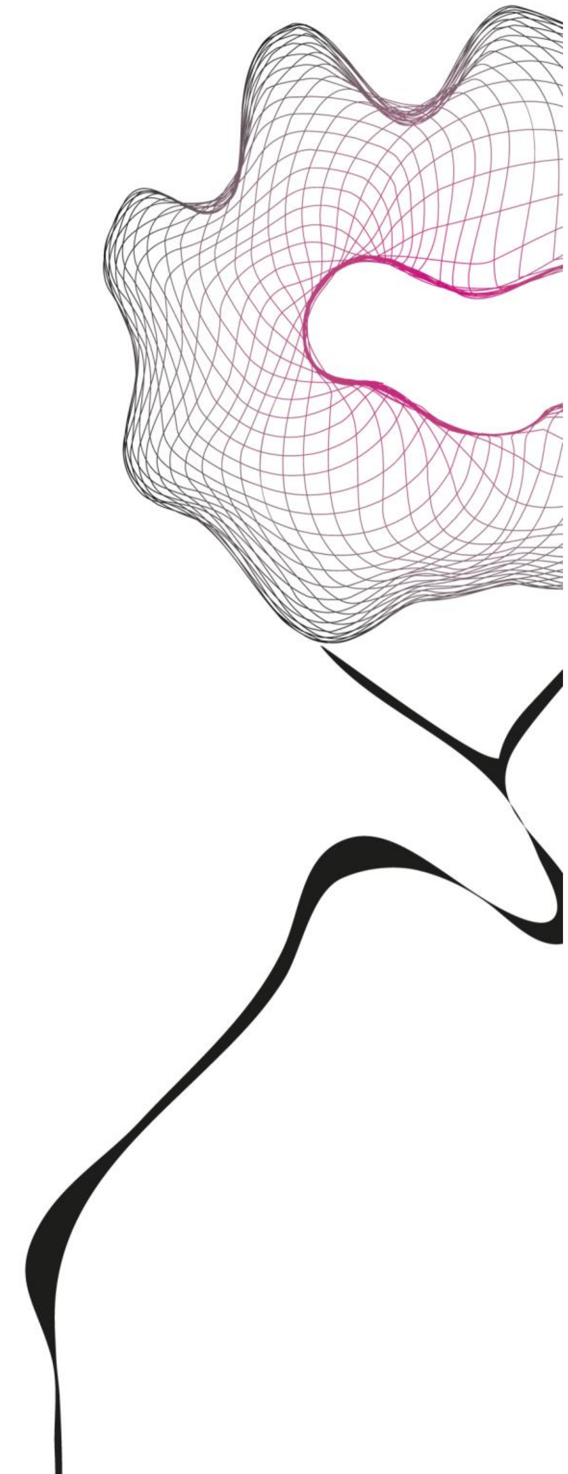
MASTER THESIS



DESIGN OF A CONTROL STRATEGY FOR OBSTACLE CROSSING IN A LOWER LIMB EXOSKELETON FOR SCI PATIENTS

L.M.E. van Opheusden

FACULTY OF ENGINEERING TECHNOLOGY
DEPARTMENT OF BIOMECHANICAL ENGINEERING

DOCUMENT NUMBER
BW - 596

UNIVERSITY OF TWENTE.

15 DECEMBER 2017

Contents

1. Introduction

Every year, tens of thousands of people worldwide are struck by the disastrous effects of spinal cord injury (SCI) [1]. Due to a lesion in the spinal cord, many patients suffering from SCI are paraplegic, making them unable to walk. A person's quality of life is determined greatly by their self-sufficiency. Loss of mobility makes SCI patients very dependent on their environment and the health care system, which in turn greatly diminishes their quality of life. In recent years, scientists have put great effort into curing and treating SCI [2]. However, many SCI patients remain confined to a wheelchair after their rehabilitation period. Recent developments in the field of robotics have given birth to a new range of assistive devices. Through the use of powered exoskeletons, patients with SCI can be given back the ability to walk independently.
The term "exoskeleton" in this context is defined as a mechanical device that fits around the human body and moves in parallel with it, as if the patient is wearing a mechanical suit [3]. Powered exoskeletons distinguish themselves by the fact that they can, through the use of active elements, provide a net effort to the wearer. They can be used to augment the physical capabilities of an able-bodied person, like the BLEEX exoskeleton (Figure 1.1a), or to provide assistance to patients with a limb pathology, like the Ekso (Figure 1.1b).

Even though powered exoskeletons carry the promise to become invaluable assistive tools, the current state of the art provides people suffering from SCI with limited functionality. The devices are tailored to flat grounds or shallow slopes, which hinders the wearer to move freely outside of a laboratory setting. In order to help SCI patients regain their mobility and self-sufficiency, exoskeleton technology must be taken out of the laboratory environment and into daily life scenarios. Our natural environment is highly unstructured terrain, as it is filled with obstacles, such as doorsteps, curbs and stairs. Powered exoskeletons are mechanically capable of overcoming these obstacles, but the design of their controllers falls short in this regard. Therefore, there is a need to improve upon currently available control strategies, to help SCI patients function in our unpredictable and three-dimensional world. *The aim of this research is to design a control strategy that allows the wearer of an exoskeleton to step over an obstacle.* This makes it possible for exoskeleton users to traverse doorsteps and uneven terrain and will allow the exoskeleton to help its wearer to successfully navigate daily life.

