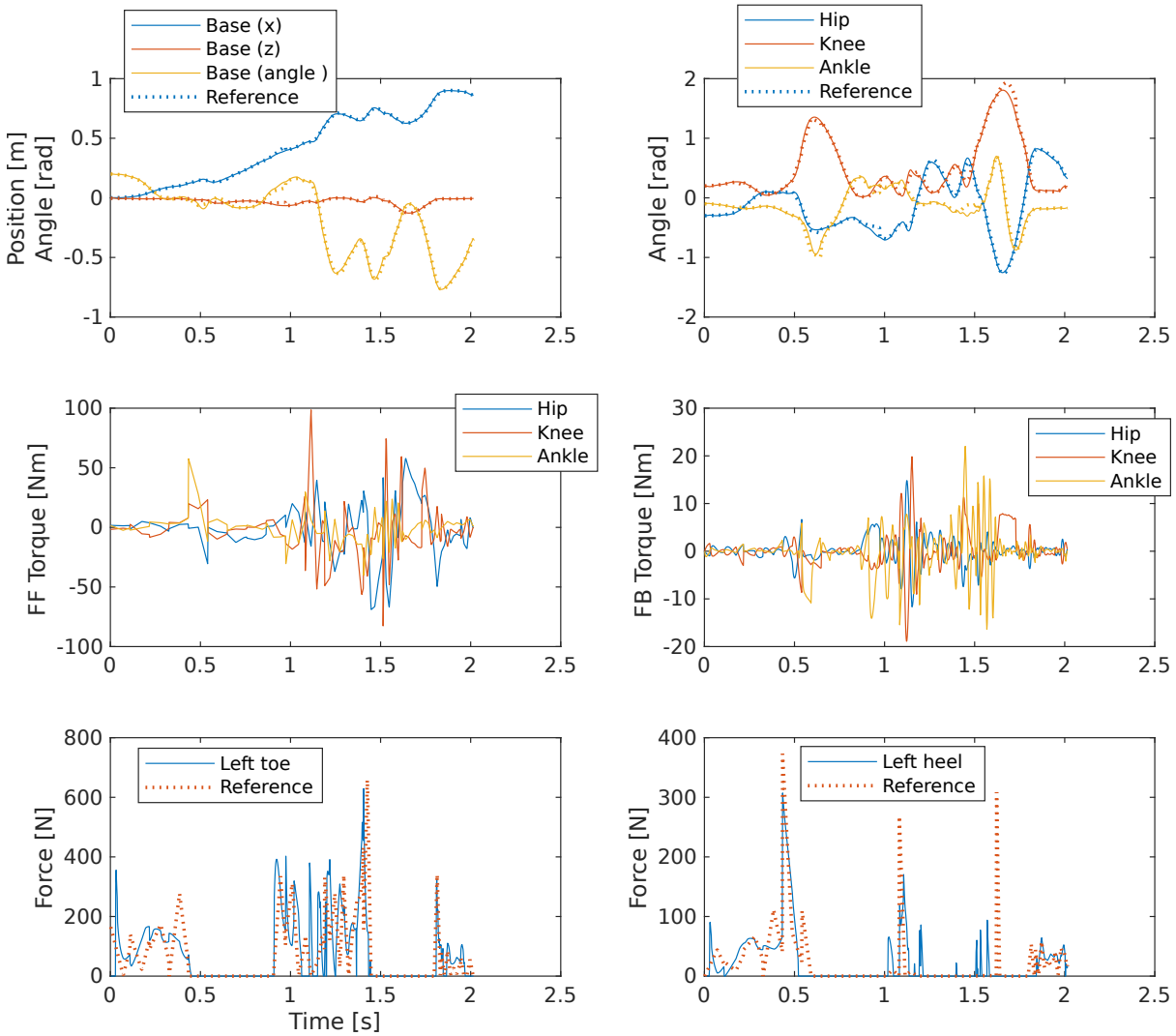

Figure 42: Graphs of the applied MPC with feedback on a biped in the sagittal plane. Figure 41 shows the corresponding snapshots. 'FF Torque' represents the feedforward torque (optimization result) and 'FB Torque' the feedback torque.

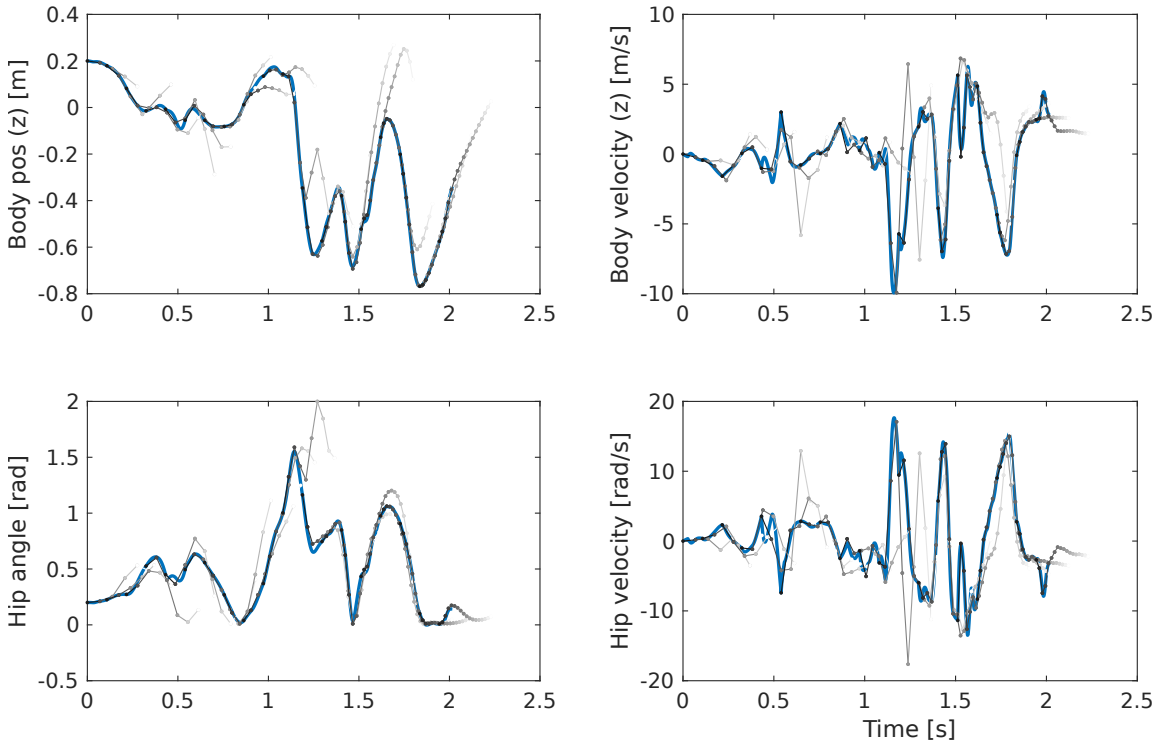

Figure 43: Plot of the body angle and hip angle of the biped walker in the sagittal plane, corresponding to the simulation results in Fig. 41. The blue line shows the simulated result and the gray lines the repeated references from the trajectory generation. The markers correspond to the collocation points. The references are shown for longer than ΔT and fade out, showing how new optimized trajectories differ from previously optimized ones.

Figures 44 and 45 show the snapshots and graphs of the 3d biped. The simulation was not successful, the robot starts to collapses during the first step. When the robot is collapsing the optimizations also fail to converge, indicating the pose becomes irrecoverable. The MPC interval ΔT was 0.1. Decreasing it further does not improve performance for this model, the motions then only become more erratic. Figure 46 shows how the real trajectories quickly diverge from the optimized ones.

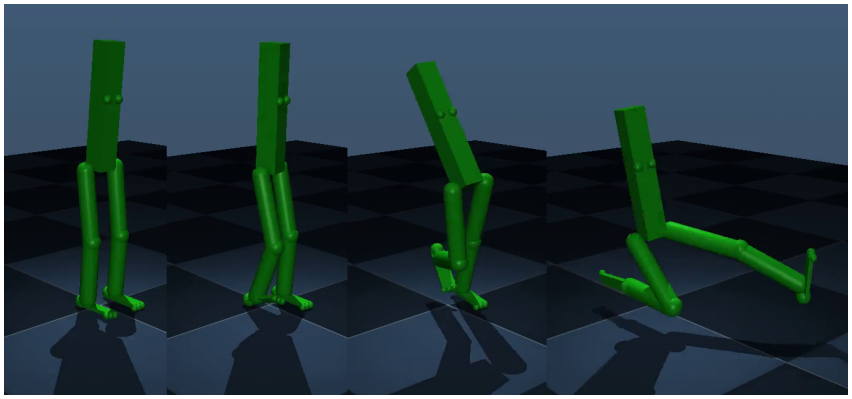

Figure 44: Applied MPC with feedback on a three dimensional biped, including heels and toes. The MPC rate is 0.1 seconds and feedback settings $K_p = 20$, $K_d = 0.5$. The initial and final velocities are zero. The robot collapses after about 0.6 seconds.