(a) BLEEX

(b) Ekso

Figure 1.1.

1.1. Powered Exoskeletons for Spinal Cord Injury

There are several exoskeletons available on the market that are designed specifically with people suffering from the effects of SCI in mind. An extensive overview of the state of the art can be found in one of many outstanding review articles [4–6]. In this section, we will consider the most relevant devices and create an overview of their capabilities, see Table 1.1.

Most exoskeletons provide powered hip and knee flexion and extension using a pre-recorded gait pattern. Ankle actuation is usually performed passively using a spring, but the Indego is designed to work alongside a stand-alone ankle foot orthosis (AFO). Joint angles are controlled using a PD controller. A major drawback of many of these devices is that the wearer requires a stabilizing tool, such as crutches or a walker. Moreover, by using a pre-recorded gait trajectory, usage of the device is constrained to known and controlled environments.

EKSO [7, 8] Ekso Bionics created the EKSO device for SCI patients, based on the eLEGS platform. It is hard to find information of the current state of affairs the EKSO, since the device is being produced commercially. The eLEGS platform, however, is well documented. The exoskeleton supports powered flexion and extension of the hip and knee, as well as spring powered actuation of the ankle in the sagittal plane. Wearing the EKSO enables one to perform flat ground walking, sitting to to standing (STS) transitions and making turns.

Indego [9] The Indego was developed at the Vanderbilt University and has since been commercialized by Parker Hannifin Corp. Because of this, there are few sources covering the Indego, but design and control of its predecessor, the Vanderbilt exoskeleton, is reported extensively in literature. It consists of actuated hip and knee joints that allow flexion and extension. The ankle joint is meant to be actuated using a stand-alone AFO. Balance is kept using forearm crutches. The exoskeleton enables walking, STS transitions and stair ascent and descent. A PD controller enforces pre-defined joint angle trajectories that have been recorded from a healthy user wearing the device.

Hybrid Assistive Limb (HAL) [10] The HAL is a commercially available exoskeleton which is marketed by Cyberdyne Inc. for subjects with incomplete SCI or impaired motor function. It provides powered support on the sagittal plane in the hip, knee and ankle joints. HAL supports walking and STS transitions, but requires support in the form of a walker or a parallel bar for balance. The hip and knee are actuated using a PD controller, according to a gait pattern that was recorded from a healthy subject. The ankle is actuated using a spring.

ReWalk [11] ReWalk is a commercially available exoskeleton designed to assist thoracic-level SCI patients in activities of daily life. It includes powered hip and knee flexion and extension and a spring-assisted ankle. The joint angles are controlled to follow a predefined gait pattern. The device can perform straight-walking, STS transitions and stair walking. Using the ReWalk requires a walking aid, such as crutches or a walker.

Mina [12] Mina has only two degrees of freedom on each leg: hip flexion/extension and knee flexion/extenstion. The ankles are connected using a rigid joint, constraining its movement. Crutches are needed to maintain balance. The actuators in the hips and knees are controlled using a PD controller and prerecorded patterns obtained from healthy subjects walking in the device. Because ankle movement is constrained in the Mina, the recorded gait is similar to walking on a slippery surface

ATLAS [13] ATLAS is an orthosis designed for a child with quadriplegia, who cannot control her torso. The device provides active actuation of the knee and hip in the sagittal plane. The ankle is connected to the knee and hip using a cable, which results in a synergetic action in the ankle joint. Ambulation is performed fully autonomous by the exoskeleton, as the wearer only provides very high-level input, such as start-stop signals. The system works in a walking frame to ensure stability. The impedance controller uses prerecorded and then parametrized joint trajectories as reference, and features variable stiffness according to the gait phase. In swing, stiffness is high in order to follow the trajectory closely and in stance, the leg is more compliant to account for any perturbations. The trajectory parameters, such as ground clearance or step length, can be changed real-time, so that the system can adapt to the needs of the wearer.

MINDWALKER [14, 15] MINDWALKER separates itself from the others with its 5 degrees of freedom on each leg, of which three are actuated. Besides movement in the sagittal plane, MINDWALKER also performs active hip ad- and abduction (HAA) and

	Flat Ground	STS	Stairs	Turning
Ekso	x	x	x	x
Indego	x	x	x	
HAL	x	x		
ReWalk	x	x	x	
Mina	x			
ATLAS	x			
MINDWALKER	x			

Table 1.1.: Exoskeletons designed for SCI patients and the tasks they reportly support.

allows passive rotation of the hip. Patients need crutches or a rail to walk with MINDWALKER. The device is impedance controlled around prerecorded patterns obtained from able-bodied subjects in the sagittal plane. Wang et al. present a model-predictive controller that generates trajectories for flat ground walking for the MINDWALKER.

1.2. Control Strategies for Obstacle Crossing

In this section, we provide an overview of possible control strategies for obstacle crossing in exoskeletons. To the best of our knowledge, none of the exoskeletons shown in the previous sections can perform the complex task of obstacle avoidance. Therefore, we also draw inspiration from the nearby field of the bipedal robots.

One of the most popular strategies to control bipedal walkers, especially in the early days, was to generate a joint trajectory at low frequency, which is then tracked using a PD controller [5, 8, 10]. Exoskeletons mainly use trajectories recorded from healthy subjects walking in the device as reference for the PD controllers, where humanoids generally base their trajectories on an optimization function. A big advantage of this scheme is that it requires very little computational power. However, because the reference patterns are fixed, these controllers generally lack the robustness to perform outside a laboratory. Moreover, using a controller like this neglects any passive dynamics of the system and may therefore by energetically very unfavorable. Finally, for a complex scenario like obstacle avoidance, a single reference trajectory does not suffice, but a large database is necessary or a method to modulate a standard pattern must be designed.

The neuromuscular controller (NMC) presented by Thatte and Geyer is based on very basic muscle interactions in the human body [16]. They make use of a model that includes nine lower extremity muscles, which are all modeled using a simple model that relates muscle force to its length, velocity and activation. Then reflex loops relating the activation of a certain muscle to the other muscles are defined. The parameter set defining these interactions is then optimized by minimizing the error in the landing leg angle and the joint torques. The resulting system can generate joint trajectories that resemble human walking without needing central processing. Using this method, they produce simulations of a bipedal walker traversing very rough terrain. This method requires a lot of computation time to solve the initial optimization problem. After a parameter set is found, joint torque trajectories are generated very quickly, which makes it very promising as a control strategy for obstacle crossing. However, the method is not endpoint related and thus does not guarantee avoidance of the obstacle. For this reason, NMC is not regarded suitable for our application.

The generally accepted method to control bipedal robots is using an optimization based scheme, through which they are able to perform very complex tasks, such as obstacle crossing [17–22]. Optimization based controllers generally use a two tiered system. At low frequency, the environment is scanned and processed to find optimal foot placement targets a few steps ahead, taking into account changes in step length, velocity and direction. Then for each step, a trajectory generator is called that computes the most advantageous trajectory that ends in the given end position of the swing foot. This is done by minimizing a cost function that weighs energy expenditure over performance. Then for every step, the solution space is scanned for the trajectory with the minimal cost. An advantage of this method is that it finds the best possible trajectory, which may increase performance. However, optimization is computationally speaking very expensive.
Most of the controllers shown generate motion patterns once per step and track these using a PD controller. The advantage of this is that the computationally expensive task of trajectory optimization can be performed at a low frequency. However, by designing a trajectory for the entire step and tracking this, we become vulnerable to errors in the case that the environment changes. For laboratory settings, where the environment is relatively controlled, such a method suffices. For use outside, especially when we are continuously gathering new intelligence about the surroundings of the walker through a machine vision algorithm, we need to be able to adapt the trajectory during a step.
In model predictive control (MPC) the current state of a system is used with a model to predict the future behavior of the system within a finite horizon. Using this information, control action is determined using an optimization scheme. At every time step, the optimization is re-iterated to account for changes in the environment and tracking errors. Model predictive control has been applied in both humanoid robots and exoskeletons. An advantage of MPC is that it explicitly handles constraints in the optimization step. Moreover,it makes explicit use of the dynamic model of the system. As a result, the performance of the controller is directly dependent on the quality of the model. Because the MPC continually updates its trajectory, it is very robust to changes in the environment. The main obstacle in MPC is long computation times, but the advent of faster processors makes this less of a problem than before.
Wang et al present a MPC tailored for the LOPES exoskeleton that generates swing leg trajectories [15]. In this paper, a strategy that needs only basic gait descriptors, such as step length and swing duration as input is shown to be successful for normal walking. In this research, the swing leg trajectory is designed. This may be suitable for treadmill-based exoskeletons, but for a standalone device, we expect that a stance leg strategy is required. Therefore, this method should be extended to incorporate both

stance and swing legs for it to be feasible for obstacle crossing.
Work done by Kalamian et al. shows an obstacle crossing strategy for a five DOF bipedal robot, based on a non-linear MPC. In contrast to other methods, they not only minimize energy consumption, but maximize the average velocity of the center of mass (COM). The internal model used for the optimization is collection of multilayered perceptron neural networks, one for the joint angle and one for the angular velocity of each joint. The models are trained offline using data collected from the robot. The method controls both single and double support phases and allows the robot to cross over an obstacle that is 40cm tall and 15cm long.
Taking all observations in this section into consideration, we conclude that model predictive control is the most promising strategy to perform obstacle crossing with an exoskeleton.

1.3. Research Goal

The aim of this research is to design an MPC that generates a movement trajectory that allows the wearer of an exoskeleton to step over an obstacle. Because of the high dimensionality of the system, we look at non-linear MPC's (NMPC). We assume that the wearer of the fully paraplegic, so that the system of the exoskeleton and the human behaves like a bipedal walker. The NMPC contol scheme is applied to a model of the most common device, which is a lower limb exoskeleton with actuated hips and knees. The bipedal walker is able to react to its surroundings by taking input in the form of a height map of the ground in the sagittal plane. For implementation of the device, this is the output of a machine vision algorithm, but in this research we assume that the height map is fully known. We investigate a selection rectangular obstacles with heights and lengths ranging up to 0.3m, which is about 30% of the leg length. The performance of the NMPC is validated by feeding the designed torque trajectories to simulations of the real-time behavior of the system. The results of the dynamical simulation are sent back to the NMPC to form a feedback loop. One of the great advantages of the NMPC control scheme is its versatility. Since it makes no use of predefined reference trajectories, we can cover a wide range of walking scenarios with a single control scheme. This research provides a proof of concept for the NMPC scheme as a control method for unstructured terrain and is a first step towards implementation on a real device.