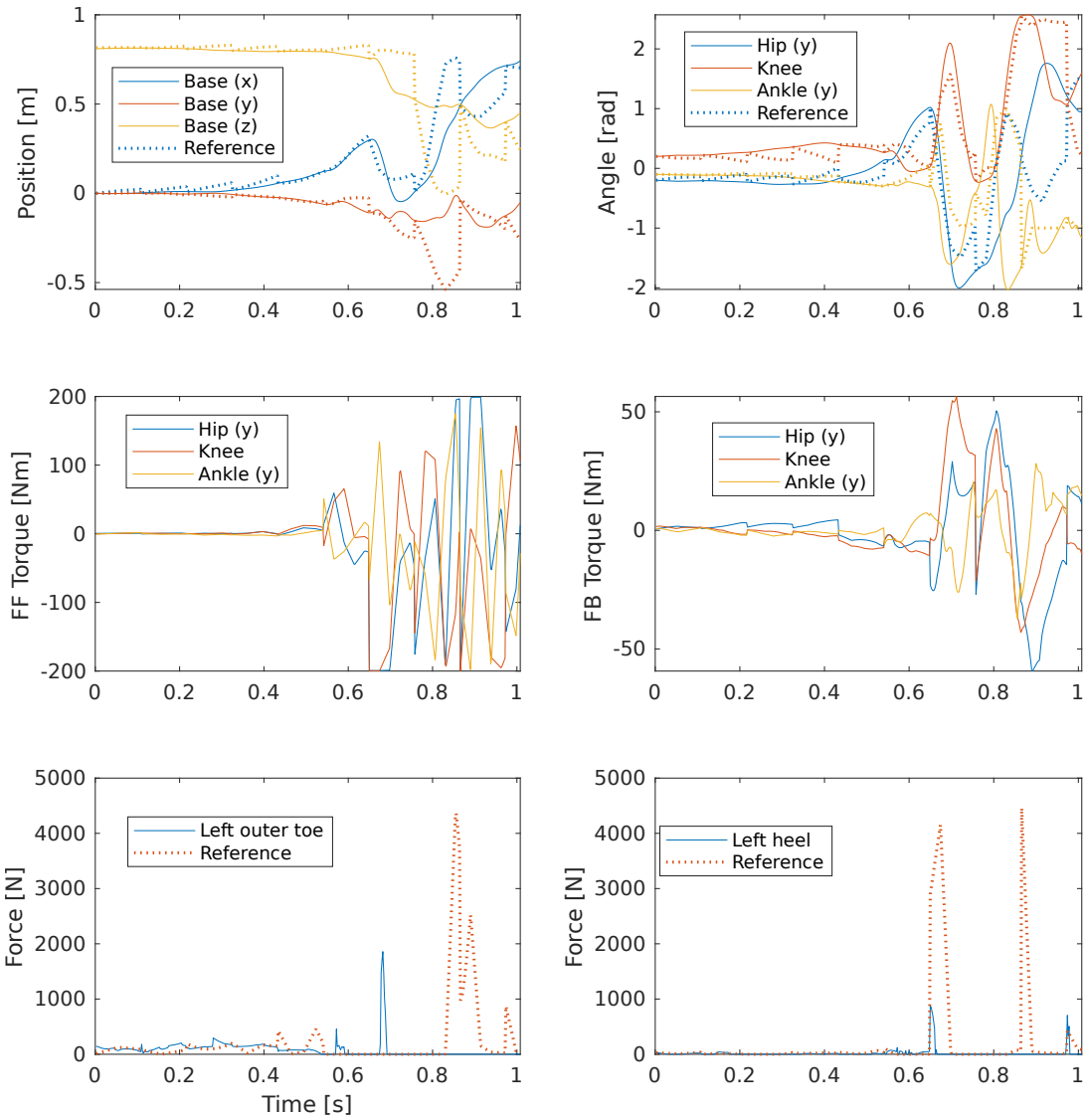

Figure 45: Graphs of the applied MPC with feedback on a biped in 3D. Figure 44 shows the corresponding snapshots. 'FF Torque' represents the feedforward torque (optimization result) and 'FB Torque' the feedback torque. The robot collapses after about 0.6 seconds. The references are being tracked poorly from the start. The graphs show how every 0.1 second the references are reset to the current position.

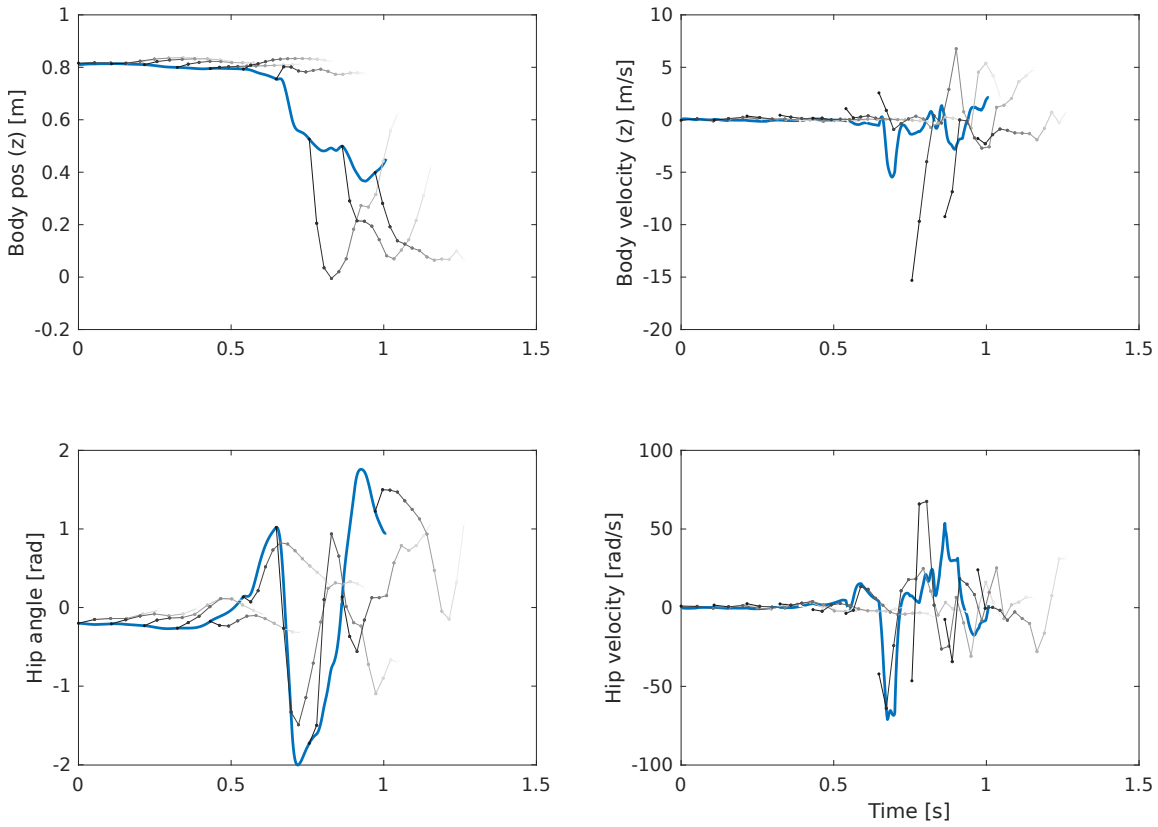

Figure 46: Plot of the body position and hip y-angle of the biped walker 3D, corresponding to the simulation results in Fig. 44. The blue line shows the simulated result and the gray lines the repeated references from the trajectory generation. The markers correspond to the collocation points. The references are shown for longer than ΔT and fade out, showing how new optimized trajectories differ from previously optimized ones. In the velocity plots the optimized trajectories do not always start from the real trajectory. This is a consequence of the optimizations not converging properly.

A quadruped robot is inherently more stable than a biped, as it has at least four contact points which span a large support surface underneath the body. So the same online MPC experiment was performed with a quadruped with pin-feet in 3D. Figures 47 and 48 show the snapshots of such a simulation and graphs describing the motion. A ΔT of 0.01 seconds was needed to result in any motion.
Although the quadruped stays largely upright, it barely moves forward. The feet bounce on the floor and the robot ends with its body tilted backwards slightly, making it that the base only moved backwards. Many of the repeated optimizations did not converge properly.

The biped requires repeated optimizations to stay upright. The same is not true for the quadruped, only a single optimization combined with a stronger feedback controller is sufficient. The additional support surface is helpful in this regard. Gait simulations without repeated optimizations for the quadruped resulted in very similar motions as Figs. 47 and 48, only with feedback control torques exceeding the feedforward torques. This leaves little optimal nature to the realized motion. No forward motion was achieved either.

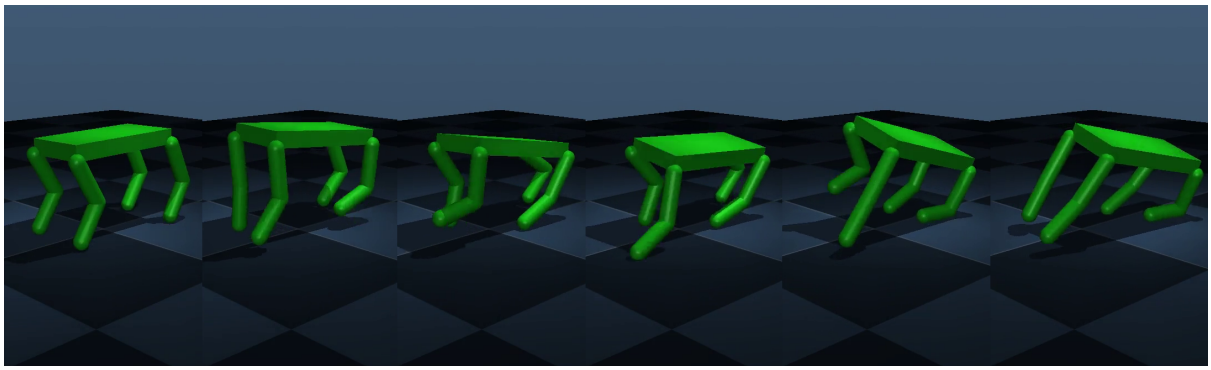

Figure 47: Applied MPC with feedback on a quadruped in 3D for $\Delta T = 0.01$ and feedback settings $K_p = 20$, $K_d = 2$. The motions were jittery and the robot does not actually move forward.

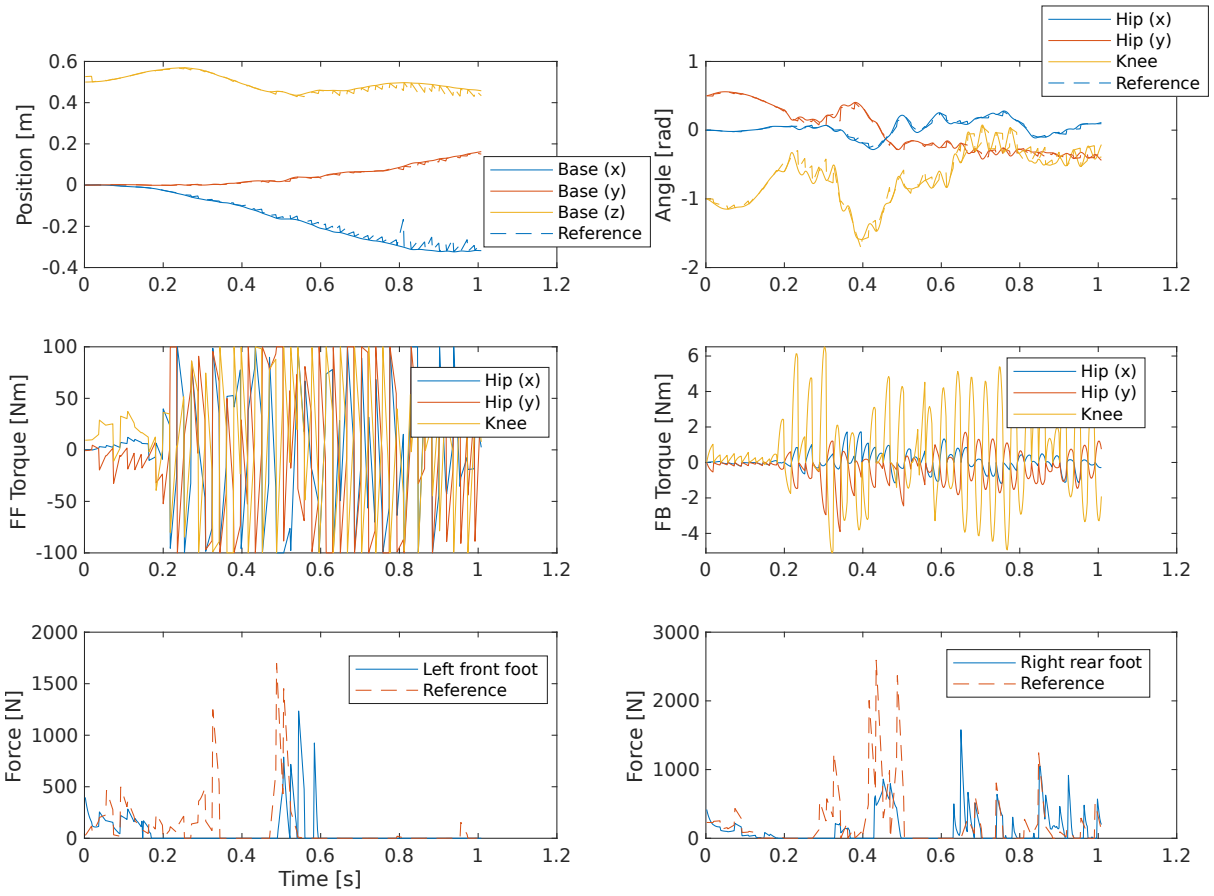

Figure 48: Graphs of the applied MPC on a quadruped in 3D. Figure 47 shows snapshots of the corresponding motion. The angles of the left-front leg are shown. The reaction forces are from the left-front and right-rear feet. The plot of the base position shows the robot slumps backwards, instead of moving forward with the desired 0.4 m. Many of the repeated optimizations did not converge properly, indicated by the very noisy feedforward plot.

5 - Conclusion and Discussion

5.1 Conclusion

To evaluate the designed framework it is compared to the original requirements and objectives. Table 2 summarizes this evaluation. Both requirements have been satisfied: walking gaits can be produced with regard to whole-body dynamics. The gait is optimized in joint torque, which is the most useful minimization.

Most objectives were accomplished well. The optimizations are robust as no settings need to be tweaked and no specific initial guess is needed. The foothold locations do not need to be chosen manually but are found naturally, given a basic interpolation from initial to final configuration. Most useful is the flexibility to the robot model. Tweaking model setting or modifying the kinematic chain can be done in the model XML and are handled immediately by the framework. Adding new models can be done easily by providing a new XML, noting the named end-effectors and describing a basic joint configuration. The number of end-effectors in such a model is entirely free as well. These comments are specific to the implementation with MuJoCo, though it could be easily replaced with a different simulator or analytical functions by extending the abstract robot class.

Requirement	*Succeeded*	
Produce walking gaits with efficient actuation	Yes	Gaits are produced with optimized joint torque
Consider whole body in optimization	Yes	With the MuJoCo simulator all degrees of freedom are incorporated
Objective	*Score*	
Optimize footholds along with joint trajectory	++	Foothold locations are found automatically
Require little user input to produce a gait	-	Gait phases have to be predefined
Fast to compute	-	Optimizations are typically in the order of 10 seconds
Robustness to optimization parameters	+	Optimizations are easily set-up and always succeed
Variable number of legs	+	There are no limitations to robot kinematic layout
Easy to modify/replace robot model	++	Any MuJoCo xml file can be used, with minimal manual settings

Table 2: The design requirements and objectives and whether they were satisfied by the new framework.

Although the generation of locomotion trajectories on itself is widely applicable and works as desired, it proved ineffective when applied in a practical online MPC fashion. Only for a simpler model reasonable closed-loop performance was established. For other models the closed loop simulations were not stable, even for higher MPC rates.

Perhaps this new framework is best suited for offline use, for instance to provide training data for self-learning models or to optimize machines that repeat the same motion, where the computation time is of less significance. Compared to self-learning algorithms which typically train for hours [33], this framework is still fairly fast. The optimization times are within a scale where it is comfortable to try different inputs and seeing their impact on the results.

In conclusion, we accomplished designing a model predictive control framework for legged locomotion, based on explicit gait phases. However, we showed it cannot be directly applied in an online fashion for robust control.

5.2 Discussion

5.2.1 Gait Generation

Not all objectives were adhered to. Most lacking is the computation speed. Simpler 2D models with fewer collocation points can be optimized in the order of seconds on a regular computer, while more complex 3D models with more collocation points need closer to 30 seconds. Considering an online MPC application generally has an update rate in the order of 0.1 seconds, our framework is 10 to 100 times too slow.

Moreover, the gait generation is not free of manual input. The gait phases have to be completely predefined. Although this is made easier with the mode-based generator, every sequence needs a deliberate input.

It is clear a method of implicit contact is preferable over explicit contact, as it requires less manual input and every aspect of the gait is optimized. In this work we concluded explicit contact is not feasible in combination with direct collocation MPC. However, Neunert et al. [12] did succeed in this, and even with an impressively fast platform. It is possible the difference is in the numerical solver: Neunert et al. [12] tailored their own iLQR-NMPC solver, whereas we used the of-the-shelf solver IPOPT. If at all possible, it is recommended to choose a method of implementation that allows such implicit contacts.

5.2.2 Online MPC

As pointed out before, the framework is not directly suitable for online model predictive control. A hypothesis is this is because of the reaction forces on the system. As can be seen in Figs. 42, 45 and 48, the predicted reaction forces differ greatly from the reaction forces in simulation. This is not completely surprising, as the optimizer is free to choose the reaction forces as long as they fit the force and dynamics constraints. The expectation was they would be close to the real forces, though we showed this was not the case. With different forces on the system than during the prediction the trajectories diverge.
To investigate the performance of online MPC, in another experiment the predicted forces were applied directly during simulation, while collision detection was disabled. In this setting with disabled contacts, there is no difference between modelled and real reaction forces. The expectation is the produced gait will now be realized more properly. The results are shown in Figs. 55 to 58 in appendix F. With a high number of collocation points these non-realistic simulations show better results than the ones with realistic contact. This is most clear for the quadruped, for the biped models the feet slip away to the sides quickly. This shows that discrepancies between modelled and real reaction forces are a factor but not the sole cause of poor simulation performance.
The fact remains that a walking robot is a highly underactuated system. And simple feedback on the joints which are actuated is not sufficient to keep the robot stable. Perhaps a more complex controller that is better equipped for an underactuated system could make the simulation results better.
All things considered there is no proven cause for the inadequate simulation results.

Platforms using centroidal dynamics [1, 10] did seem to be successful in practical implementation. It would be interesting to apply this new framework and an existing centroidal frame-

work plus whole-body controller on the same MuJoCo simulation to compare their performance.

This work set out to develop a framework focused on whole-body dynamics, with the notion that only with whole-body dynamics a truly optimized trajectory can be found. The results of this work indicate that this notion could be flawed. If the real system and reaction forces are not going to match with the prediction, a significant amount of additional control torque will be needed, defeating the purpose of the torque optimization. It might therefore be preferable to have a more abstract optimization based on centroidal dynamics that provides a less specific but more robust trajectory. Moreover, such a framework based on simplified dynamics could run significantly faster and therefore more often, resulting in even more robustness.

5.3 Future Work

To improve the current framework the code could be optimized. The MuJoCo model could be used more effectively for instance. Currently the model is updated for each node for each constraint. It would be more efficient to update the model for a node, then compute all constraints and then move to the next node, requiring at most N updates.
Additionally the finite-difference approach in MuJoCo could be sped up by using multi-threading: each differentiation variable could be varied in parallel. OpenMP is used for this purpose in MuJoCo examples [28].

The biggest shortcoming of this framework compared to TOWR [1] is the lack of phase optimization. As mentioned before, the use of universal nodes makes it impossible to have complete freedom in the gait phases. However, it is still possible to shift the entire nodes in time (see Fig. 49). For N nodes, $N-1$ optimization variables could be added each representing the time between two nodes. In each node the state for an end-effector is then fixed, though the durations can still be optimized. In a hyper-parameter optimization the order of gaits could be changed to still get a global optimum with respect to gait phases. The difficultly in adding node times as variables is defining jacobians to all constraints with respect to these durations.

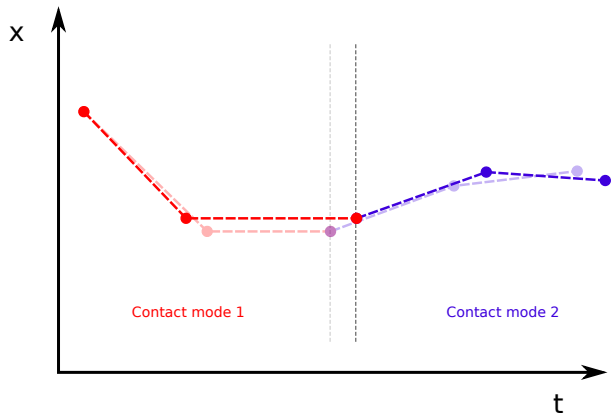

Figure 49: Two iterations of a optimization solution when node durations can be varied. If node durations were included in optimization variables, the contact mode duration could be varied as well, unlike in the current implementation. Note that because universal nodes are used, only complete contact modes can be varied, not individual end-effector phases.

Another improvement would be removal of the need to explicitly define gait phases at all. The definition of gait phases limits the freedom of the optimization greatly, limiting it to the insight of the human user. Such an improvement will have to be accomplished with a method of implicit contact.

References

[1] A. W. Winkler, D. C. Bellicoso, M. Hutter, and J. Buchli, "Gait and Trajectory Optimization for Legged Systems through Phase-based End-Effector Parameterization," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, pp. 1560–1567, July 2018.

[2] S. Jazayeri, S. Beygi, F. Shokraneh, E. Hagen, and V. Rahimi-Movaghar, "Incidence of traumatic spinal cord injury worldwide: a systematic review," *European Spine Journal*, vol. 24, no. 5, pp. 905–918, 2015.

[3] University of Twente, "Symbitron+ Exoskeleton." `www.wearableroboticslab.nl/Symbitron-Exoskeleton-Race-Team/`.

[4] I. Opheusden, "Design of a control strategy for obstacle crossing in a lower limb exoskeleton for SCI patients." Master Thesis, Univeristy of Twente, Dec. 2017.

[5] V. Paas, "Feasability of model predictive control for exoskeletons." Master Thesis, University of Twente, May 2019.

[6] M. Kelly, "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation," *Society for Industrial and Applied Mathematics*, Feb. 2016.

[7] M. Bjelonic, P. Sankar, C. Bellicoso, H. Vallery, and M. Hutter, "Rolling in the deep - hybrid locomotion for wheeled-legged robots using online trajectory optimization," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3626–3633, 2020.

[8] M. Bjelonic, C. Bellicoso, Y. De Viragh, D. Sako, F. Tresoldi, F. Jenelten, and M. Hutter, "Keep rollin'-whole-body motion control and planning for wheeled quadrupedal robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2116–2123, 2019.

[9] V. Medeiros, E. Jelavic, M. Bjelonic, R. Siegwart, M. Meggiolaro, and M. Hutter, "Trajectory optimization for wheeled-legged quadrupedal robots driving in challenging terrain," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4172–4179, 2020.

[10] J. Di Carlo, P. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," *IEEE International Conference on Intelligent Robots and Systems*, pp. 7440–7447, 2018.

[11] B. Katz, J. Carlo, and S. Kim, "Mini cheetah: A platform for pushing the limits of dynamic quadruped control," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May, pp. 6295–6301, 2019.

[12] M. Neunert, M. Stauble, M. Giftthaler, C. Bellicoso, J. Carius, C. Gehring, M. Hutter, and J. Buchli, "Whole-body nonlinear model predictive control through contacts for quadrupeds," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1458–1465, 2018.

[13] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," *IEEE International Conference on Intelligent Robots and Systems*, pp. 4906–4913, 2012.

[14] M. Posa and R. Tedrake, "Direct Trajectory Optimization of Rigid Body Dynamical Systems through Contact," in *Algorithmic Foundations of Robotics X*, (Berlin, Heidelberg), pp. 527–542, Springer Berlin Heidelberg, 2013.

[15] A. W. Winkler and P. Joslin, *Optimization-based motion planning for legged robots.* PhD thesis, ETH, Germany, 2018.

[16] T. Erez, Y. Tassa, and E. Todorov, "Simulation Tools for Model-Based Robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX," *International Conference on Robotics and Automation*, 2015.