1.3.1. Model

The model used in this research is based largely on the system presented in a tutorial written by Matthew Kelly and shown in Figure 1.2 [23].
We model the system of the human and exoskeleton as a multi-joint rigid body system. A rigid body does not deform. As a result, the moment of inertia of the limbs around its center of mass is constant. The human and exoskeleton are assumed to be attached rigidly, so that the human leg and the exoskeleton can be treated as a single rigid body. Interaction between the human and exoskeleton is taken into account in the joint torques.

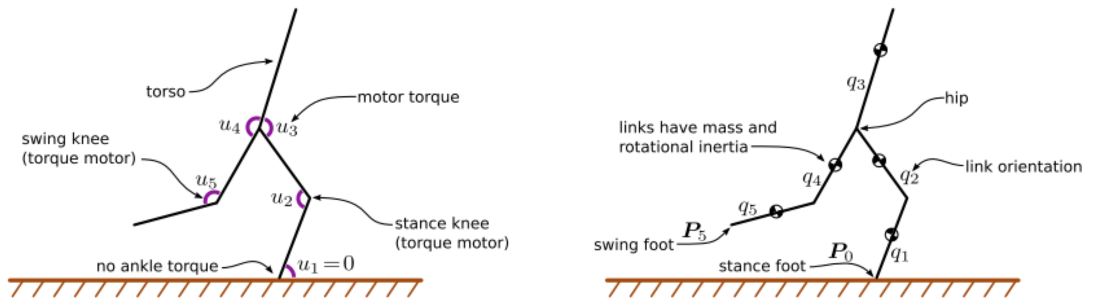


Figure 1.2.: The dynamical model as described by Matthew Kelly [23].

The head, arms and trunk (HAT) are modeled as a single rigid body.
For simplicity, only movement in the sagittal plane is taken into account. The swing and stance legs and the trunk are assumed to meet in the same point in the sagittal plane. The model thus consists of five rigid bodies: the lower stance leg, the upper stance leg, the HAT, the upper swing leg and the lower swing leg. The feet are not modeled. We consider only the single support phase of gait. The stance foot remains in the same position on the ground and does not slip. The joints are assumed to be frictionless and have no stiffness. Parameters, such as the position of the center of mass of the rigid bodies and their moments of inertia, are determined using a model of the average human body. They are gathered in a table in Appendix A.
As a result of these modeling simplifications, the model has five degrees of freedom, which are represented by the angles between the longitudinal axes of the bodies and the upwards direction, see Figure 1.2b. They are gathered in the state vector $\boldsymbol{q} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 & q_5 \end{bmatrix}^T$. The kinematics of the system provide the joint positions $\boldsymbol{P} = \begin{bmatrix} P_1 & P_2 & P_3 & P_4 & P_5 \end{bmatrix}^T$ as a function of the state $\boldsymbol{q}$. The input of the system is given by the joint torques, which are written in the control vector $\boldsymbol{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 & u_5 \end{bmatrix}^T$. The ankle torque u_1, see Figure 1.2a, is written here for completeness, but since the bipedal walker can only provide hip and knee torque, u_1 is constrained to remain zero. The dynamics of rigid bodies can be described by a second order non-linear differential equation, as is shown in Appendix B.1, and are given in their most general form as:

$$\boldsymbol{\mathcal{M}}(\boldsymbol{q})\ddot{\boldsymbol{q}} = \boldsymbol{\mathcal{F}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}), \tag{1.1}$$

where $\mathcal{M}$ is the inertia matrix and $\mathcal{F}$ contains all forces working on the system.

1.3.2. Control Task

Given the assumptions in the previous section, we define the objective and constraints for the NMPC.

Objective The goal of the research is to design a motion trajectory that is presented to a human. For optimal intuitive cooperation between the exoskeleton and the wearer, the exoskeleton should mimic human walking. It has been hypothesized that humans have, through evolution, specialized physiologically and anatomically for endurance running, which is characterized by a minimal energy expenditure [24]. Therefore, we assume that anthropomorphic motion trajectories can be generated by minimizing the cost of transport (CoT).

C1 Dynamics: The joint and torque trajectories should match with the dynamics of the system.

C2 Joint Constraints: The biomechanical joint constraints of the human must not be violated. The human body is limited in its range of motion and applied velocities and torques. Violating these constraints may lead to injury to the wearer.

C3 Balance: The bipedal walker must maintain dynamic balance, which in this context is defined as the ability of the system to recover to a stable position. Dynamic balance is necessary to avoid tripping, which may lead to damage to the exoskeleton and injury to the wearer.

C4 Obstacle Crossing: The bipedal walker must not make contact with the obstacle. Contact with the object may lead to damage to the object or exoskeleton, injury to the wearer, or loss of balance.

C5 Velocity: The bipedal walker must maintain forward velocity.