[17] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," *International Conference on Intelligent Robots and Systems*, 2012.

[18] E. Todorov, "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 6054–6061, 2014.

[19] C. Jin, *A Sequential Process Monitoring Approach using Hidden Markov Model for Unobservable Process Drift.* PhD thesis, CyberInsight Tech. Co. Ltd., July 2015.

[20] A. Wächter and L. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[21] P. Gill, W. Murray, and M. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Review*, vol. 47, no. 1, pp. 99–131, 2005.

[22] FICO, "Xpress optimization." `https://www.fico.com/en/products/fico-xpress-optimization`.

[23] M. S. Andersen, J. Dahl, and L. Vandenberghe, "CVXOPT: A python package for convex optimization." Available at `http://cvxopt.org/`, 2012.

[24] J. Mattingley and S. Boyd, "Cvxgen: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.

[25] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control.* Springer, June 2016.

[26] H. Dai, A. Valenzua, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," *IEEE-RAS International Conference on Humanoid Robots*, vol. 2014, Nov. 2014.

[27] A. Carvalho and J. Martins, "Exact restitution and generalizations for the hunt-crossley contact model," *Mechanism and Machine Theory*, vol. 139, pp. 174–194, 2019.

[28] E. Todorov, "MuJoCo: Advanced Physics Simulator." `www.mujoco.org/index.html`.

[29] G. Schultz and K. Mombaur, "Modeling and Optimal Control of Human-Like Running," *Mechatronics, IEEE/ASME Transactions on*, vol. 15, pp. 783–792, Nov. 2010.

[30] D. Pardo, M. Neunert, A. Winkler, R. Grandia, and J. Buchli, "Hybrid direct collocation and control in the constraint-consistent subspace for dynamic legged robot locomotion," *Robotics: Sciend and Systems 2017*, vol. 2017, July 2017.

[31] A. Hereid, E. A. Cousineau, C. M. Hubicki, and A. D. Ames, "3d dynamic walking with underactuated humanoid robots: A direct collocation framework for optimizing hybrid zero dynamics," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1447–1454, May 2016.

[32] S. Caron. `https://scaron.info/teaching/friction-cones.html`.

[33] L. Liu and J. Hodgins, "Learning to schedule control fragments for physics-based characters using deep q-learning," *ACM Transactions on Graphics*, vol. 36, no. 3, 2017.

[34] B. Graf, "Quaternions and dynamics," 2008.

A - Quaternions

A.1 Quaternion derivative and angular velocity

[34] A quaternion is a four element structure to store a 3D orientation without ambiguity or susceptibility to singularities. Angular velocity and angular acceleration can be safely represented by x, y and z values, unlike orientation.

We denote a quaternion with $\mathbf{Q}$, not to be confused by the joint positions vector $\mathbf{q}$.

MuJoCo for instance uses quaternions to store 3D rotation, for the floating base and for any ball joints. Because of this, the number of position variables (q) can be bigger than the number of velocity variables (v).

Orientations and angular velocities can both be represented in global (inertial) frame or body-fixed (rotated) frame. We will use the body-fixed notation, because this corresponds to the use of joints in numerical models.

We can relate the quaternion time derivative to the angular velocity in body-fixed frame[2] as follows:

$$\dot{\mathbf{Q}} = \frac{1}{2}\mathbf{Q} \otimes \mathbf{\Omega} \tag{60}$$

Here $\otimes$ denotes the quaternion product and $\mathbf{\Omega}^T = [0, \omega_x, \omega_y, \omega_z]^T$

Working out the product yields the following matrix-vector multiplication:

$$\dot{\mathbf{Q}} = \frac{1}{2}\begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}\mathbf{Q} \tag{61}$$

$$= \frac{1}{2}\tilde{\mathbf{\Omega}}\mathbf{Q} \tag{62}$$

This skew matrix form of the angular velocity is comparable to how a vector cross product is identical to a matrix-vector multiplication of the cross skew form of the left vector.

The product can also be written as another matrix-vector product:

$$\dot{\mathbf{Q}} = \frac{1}{2}\begin{bmatrix} -Q_x & -Q_y & -Q_z \\ Q_w & -Q_z & Q_y \\ Q_z & Q_w & -Q_x \\ -Q_y & Q_x & Q_w \end{bmatrix}\boldsymbol{\omega} \tag{63}$$

$$= \frac{1}{2}\tilde{Q}_\omega \boldsymbol{\omega} \tag{64}$$

This makes the jacobians of the quaternion derivative which are needed by the optimizer easy to show:

$$\frac{\partial \dot{\mathbf{Q}}}{\partial \mathbf{Q}} = \frac{\partial\left(\frac{1}{2}\tilde{\Omega}\mathbf{Q}\right)}{\partial \mathbf{Q}} = \frac{1}{2}\tilde{\Omega} \tag{65}$$

$$\frac{\partial \dot{\mathbf{Q}}}{\partial \boldsymbol{\omega}} = \frac{\partial\left(\frac{1}{2}\tilde{Q}\boldsymbol{\omega}\right)}{\partial \boldsymbol{\omega}} = \frac{1}{2}\tilde{Q}_\omega \tag{66}$$

[2] In global frame the order is reversed: $\dot{\mathbf{Q}} = \frac{1}{2}\Omega \otimes \mathbf{Q}$

Eq. (60) can also be reversed to find the angular velocity from a quaternion derivative, again in body-fixed frame[3]:

$$\boldsymbol{\Omega} = 2\bar{Q} \otimes \dot{Q} \tag{67}$$

Here $\bar{Q}$ is the conjugate of Q (or the inverse if Q, because it is a unit quaternion): $\bar{Q} = (Q_w, -Q_x, -Q_y, -Q_z)$.

In the matrix form:

$$\boldsymbol{\Omega} = 2 \begin{bmatrix} Q_w & Q_x & Q_y & Qz \\ -Q_x & Q_w & Q_z & -Qy \\ -Q_y & -Q_z & Q_w & Q_x \\ -Q_z & Q_y & -Q_x & Q_w \end{bmatrix} \dot{Q} \tag{68}$$

Leaving out the first row to find ω instead of Ω:

$$\boldsymbol{\omega} = 2 \begin{bmatrix} -Q_x & Q_w & Q_z & -Qy \\ -Q_y & -Q_z & Q_w & Q_x \\ -Q_z & Q_y & -Q_x & Q_w \end{bmatrix} \dot{Q} \tag{69}$$

$$= 2\tilde{Q}_Q \dot{Q} \tag{70}$$

$$= 2\tilde{Q}_\omega^T \dot{Q} \tag{71}$$

Note that this skew matrix is simply the transpose of one showed earlier: $\tilde{Q}_\omega^T = \tilde{Q}_Q$

A.2 Quaternions in joint position

With a floating base and/or ball joints in the robot model quaternions will appear in the joint position. However, the joint velocity vector will contain angular velocities instead of quaternions derivatives. Hence there is a non-trivial conversion between joint rates $\dot{\mathbf{q}}$ and joint velocities $\mathbf{v}$:

$$\dot{\mathbf{q}} = \mathrm{B}(\mathbf{q})\mathbf{v} \tag{72}$$

For a system not involving quaternions $\mathrm{B}(\mathbf{q})$ would simply be unity. When quaternions are involved $\mathrm{B}(\mathbf{q})$ is rectangular and a combination of identity blocks and skew quaternion blocks.

To give an example to illustrate this:

$$\mathbf{q}^T = \begin{pmatrix} Q_{1,w} & Q_{1,x} & Q_{1,y} & Q_{1,z} & \theta_1 & \theta_2 & Q_{2,w} & Q_{2,x} & Q_{2,y} & Q_{2,z} \end{pmatrix} \tag{73}$$

$$\mathbf{v}^T = \begin{pmatrix} \omega_{1,x} & \omega_{1,y} & \omega_{1,z} & \dot{\theta}_1 & \dot{\theta}_2 & \omega_{2,x} & \omega_{2,y} & \omega_{2,z} \end{pmatrix} \tag{74}$$

Then:

$$\mathrm{B}(\mathbf{q}) = \begin{bmatrix} & & & 0 & 0 & 0 & 0 & 0 \\ & & & 0 & 0 & 0 & 0 & 0 \\ & \frac{1}{2}\tilde{Q_{1,\omega}} & & 0 & 0 & 0 & 0 & 0 \\ & & & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & & & \\ 0 & 0 & 0 & 0 & 0 & & & \\ 0 & 0 & 0 & 0 & 0 & & \frac{1}{2}\tilde{Q_{2,\omega}} & \\ 0 & 0 & 0 & 0 & 0 & & & \end{bmatrix} \tag{75}$$

[3]And again, in global frame the order would be reversed: $\boldsymbol{\Omega} = 2\dot{Q} \otimes \bar{Q}$

The relation can also be inverted:

$$\mathbf{v} = B^{-1}(\mathbf{q})\dot{\mathbf{q}} \tag{76}$$

Note that $B^{-1}(\mathbf{q})$ is not the matrix inverse of $B(\mathbf{q})$, which would not exist because it is not square when at least one quaternion is present. From Eq. (71) we can find a definition. To continue with the same example:

$$B^{-1}(\mathbf{q}) = \begin{bmatrix} & & & & 0 & 0 & 0 & 0 & 0 & 0 \\ & 2\tilde{Q_{1,Q}} & & & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & & & 2\tilde{Q_{2,Q}} & \\ 0 & 0 & 0 & 0 & 0 & 0 & & & & \end{bmatrix} \tag{77}$$

A.3 Geometric Jacobian

When quaternions are involved in the position vector $\dot{q} \neq v$. This also results in two notations for the geometric jacobian. On the one hand we have the jacobian based on joint rates (including quaternion derivatives):

$$\mathbf{p} = f(\mathbf{q}) \tag{78}$$

$$\dot{\mathbf{p}} = \frac{\partial f(\mathbf{q})}{\partial \mathbf{q}}\dot{\mathbf{q}} = \mathrm{J}_q(\mathbf{q})\dot{\mathbf{q}} \tag{79}$$

Whereas there is also the jacobian for the joint velocities (including angular velocities):

$$\dot{\mathbf{p}} = \mathrm{J}_v(\mathbf{q})\mathbf{v} \tag{80}$$

A relationship was already established for the joint rates $\dot{\mathbf{q}}$ and the joint velocities $\mathbf{v}$:

$$\dot{\mathbf{q}} = \mathrm{B}(\mathbf{q})\mathbf{v} \tag{81}$$

And so we find:

$$\dot{\mathbf{p}} = \mathrm{J}_v(\mathbf{q})\mathbf{v} = \mathrm{J}_q(\mathbf{q})\dot{\mathbf{q}} = \mathrm{J}_q(\mathbf{q})\mathrm{B}(\mathbf{q})\mathbf{v} = \mathrm{J}_v(\mathbf{q})B^{-1}(\mathbf{q})\dot{\mathbf{q}} \tag{82}$$

For $\mathbf{v} \neq 0$ and $\dot{\mathbf{q}} \neq 0$:

$$\mathrm{J}_v(\mathbf{q}) = \mathrm{J}_q(\mathbf{q})\mathrm{B}(\mathbf{q}) \tag{83}$$

$$\mathrm{J}_q(\mathbf{q}) = \mathrm{J}_v(\mathbf{q})\mathrm{B}^{-1}(\mathbf{q}) \tag{84}$$

Conventionally the joint rates are not as important and only the regular geometric jacobian J_v is used. However, our direct collocation NLP contains constraints based on end-effectors. The jacobians of these constraints rely (through the chain rule) on both J_q and J_q, hence the need for an explicit difference.