2. Methods

The goal of trajectory optimization is to find an optimal trajectory that performs a certain task, while satisfying a collection of conditions. The optimal trajectory is defined mathematically as the minimum of an objective function, which is a function of state and control variables. The optimization problem can be written in standard form as:

$$\begin{aligned} \min_{\boldsymbol{z}} J(\boldsymbol{z}) \quad & \text{subject to} \\ & \boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{0} \\ & \boldsymbol{g}(\boldsymbol{z}) \leq \boldsymbol{0} \\ & \boldsymbol{z}_{low} \leq \boldsymbol{z} \leq \boldsymbol{z}_{upp} \end{aligned} \tag{2.1}$$

Here, $\boldsymbol{z}$ is called the *decision variable*, which is what the optimizer adjusts in order to find a minimum of the objective function $J(\boldsymbol{z})$. The decision variable usually contains the evolution of the state and control variables over time. The optimization may be subject to an *equality constraint* $\boldsymbol{f}(\boldsymbol{z})$, an *inequality constraint* $\boldsymbol{g}(\boldsymbol{z})$ and *bounds* $\boldsymbol{z}_{low}$ and $\boldsymbol{z}_{upp}$.
Because of the high dimensionality and non-linearity of the model and control task, it is not feasible to solve the optimization problem analytically. Therefore, we solve the optimization problem using a numerical method. We reduce the non-linear optimization problem to a system of a discrete number of equations called a non-linear program (NLP), which is solved using a commercially available solver. In our case, we use the built-in MATLAB function fmincon, where the underlying method is set to be an interior point algorithm.

2.1. Numerical Methods for Trajectory Optimization

In this section, we create a small overview of the numerical methods available for trajectory optimization. There are three general approaches to numerical solving of optimal control problems: Dynamic Programming, indirect methods and direct methods [25].
Dynamic Programming concerns itself with finding an optimal policy. In contrast to an optimal trajectory, an optimal policy provides the optimal control law for every point in the state space. A big advantage of Dynamic Programming is that once the optimal policy has been derived, it can be applied to a real system directly. Moreover, it will always find the global optimum, because it considers the entire state space. However, the computational cost of deriving an optimal policy scales exponentially with the dimension of the problem. Therefore, the method is especially relevant for low dimensional systems. The system we are considering in this research is very high-dimensional, which

makes Dynamic Programming unsuitable.
In an indirect method, the optimal control problem is solved by rewriting the necessary conditions for optimality to a boundary value problem in ordinary differential equations [26]. This boundary value problem is then discretized and solved numerically. A drawback of indirect methods is that the conditions for optimality must be known analytically in order to construct the boundary value problem. For a system of the size and complexity that we are considering, an indirect method is therefore less suitable than a direct method.
Direct methods make a parametrization of the control trajectory. The optimality criterion and the constraint are then written in terms of these parameters, creating a finite dimensional NLP. This can then be solved using a commercially available solver. An advantage of direct methods compared to indirect methods is that inequality constraints are implemented much easier [25]. Taking these considerations into account, we have decided to solve the optimization problem using a direct method.

All direct methods are based on a parametrization of the control trajectory. However, there are different optimization strategies that can be classified by how they treat the state trajectory. In general, we consider two types: sequential methods and simultaneous methods. Single shooting is a popular sequential method and regards the state trajectory as a function of the control trajectory, along with an initial state. In every iteration of the NLP solver, the state trajectory can be found with the help of a simulation, using an ODE solver with the control trajectory as input. This is then tested against any constraints, after which a new iteration starts. Thus, the control and state trajectory are determined in sequence. In contrast, simultaneous methods keep both the control and state trajectories in the decision variables. Their relationship, which is determined by the dynamics of the system, is implemented in the NLP using equality constraints. One of the most popular simultaneous methods is direct collocation.
Shooting methods are advantageous when dealing with simple systems with few constraints. For larger systems or non-linear constraints, the relationship between the control trajectory and the constraint violation is often hard to determine, which can make it hard for the solver to converge. Especially a constraint which is active along the entire state trajectory is hard to implement, because the evolution of the state variables is not part of the decision variable, like it is using direct collocation.
By considering both the control and state trajectories as decision variables in direct collocation, we drastically increase the size of the NLP, compared to if only the control was taken into account. However, we also add equally many equality constraints for the dynamics. This results in a larger NLP, but it is mostly sparse. Commercial solvers have built-in algorithms to efficiently deal with sparse optimization problems [25].
An advantage of sequential methods is that they always result in dynamically feasible solutions, even if the NLP solver has not reached the optimum yet. In contrast, in a collocation method, the NLP solver first has to converge so that the dynamics constraints are satisfied.
Both shooting methods and collocation methods a guess trajectory to start the optimization process. A perk of collocation methods is that they offer the option to describe

this guess in terms of the state trajectory. In general, there is more information on the expected state trajectory than there is on the control trajectory.

The problem we consider in this research is relatively large, non-linear and contains bounds on the state trajectories. Taking the considerations given above into account, we can conclude that direct collocation is the superior solution for this problem. Moreover, it is been proven to be feasible for use in real time in a similar engineering problem [27]. Pardo et al. present an algorithm that uses direct collocation to generate motion patterns for a four-legged robot. Therefore, we decide to make use of a direct collocation scheme to solve the optimization problem.

2.2. Trajectory Optimizer for a Single Step

2.2.1. Setting Up the Non-Linear Program

In direct collocation, the trajectory optimization problem is discretized, so that it can be converted to an NLP. In order to do this, we define a grid of N time instances, called the collocation points. All continuous trajectories are then represented by their values at the collocation points:

$$\begin{aligned} t &\rightarrow t_0 ... t_k ... t_N \\ \boldsymbol{q} &\rightarrow \boldsymbol{q}_0 ... \boldsymbol{q}_k ... \boldsymbol{q}_N \\ \dot{\boldsymbol{q}} &\rightarrow \dot{\boldsymbol{q}}_0 ... \dot{\boldsymbol{q}}_k ... \dot{\boldsymbol{q}}_N \\ \boldsymbol{u} &\rightarrow \boldsymbol{u}_0 ... \boldsymbol{u}_k ... \boldsymbol{u}_N \end{aligned} \tag{2.2}$$

The non-linear continuous-time behavior of the system is approximated with polynomial splines, which are fitted through these collocation points. At every time instance, we have to consider five angles, five angular velocities and five torques. Therefore, the NLP has been reduced to a set of $15N$ decision variables. These are collected in the vector $\boldsymbol{z}$, which is the argument of the objective function $J(\boldsymbol{z})$. The optimization problem is solved by minimizing $J(\boldsymbol{z})$, while making sure that $\boldsymbol{z}$ adheres to the constraints.

Objective As stated, the objective of the optimizer is to minimize the CoT, which is generally expressed as the amount of work delivered to the system per distance traveled, or the time integral of the absolute value of the energy delivered. The NLP solver makes use of the gradient of the objective function to find the optimum. This creates a problem, because the gradient of the absolute value function is not defined everywhere on the domain, which may cause problem in convergence of the NLP. Therefore, we define the objective function as the integral of squared torques:

$$J = \sum_{i=1}^{5} \left(\int_0^T u_i^2(\tau) d\tau \right). \tag{2.3}$$

The integral term in this function must be described in terms of the collocation points. This is done using the trapezoidal quadrature, in which we approximate the continuous-time argument $f(\tau)$ of an integral with a linear function between t_k and t_{k+1}, see Figure 2.1. The integral of the function then equals the area under a straight line, which is given by:

$$\int_{t_k}^{t_{k+1}} f(\tau)d\tau \approx \frac{1}{2}(t_{k+1}-t_k)(f_k+f_{k+1}), \tag{2.4}$$

where f_k equals $f(t_k)$. We do this for all $N-1$ segments and add them to find:

$$\int_0^T f(\tau)d\tau \approx \sum_{k=0}^{N-1}\frac{1}{2}(t_{k+1}-t_k)(f_k+f_{k+1}). \tag{2.5}$$

Combining 2.5 with 2.3 results in the following expression for the objective function:

$$J(\boldsymbol{z}) \approx \sum_{i=1}^{5}\sum_{k=0}^{N-1}\frac{1}{2}(t_{k+1}-t_k)(u_{i,k}^2+u_{i,k+1}^2). \tag{2.6}$$

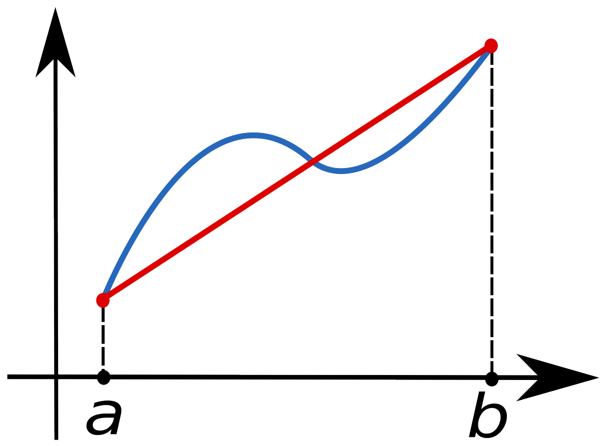


Figure 2.1.: The trapezoidal quadrature. A continuous function $f(t)$ shown in blue is approximated by a linear spline between the points a and b, shown in red. The integral from a to b of the function is then $\frac{1}{2}(b-a)(f(a)+f(b))$. Image from Wikipedia

C1 Dynamics: A key feature of direct collocation is that it finds both the state and control variables, while the dynamics are implemented as a constraint. For every pair of neighboring collocation points t_k and t_{k+1}, the continuous-time dynamics of the system can be described as:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \int_{t_k}^{t_{k+1}} \dot{\boldsymbol{x}}(t)dt, \tag{2.7}$$

where $\boldsymbol{x}$ is the state of the system, which is in our case:

$$\boldsymbol{x}_k = \begin{bmatrix}\boldsymbol{q}_k\\ \dot{\boldsymbol{q}}_k\end{bmatrix}, \text{so that } \dot{\boldsymbol{x}}_k = \begin{bmatrix}\dot{\boldsymbol{q}}_k\\ \ddot{\boldsymbol{q}}_k\end{bmatrix} \tag{2.8}$$

The angular accelerations at the collocation points are given as a function of the state and control variables in Equation B.14, which is discretized as:

$$\ddot{\boldsymbol{q}}_k = (\boldsymbol{\mathcal{M}}_k)^{-1}\boldsymbol{\mathcal{F}}_k, \tag{2.9}$$

where $\boldsymbol{\mathcal{M}}_k$ is the inertia matrix and $\boldsymbol{\mathcal{F}}_k$ contains the sum of all forces working on the system. They are defined as:

$$\begin{aligned} \boldsymbol{\mathcal{M}}_k &= \boldsymbol{\mathcal{M}}(\boldsymbol{q}_k), \\ \boldsymbol{\mathcal{F}}_k &= \boldsymbol{\mathcal{F}}(\boldsymbol{q}_k, \dot{\boldsymbol{q}}_k, \boldsymbol{u}_k). \end{aligned} \tag{2.10}$$