B - Inverse Kinematics

To create the initial guess for the optimization problem a basic inverse kinematics solver is needed. The IK problem can be solved iteratively using end-effector jacobians. Take a robot with joint coordinates $\mathbf{q}$, end-effector position $\mathbf{p}$ (Fig. 50a) and a desired end-effector position $\mathbf{r}$. We can relate the end-effector velocity to the joint velocities:

$$\dot{\mathbf{p}} = \mathrm{J}(\mathbf{q})\dot{\mathbf{q}} \tag{85}$$

Because the jacobian is not square in most cases, we cannot take the matrix inverse. That is, when the jacobian is not square there is not a unique combination of joint velocities resulting in the end-effector velocity. A solution can still be found using a pseudo inverse:

$$\dot{\mathbf{q}} = \mathrm{J}^{\dagger}(\mathbf{q})\dot{\mathbf{p}} \tag{86}$$

Here $\mathrm{J}^{\dagger}(\mathbf{q})$ is the Moore-Penrose inverse, which is effectively the solution to a least sum squares problem.

To solve the inverse kinematics problems we iteratively move the end-effector to the reference, calculating a desired end-effector velocity based on the current error:

```
while (norm(r - p) > threshold):
        error = r - p
        dp = error * eps
        dq = pseudoInverse(J) * dp
        q += dq
```

This works well for a robot with a fixed base, though less so for a robot with a floating base such as we encounter in the optimization problems because the entire robot will simply be displaced (see Fig. 50b). It can be improved by adding multiple end-effector references, including a reference to the base. In this way the base and end-effectors will all tend to their desired position.
This can be accomplished by steering multiple desired end-effector velocities:

$$\dot{\mathbf{p}}_1 = \mathrm{J}_1(\mathbf{q})\dot{\mathbf{q}} \tag{87}$$
$$\dot{\mathbf{p}}_2 = \mathrm{J}_2(\mathbf{q})\dot{\mathbf{q}} \tag{88}$$
$$\vdots$$

Which can be combined into a single pseudo inverse:

$$\dot{\mathbf{q}} = \begin{bmatrix} \mathrm{J}_1(\mathbf{q}) \\ \mathrm{J}_2(\mathbf{q}) \\ \vdots \end{bmatrix}^{\dagger} \begin{bmatrix} \dot{\mathbf{p}}_1 & \dot{\mathbf{p}}_2 & \dots \end{bmatrix} \tag{89}$$

And again the end-effector velocities are based on the error in position.

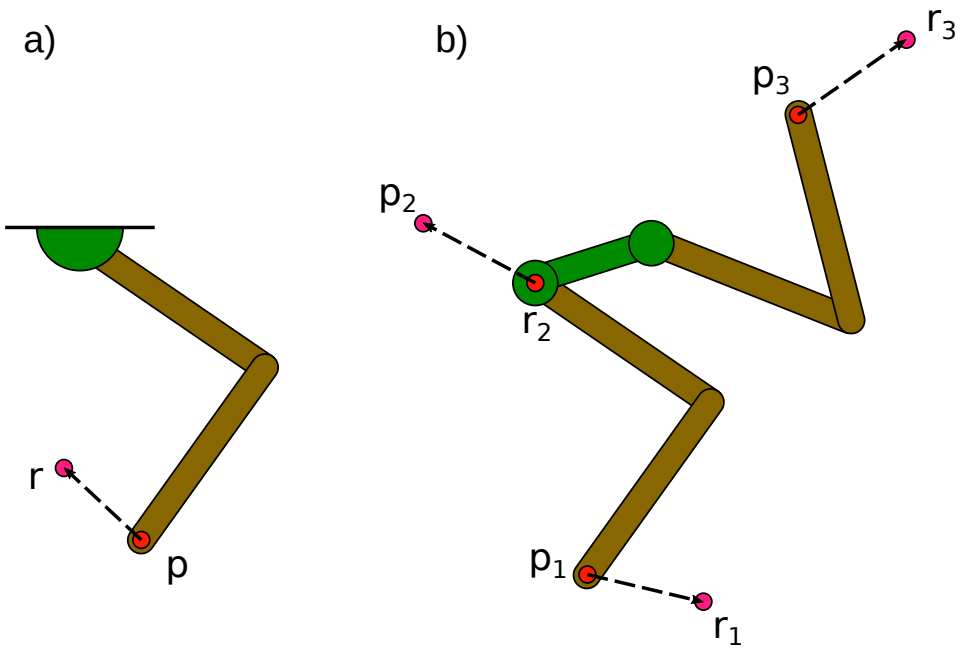

Figure 50: Inverse kinematics for a.) a fixed model (with a single reference) and b.) for a floating model (with multiple references).

B.1 MuJoCo

The above algorithm can be implemented in MuJoCo easily because the geometric jacobian of any body is readily available. These jacobians are already respective to the entire joint velocity space, requiring no model specific information to solve the IK problem.

Some attention should be paid to the integration of intermediate joint velocities to joint positions. The iterative velocity integration is then:

$$\mathbf{v} = \begin{bmatrix} \mathrm{J}_1(\mathbf{q}) \\ \mathrm{J}_2(\mathbf{q}) \\ \vdots \end{bmatrix}^{\dagger} \begin{bmatrix} \dot{\mathbf{p}}_1 & \dot{\mathbf{p}}_2 & \dots \end{bmatrix} \tag{90}$$

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \mathrm{B}(\mathbf{q}_k)\mathbf{v} \tag{91}$$

C - End-effector Equations of Motion

Equations of motion are typically described in joint space:

$$\mathrm{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathrm{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathrm{N}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\tau} \tag{92}$$

This can be rewriting to end-effector coordinates using:

$$\dot{\mathbf{p}} = \mathrm{J}(\mathbf{q})\dot{\mathbf{q}} \tag{93}$$

$$\ddot{\mathbf{p}} = \mathrm{J}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathrm{J}}(\mathbf{q})\dot{\mathbf{q}} \tag{94}$$

$$\boldsymbol{\tau} = \mathrm{J}^T(\mathbf{q})\mathbf{F} \tag{95}$$

Where $\mathrm{J}(\mathbf{q})$ is the end-effector jacobian.

The equations of motion can now be written in end-effector space:

$$J^T(\mathbf{q})\mathrm{M}(\mathbf{q})\ddot{\mathbf{q}} + J^T(\mathbf{q})\mathrm{C}(\mathbf{q}, \dot{\mathbf{q}}) + J^T(\mathbf{q})\mathrm{N}(\mathbf{q}, \dot{\mathbf{q}}) = J^T(\mathbf{q})\boldsymbol{\tau} \tag{96}$$

$$J^T(\mathbf{q})\mathrm{M}(\mathbf{q})\left(\mathrm{J}^{-1}(\mathbf{q})\ddot{\mathbf{p}} + \dot{\mathrm{J}}^{-1}(\mathbf{q})\dot{\mathbf{p}}\right) + J^T(\mathbf{q})\mathrm{C}(\mathbf{q}, \dot{\mathbf{q}}) + J^T(\mathbf{q})\mathrm{N}(\mathbf{q}, \dot{\mathbf{q}}) = J^T(\mathbf{q})\boldsymbol{\tau} \tag{97}$$

$$\left[J^T(\mathbf{q})\mathrm{M}(\mathbf{q})\mathrm{J}^{-1}(\mathbf{q})\right]\ddot{\mathbf{p}} + \left[J^T(\mathbf{q})\mathrm{M}(\mathbf{q})\dot{\mathrm{J}}^{-1}(\mathbf{q})\dot{\mathbf{p}} + J^T(\mathbf{q})\mathrm{C}(\mathbf{q}, \dot{\mathbf{q}})\right] + \left[J^T(\mathbf{q})\mathrm{N}(\mathbf{q}, \dot{\mathbf{q}})\right] = J^T(\mathbf{q})\boldsymbol{\tau} \tag{98}$$

$$\mathrm{M}_p(\mathbf{p})\ddot{\mathbf{p}} + \mathrm{C}_p(\mathbf{p}, \dot{\mathbf{p}}) + \mathrm{N}_p(\mathbf{p}, \dot{\mathbf{p}}) = J^T(\mathbf{p})\boldsymbol{\tau} + \mathbf{F} \tag{99}$$

D - Non-Linear Problem Formulation

D.1 Constraints

The $[k]$ is omitted where possible. All constraints are applied at each collocation point.

Integration Constraints ensure the velocities add up to the positions:

$$\mathbf{\Phi}[k] = \mathbf{q}[k] - \mathbf{q}[k+1] + \frac{\Delta t}{2}\left(\mathrm{B}_k \mathbf{v}[k] + \mathrm{B}_{k+1}\mathbf{v}[k+1]\right) \tag{100}$$

These obviously depend on $\mathbf{v}$:

$$\frac{\partial \mathbf{\Phi}[k]}{\partial \mathbf{v}[k]} = \frac{\Delta t}{2}\mathrm{B}_k \tag{101}$$

$$\frac{\partial \mathbf{\Phi}[k]}{\partial \mathbf{v}[k+1]} = \frac{\Delta t}{2}\mathrm{B}_{k+1} \tag{102}$$

$$\tag{103}$$

And depend on $\mathbf{q}$ directly and also indirectly through $\mathrm{B}(\mathbf{q})$:

$$\frac{\partial \mathbf{\Phi}[k]}{\partial \mathbf{q}[k]} = \mathrm{I} + \frac{\Delta t}{2}\frac{\partial(\mathrm{B}_k \mathbf{v}[k])}{\partial \mathbf{q}_k} \tag{104}$$

$$\frac{\partial \mathbf{\Phi}[k]}{\partial \mathbf{q}[k+1]} = -\mathrm{I} + \frac{\Delta t}{2}\frac{\partial(\mathrm{B}_{k+1} \mathbf{v}[k+1])}{\partial \mathbf{q}_{k+1}} \tag{105}$$

The derivatives $\frac{\partial(\mathrm{B}(\mathbf{q})\mathbf{v}}{\partial \mathbf{q}}$ can be assembled from Eq. (65).

Dynamics Constraints make sure the velocities match with torque and reaction forces:

$$\mathbf{\Phi}[k] = \mathbf{v}[k] - \mathbf{v}[k+1] + \frac{\Delta t}{2}(f_k + f_{k+1}) \tag{106}$$

With $f_k = f(\mathbf{q}[k], \mathbf{v}[k], \tau[k], \mathbf{F}[k])$. This makes the constraint dependent on all variables:

$$\frac{\partial \mathbf{\Phi}[k]}{\partial \mathbf{q}[k]} = \frac{\Delta t}{2}\frac{\partial f_k}{\partial \mathbf{q}[k]} \tag{107}$$

$$\frac{\partial \mathbf{\Phi}[k]}{\partial \mathbf{v}[k]} = \mathrm{I} + \frac{\Delta t}{2}\frac{\partial f_k}{\partial \mathbf{v}[k]} \tag{108}$$

$$\frac{\partial \mathbf{\Phi}[k]}{\partial \boldsymbol{\tau}[k]} = \frac{\Delta t}{2}\frac{\partial f_k}{\partial \boldsymbol{\tau}[k]} \tag{109}$$

$$\frac{\partial \mathbf{\Phi}[k]}{\partial \mathbf{F}[k]} = \frac{\Delta t}{2}\frac{\partial f_k}{\partial \mathbf{F}[k]} \tag{110}$$

And analogously for $k+1$.
The partial derivatives of the dynamics are determined numerically, like:

$$\frac{\partial f_k}{\partial \mathbf{q}[k]} \approx \frac{f(\mathbf{q}+\mathbf{h}, ...) - f(\mathbf{q}, ...)}{\mathbf{h}} \tag{111}$$

Quaternions Constraints are needed to keep the quaternion in the joint position (if there are any) to unity length:

$$\mathbf{\Phi}[k] = \begin{pmatrix} || \begin{pmatrix} \mathbf{q}_i[k] & \mathbf{q}_{i+1}[k] & \mathbf{q}_{i+2}[k] & \mathbf{q}_{i+3}[k] \end{pmatrix} || - 1 \\ \vdots \end{pmatrix} \tag{112}$$

This is only dependent on the joint positions:

$$\frac{\partial \mathbf{\Phi}[k]}{\partial \mathbf{q}[k]} = \begin{bmatrix} \cdots & 0 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 2q_i[k] & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 2q_{i+1}[k] & 0 & \cdots \\ & & \vdots & & & \end{bmatrix} \tag{113}$$

Terrain Constraints keep the end-effectors on the surface of the terrain without slip. The vertical position of an end-effector is a function of the joint positions. The height of the terrain is dependent on the x and y location, which in turn are also determined by the generalized coordinates:

$$\mathbf{\Phi}[k] = \begin{pmatrix} z_{ee}[k] - z_{terrain}[k] \\ x_{ee}[k] - x_{ee}[k-1] \\ y_{ee}[k] - y_{ee}[k-1] \end{pmatrix} = \begin{pmatrix} z_{ee}(\mathbf{q}[k]) - z_{terrain}(x_{ee}(\mathbf{q}[k]), y_{ee}(\mathbf{q}[k])) \\ x_{ee}(\mathbf{q}[k]) - x_{ee}(\mathbf{q}[k-1]) \\ y_{ee}(\mathbf{q}[k]) - y_{ee}(\mathbf{q}[k-1]) \end{pmatrix} \tag{114}$$

The jacobian is only non-zero with respect to $\bar{\mathbf{q}}$:

$$\frac{\partial \Phi_{k,1}}{\partial \mathbf{q}} = \frac{\partial z_{ee}(\mathbf{q})}{\partial \mathbf{q}} - \frac{\partial z_{terrain}(x_{ee}(\mathbf{q}), y_{ee}(\mathbf{q}))}{\partial \mathbf{q}} \tag{115}$$

$$= \mathrm{J}_{ee,z}(\mathbf{q}) - \frac{\partial z_{terrain}}{\partial x_{ee}} \frac{\partial x_{ee}}{\partial \mathbf{q}} - \frac{\partial z_{terrain}}{\partial y_{ee}} \frac{\partial y_{ee}}{\partial \mathbf{q}} \tag{116}$$

$$= \mathrm{J}_{ee,z}(\mathbf{q}) - \frac{\partial z_{terrain}}{\partial x_{ee}} \mathrm{J}_{ee,x}(\mathbf{q}) - \frac{\partial z_{terrain}}{\partial y_{ee}} \mathrm{J}_{ee,y}(\mathbf{q}) \tag{117}$$

$$\tag{118}$$

In Eq. (116) the total derivative was used.

$$\frac{\partial \Phi_{k,2}}{\partial \bar{\mathbf{q}}} = \mathrm{J}_{ee,x}(\mathbf{q}[k]) \frac{\partial \mathbf{q}[k]}{\partial \bar{\mathbf{q}}} - \mathrm{J}_{ee,x}(\mathbf{q}[k-1]) \frac{\partial \mathbf{q}[k-1]}{\partial \bar{\mathbf{q}}} \tag{119}$$

And $\frac{\partial \Phi_{k,3}}{\partial \bar{\mathbf{q}}}$ is analogous to this.

Force Constraints are used to keep the reaction forces zero during flight and within a friction cone during stance:

$$\mathbf{\Phi}[k] = \begin{pmatrix} \mathbf{F} \cdot \mathbf{n} \\ \mathbf{F} \cdot (\mathbf{t}_1 - \mu \mathbf{n}) \\ \mathbf{F} \cdot (\mathbf{t}_1 + \mu \mathbf{n}) \\ \mathbf{F} \cdot (\mathbf{t}_2 - \mu \mathbf{n}) \\ \mathbf{F} \cdot (\mathbf{t}_2 + \mu \mathbf{n}) \end{pmatrix} \tag{120}$$

Note that the terrain height, and therefore also the terrain tangents and normal, are dependent on the position:

$$\mathbf{n} = \mathbf{n}(x_{ee}, y_{ee}) = \mathbf{n}(x_{ee}(\mathbf{q}), y_{ee}(\mathbf{q})) \tag{121}$$

$$\mathbf{t} = \mathbf{t}(x_{ee}, y_{ee}) = \mathbf{t}(x_{ee}(\mathbf{q}), y_{ee}(\mathbf{q})) \tag{122}$$

$$\tag{123}$$

The jacobian is clearly non-zero for the reaction forces:

$$\frac{\partial \mathbf{\Phi}[k]}{\partial \mathbf{F}} = \begin{bmatrix} \mathbf{n}^T \\ (\mathbf{t}_1 - \mu\mathbf{n})^T \\ \vdots \\ (\mathbf{t}_2 + \mu\mathbf{n})^T \end{bmatrix} \tag{124}$$

And also non-zero for joint positions:

$$\frac{\partial \Phi_{k,1}}{\partial \mathbf{q}} = \frac{\partial (\mathbf{F}\cdot\mathbf{n})^T}{\partial \mathbf{q}} = \frac{\partial (\mathbf{F}\cdot\mathbf{n})^T}{\partial \mathbf{n}} \frac{\partial \mathbf{n}}{\partial \mathbf{q}} = \mathbf{F}^T \frac{\partial \mathbf{n}}{\partial \mathbf{p}_{ee}} \frac{\partial \mathbf{p}_{ee}}{\partial \mathbf{q}} \tag{125}$$

$$= \mathbf{F}^T \frac{\partial \mathbf{n}}{\partial \mathbf{p}_{ee}} \mathrm{J}_{ee}(\mathbf{q}) \tag{126}$$

Here $\mathbf{p}_e e = (x_{ee}, y_{ee}, z_{ee})^T$. The normal and tangents are constant for z_{ee}.

$$\frac{\partial \Phi_{k,2}}{\partial \mathbf{q}} = \frac{\partial (\mathbf{F}\cdot\mathbf{t}_1 - \mathbf{F}\cdot\mu\mathbf{n})^T}{\partial \mathbf{q}} \tag{127}$$

$$= \mathbf{F}^T \left(\frac{\partial \mathbf{t}_1}{\partial \mathbf{p}_{ee}} \mathrm{J}_{ee}(\mathbf{q}) - \mu \frac{\partial \mathbf{n}}{\partial \mathbf{p}_{ee}} \mathrm{J}_{ee}(\mathbf{q}) \right) \tag{128}$$

$$= \mathbf{F}^T \left(\frac{\partial \mathbf{t}_1}{\partial \mathbf{p}_{ee}} - \mu \frac{\partial \mathbf{n}}{\partial \mathbf{p}_{ee}} \right) \mathrm{J}_{ee}(\mathbf{q}) \tag{129}$$

Analogously for $\Phi_{k,3}$ to $\Phi_{k,5}$:

$$\frac{\partial \mathbf{\Phi}[k]}{\partial \mathbf{q}} = \begin{bmatrix} \mathbf{F}^T \left(\frac{\partial \mathbf{n}}{\partial \mathbf{p}_{ee}} \right) \\ \mathbf{F}^T \left(\frac{\partial \mathbf{t}_1}{\partial \mathbf{p}_{ee}} - \mu \frac{\partial \mathbf{n}}{\partial \mathbf{p}_{ee}} \right) \\ \mathbf{F}^T \left(\frac{\partial \mathbf{t}_1}{\partial \mathbf{p}_{ee}} + \mu \frac{\partial \mathbf{n}}{\partial \mathbf{p}_{ee}} \right) \\ \mathbf{F}^T \left(\frac{\partial \mathbf{t}_2}{\partial \mathbf{p}_{ee}} - \mu \frac{\partial \mathbf{n}}{\partial \mathbf{p}_{ee}} \right) \\ \mathbf{F}^T \left(\frac{\partial \mathbf{t}_2}{\partial \mathbf{p}_{ee}} + \mu \frac{\partial \mathbf{n}}{\partial \mathbf{p}_{ee}} \right) \end{bmatrix} \mathrm{J}_{ee}(\mathbf{q}) \tag{130}$$

E - UML

Below are UML diagrams of key components of the code of the new framework.

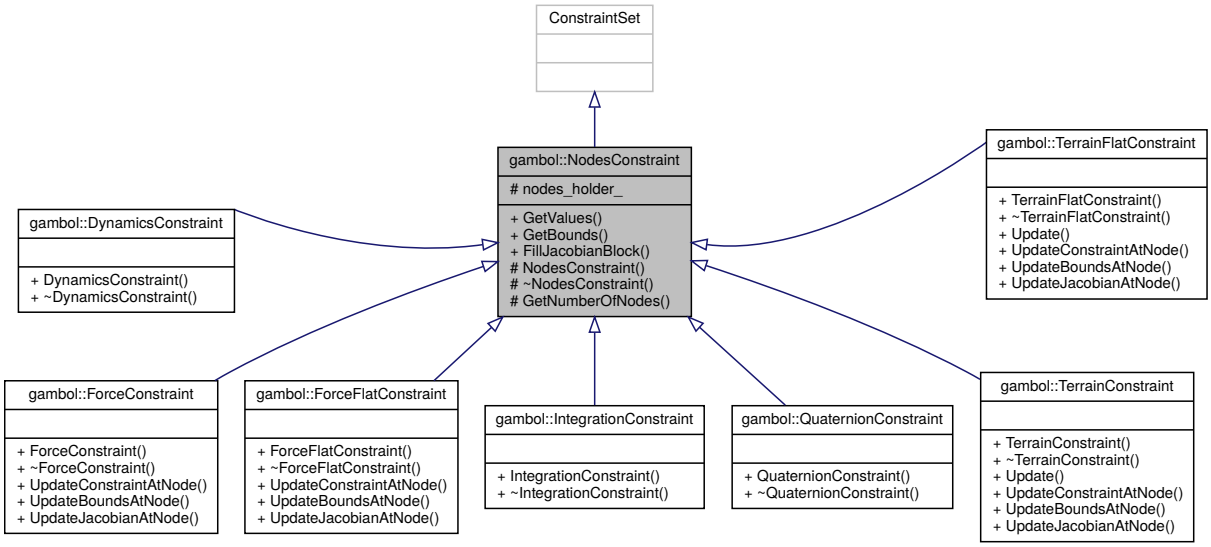

Figure 51: UML inheritance diagram of the different constraints. `ConstraintSet` is part of IFOPT.