We again use the trapezoidal quadrature:

$$\int_{t_k}^{t_{k+1}} \dot{\boldsymbol{x}}(t)dt \approx \frac{1}{2}(t_{k+1} - t_k)(\dot{\boldsymbol{x}}_k + \dot{\boldsymbol{x}}_{k+1}). \tag{2.11}$$

The dynamics constraint is enforced by requiring that for all $N-1$ pairs of neighboring collocation points, the following is true:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{1}{2}(t_{k+1} - t_k)(\dot{\boldsymbol{x}}_k + \dot{\boldsymbol{x}}_{k+1}), \tag{2.12}$$

C2 Joint Constraints: The human body is limited in its range of motion and applied velocities and torques. Therefore, we set bounds on the state and control variables:

$$\begin{aligned} \boldsymbol{q}_{min} &\leq \boldsymbol{q} \leq \boldsymbol{q}_{max}, \\ \dot{\boldsymbol{q}}_{min} &\leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{q}}_{max}, \\ \boldsymbol{u}_{min} &\leq \boldsymbol{u} \leq \boldsymbol{u}_{max}, \end{aligned} \tag{2.13}$$

where $q_{min} = -\pi$ rad, $q_{max} = \pi$ rad, $\dot{q}_{min} = -10$ rad/s and $\dot{q}_{max} = 10$ rad/s for every segment and $u_{min} = -100$ N and $u_{max} = 100$ N for every control torque, except for u_1, which is constrained to remain zero at all times. A path constraint is added to the knee joints, to prevent overstretching of the knees. The maximum knee angle for both the swing and stance leg is constrained to 5 degrees.

C3 Balance: The exoskeleton must maintain dynamic balance, which in this context is defined as the ability of the system to recover to a stable position. By this definition, if the trajectory results in a state that is in dynamic balance, the walker is in dynamic balance along the entire trajectory. Therefore, balance control is implemented by constraining the final state of the trajectory $\boldsymbol{x}(t_F)$.
We implement heel strike, the event when the swing foot strikes the ground and becomes the new stance foot, based on impulsive collision, as is derived by Kelly [23]. Working from a principal of conservation of angular momentum, we find a function relating the state of the system before and after heel strike. It is important to note that in order to determine heel strike, the time mesh must include both t_0 and t_F. Balance is maintained by constraining the final state of the bipedal walker's trajectory after heel strike to a state for which it has been proven that a next step is feasible. This final state is found in a database that contains periodic patterns for flat ground walking, the derivation of which is explained in section 3.1.

C4 Obstacle Crossing: The exoskeleton must not make contact with the obstacle. Ground clearance is performed by an inequality constraint on the height of the endpoint position $P_{5,height}$, as well as on both knee positions:

$$\begin{aligned} P_{5,height} &> G_5 \quad \text{(Swing foot)} \\ P_{4,height} &> G_4 \quad \text{(Swing knee)} \\ P_{1,height} &> G_1 \quad \text{(Stance knee)} \end{aligned} \tag{2.14}$$

The function G_i is the height of the floor at the point P_i, which is given as input to the trajectory optimization algorithm through the height map of the environment.

C5 Velocity: The exoskeleton must maintain forward velocity. This is performed by constraining the final endpoint position $P_{5,length}$ to a desired step length:

$$P_{5,length}(t_F) = \text{step length}_{des} \tag{2.15}$$

The final state constraint for $C3$ must have a distance between the feet to the step length in order to satisfy both constraints.

2.2.2. Convergence of the Non-Linear Program

The interior point algorithm used to solve the NLP is based on the gradients of the objective function and constraints. Therefore, the solver will only converge to the global minimum if both the objective function and the constraint manifold are convex. A function is convex if the line segment between any two points on the graph of the function lies above the function. Similarly, a manifold is convex if the line segment between any two points on the manifold lies inside the manifold. If either the objective or constraints are non-convex, the optimizer may get stuck in a local minimum.
The NLP constructed in the previous section is non-convex, because the dynamics are defined by a highly non-linear function. Moreover, the position of the endpoint is given by a non-convex function in the decision variables, so that the clearance constraint becomes non-convex. Therefore, the NLP will only converge to the global minimum if the initial guess lies within the region of attraction. The region of attraction is the set of initial guesses for which the gradient based method does converge to the global optimum, even though the NLP is not convex. Because the dimensionality of the problem is very high, this region is hard to determine. Methods for solving non-convex optimization problems have been heavily researched for the past few decades [26, 28]. In this research, we apply one such method, called sequential convex programming (SCP). The key element of this method is that the non-convex NLP is approximated using a simplified problem, by relaxing or removing non-convex constraints or approximating the constraint manifold with a series of convex manifolds. Then, the solution to the simplified problem is used as the initial guess for the complexer problem. We implement this method removing the clearance constraints and see that the solution actually adheres to flat ground clearance constraint, see Figure 2.2. From this, we can conclude that this

solution is also the optimum for the flat ground problem with clearance constraints. If there is an obstacle, the found solution will violate the clearance constraint, so it is not a solution to the full problem. However, we assume that the flat ground solution lies in the region of attraction, so we use this trajectory as the initial guess for the solver.