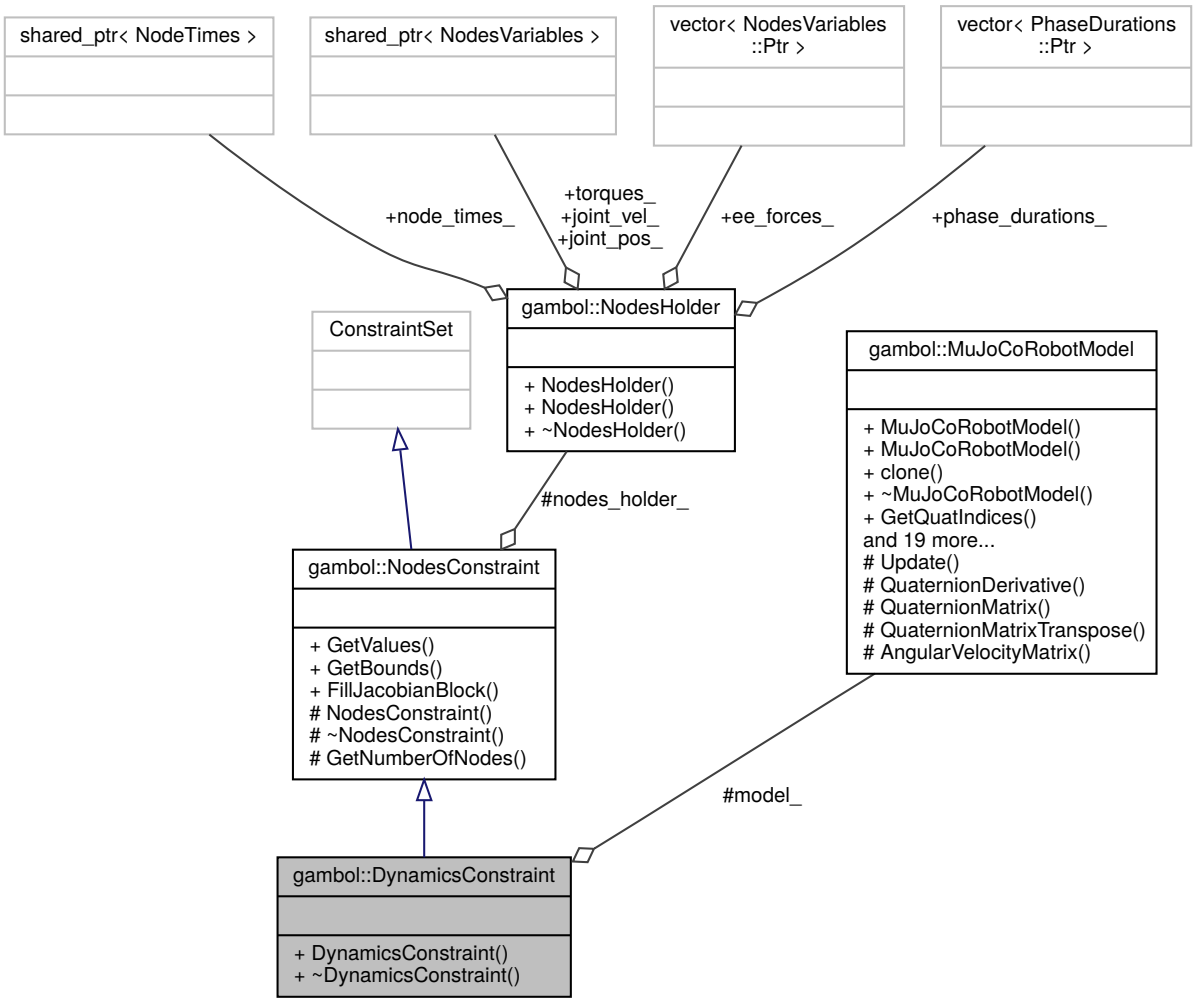

Figure 52: UML collaboration diagram of the dynamics constraint. The `NodesHolder` struct is a collection of pointers to all variables. It is used to easily combine variables inside the costs and constraints.

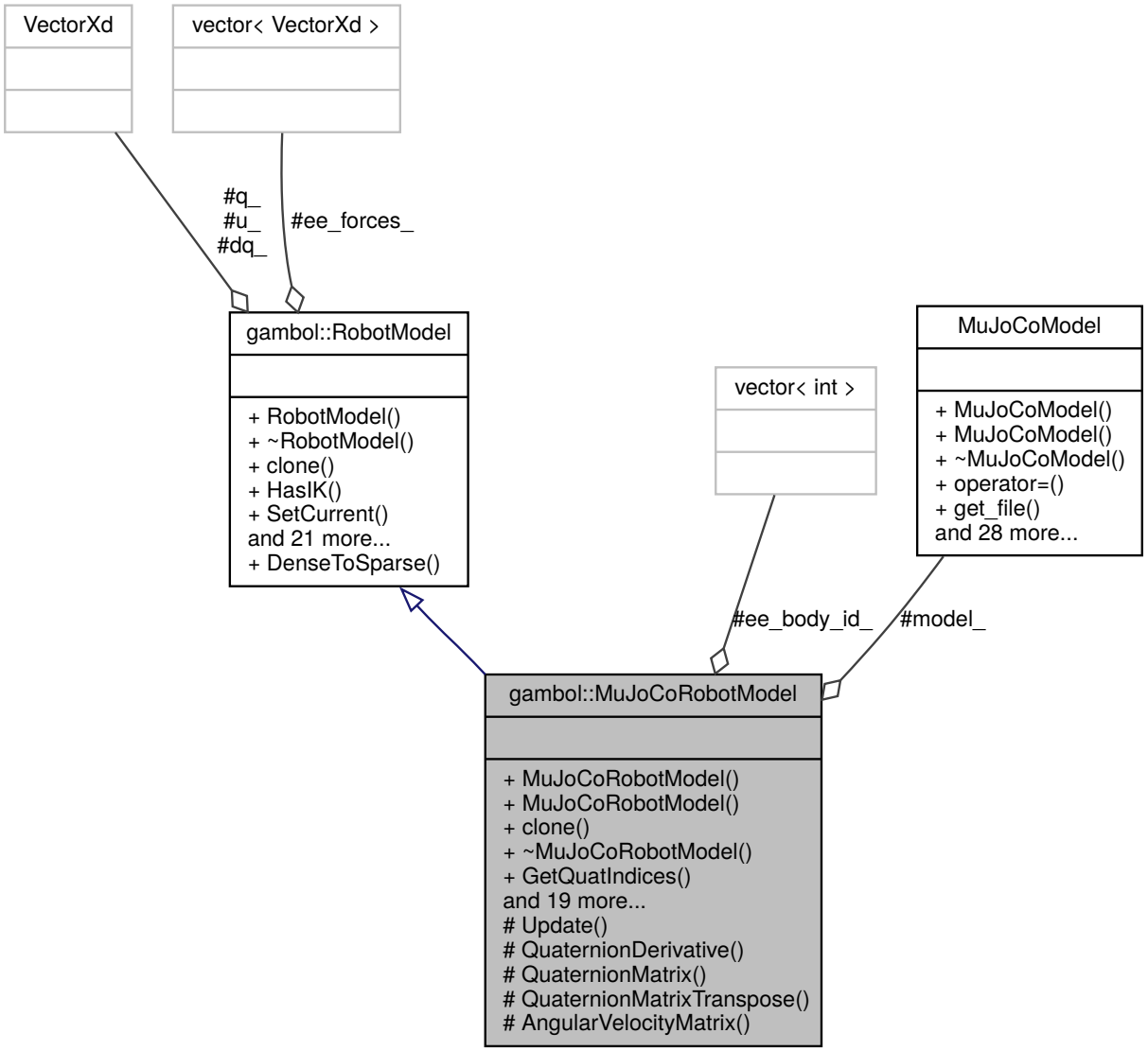

Figure 53: UML collaboration diagram of the general `RobotModel` and the MuJoCo specific implementation. The class `MuJoCoModel` is an object oriented wrapper for the MuJoCo function and `MuJoCoRobotModel` is simply an interface between the two. A new dynamics implementation would extend `RobotModel` and override the pure virtual methods.

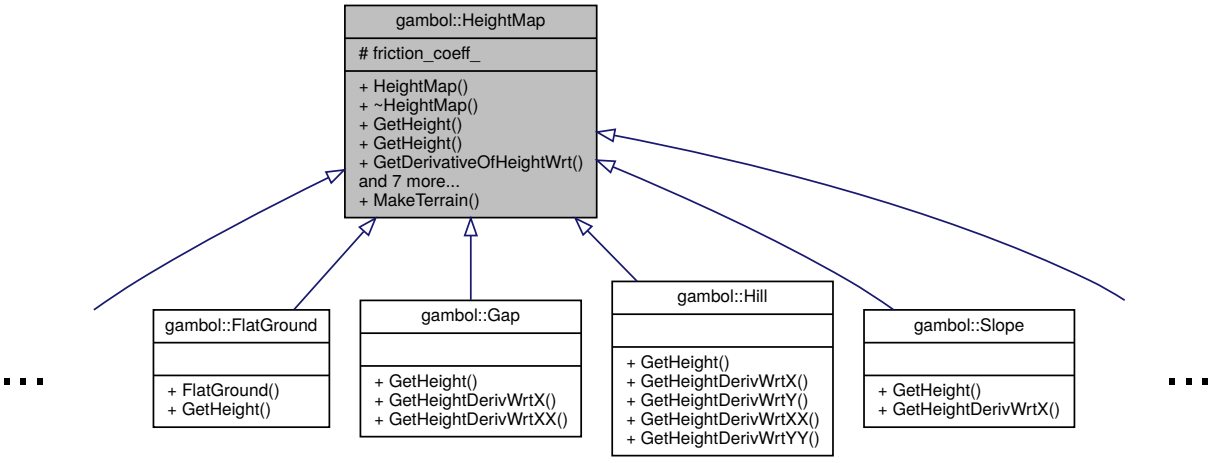

Figure 54: UML collaboration diagram of the abstract height-map class and several implementations. An instance of the a height-map class provides terrain information.

F - Additional Simulations

Figures 55 and 56 show a simulation of the biped in 3D with the optimized forces applied directly instead of simulated contact. Figures 57 and 58 show a similar simulation for the quadruped.
For both simulations the optimizations were not repeated. The results are much worse if they are, because the repeated optimizations fail due to how the end-effectors are free to penetrate the ground with the lack of collision detection.

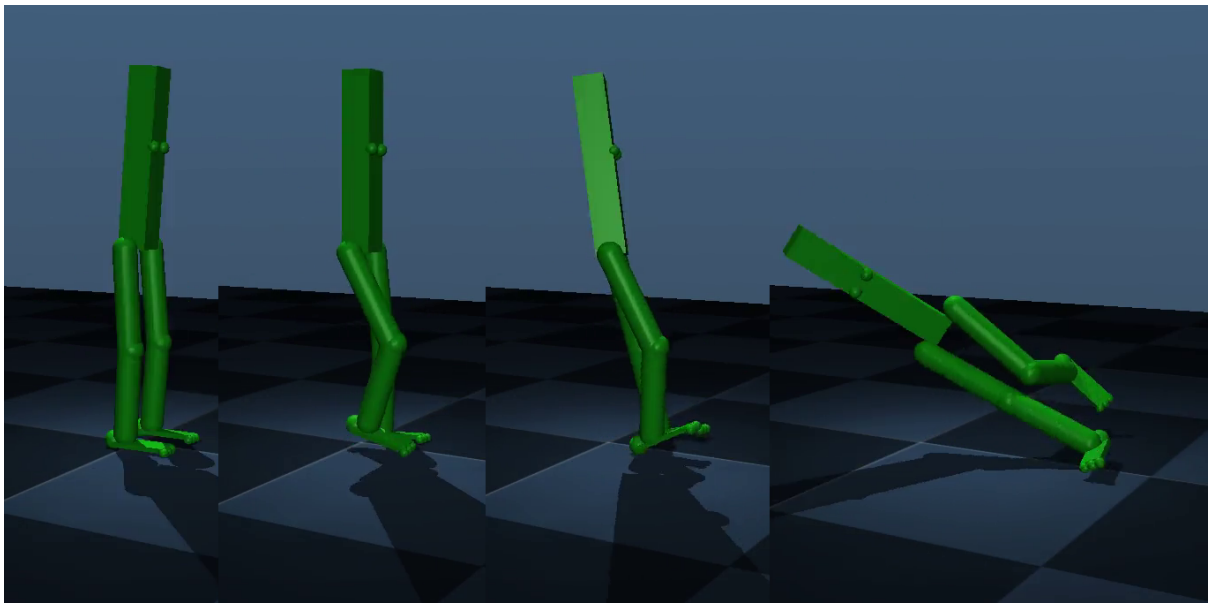

Figure 55: Simulation of the biped in 3D including heels and toes without real contacts. Instead the optimized forces are applied directly. The optimization was not repeated, $N = 50$. The legs slip away from the robot almost immediately

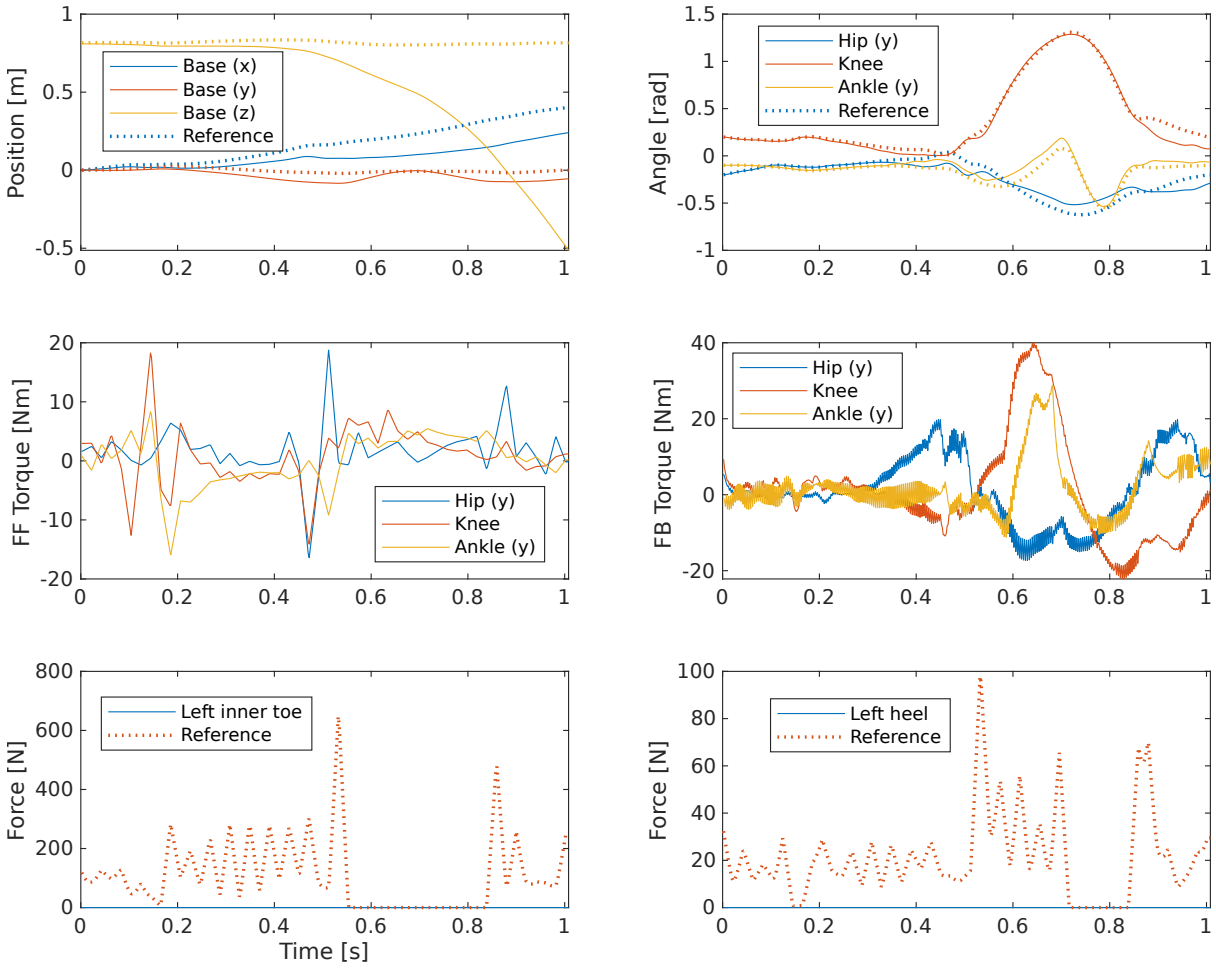

Figure 56: Graphs corresponding to the snapshots in Fig. 55. The optimized references of the body are tracked poorly, the robot also falls over quickly.

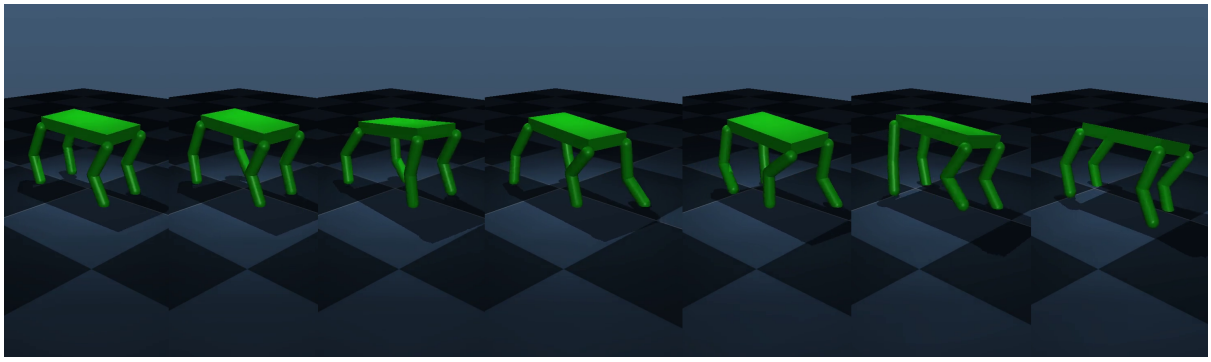

Figure 57: Simulation of the quadruped in 3D without real contacts. Instead the optimized forces are applied directly. The optimization was not repeated, $N = 50$. In the last moments of the simulation the legs slip from underneath the robot.

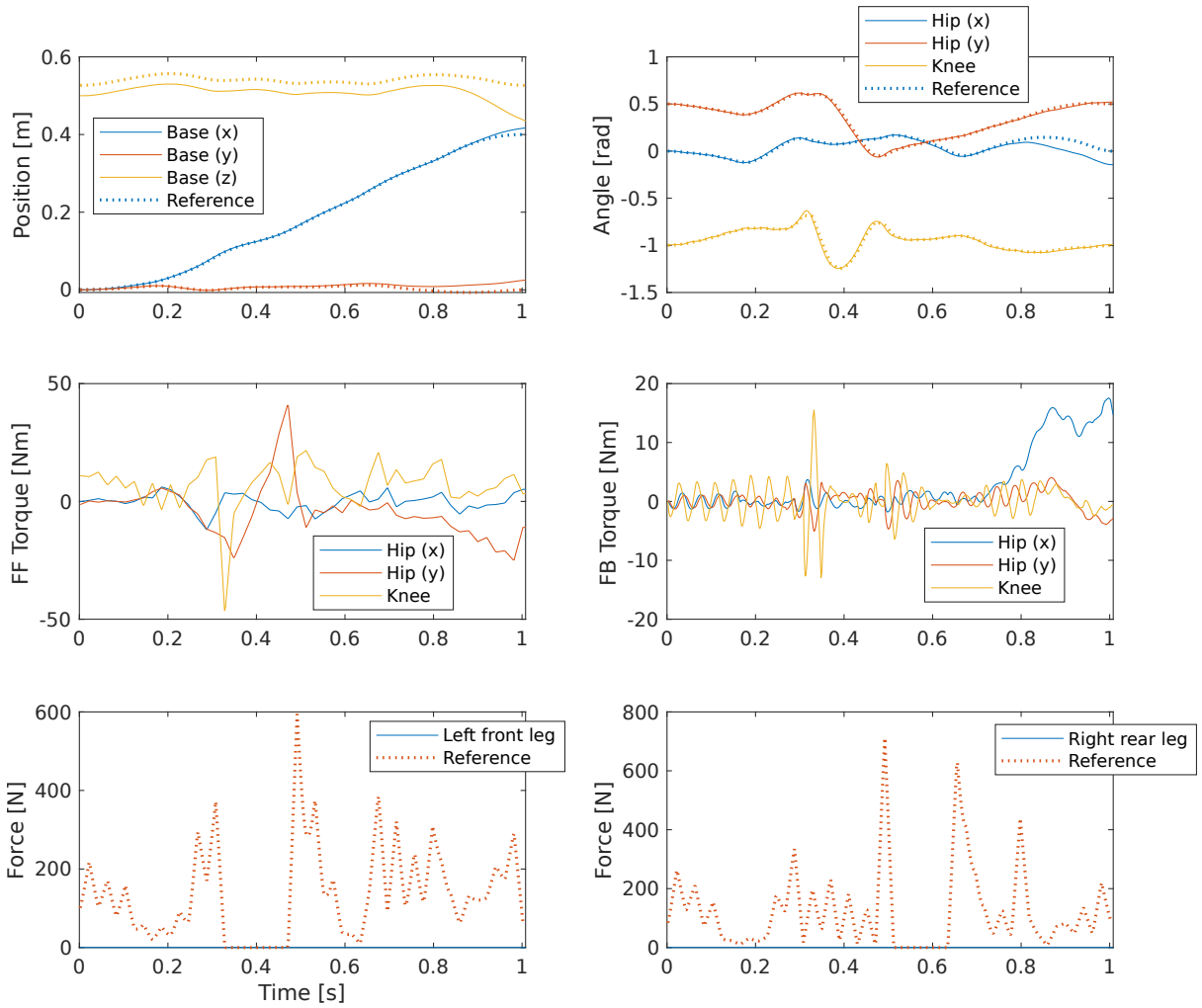

Figure 58: Graphs corresponding to the snapshots in Fig. 57. The optimized references are being tracked well. In the last 0.1 seconds the errors become larger, when the legs start to slip.