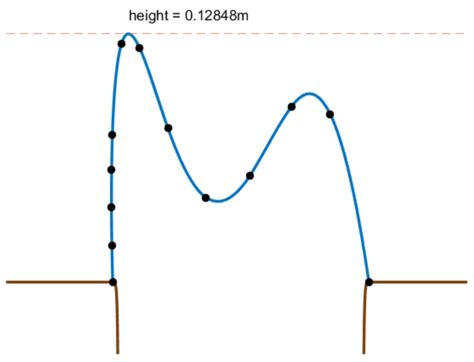


Figure 2.2.: The endpoint trajectory for the NLP when clearance constraints are removed

2.2.3. Interpolation

The trajectory optimizer finds the optimal solution in the collocation points. To obtain a continuous-time solution, the state and control variables are be interpolated. In doing so, it is important that the interpolation scheme corresponds with the collocation scheme. We applied the trapezoidal quadrature to approximate the integral action in the objective and the dynamics. We assumed that the argument $\boldsymbol{u}$ of the integral in the objective function was a linear function between two collocation points. After finding the solution for $\boldsymbol{u}$ using the direct collocation method, we interpolate between the collocation points using a linear spline. By using the trapezoidal quadrature in the dynamics, the NMPC assumes that the derivative of the state is linear between the collocation points, see Equation 2.11. Therefore, the state variables must be interpolated using a quadratic function. A more detailed description of the interpolation process can be found in the tutorial by Kelly [23].

2.2.4. Smart Meshing

The solution of the trajectory optimizer is only guaranteed to meet the constraints at the collocation points. The continuous-time function does not necessarily adhere to the constraints. It may occur, for instance, that the optimizer produces an endpoint trajectory that does not penetrate the object at the collocation points, but tunnels through it between two points. This issue is resolves by applying smart meshing. After interpolation, the continuous-time solution is checked against the constraints. If an interpolated spline segment is found to violate the constraints, a finer mesh of collocation points is applied to that segment and the optimizer is run again. This process is performed iteratively until the interpolated trajectory adheres to the constraints. The interpolated trajectory will always have a larger constraint violation than the collocation points. Therefore, we implement a buffer zone around all constraints, ie the buffer for the obstacle crossing constraint shown in Figure 2.3.

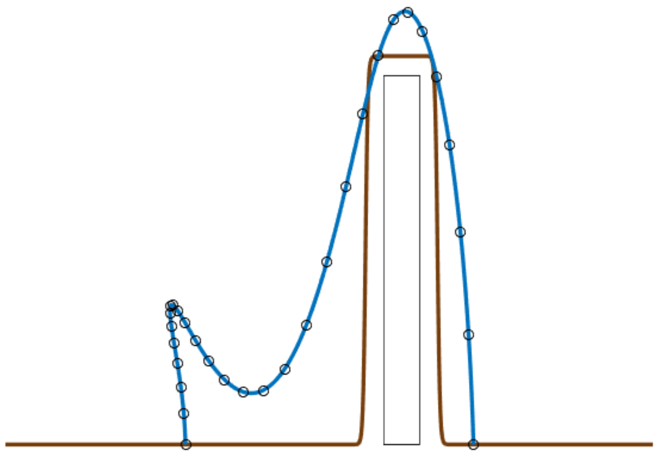


Figure 2.3.: The collocation points must lie outside the buffer, but the interpolated splines may penetrate it.

2.3. Simulated Real-time Control

The behavior of the NMPC is validated by sending the control torque trajectories found by the trajectory optimizer to a simulation of the system dynamics. This simulation is performed in MATLAB using the ODE45 function. The trajectory optimizer approximates the continuous time dynamics of the system by interpolating between the collocation points. As a result of this approximation, the motion profile obtained by applying the control torques to the system will deviate from the predicted trajectory. To make sure that the real-time motion trajectory adheres to all constraints, we need to make use of feedback control. At every discrete time step, the current state of the system is fed back to the trajectory optimizer. The optimizer then updates the predicted control trajectory accordingly, for a discrete time period in the future, called the prediction horizon. Standard MPC algorithms use a fixed horizon, creating a sliding window. The algorithm presented here uses a dynamic horizon, which always stretches

from the current time to the predicted time of heelstrike, which can be thought of as closing window control.
When updating the predicted trajectories, the previous solution to the NLP is used as initial guess. If the deviation from the original trajectory is small, the optimization problem changes very little, so that we can assume that the guess lies close to the local optimum. This way, the solver is expected to converge to it quickly, so that generating updates to the trajectory is computationally inexpensive.

2.4. Step Sequence for Obstacle Crossing

The trajectory optimizer receives input about the obstacle through the height map of the area. The initial state of the trajectory results directly from the trajectory of a previous step. The only input required from the user is the step length and the step time. In this section, we derive a strategy to make the system work fully autonomously. First, we investigate the relationship between the optimal step time and the step length, so that we can define a function that provides the step time for a given step length. We do this by applying the trajectory optimizer to a flat ground scenario and finding a periodic pattern, such as described in section 3.1. We do this for a large range of step lengths and step times and record the corresponding CoT. For every step length, we can then find the step time for which the CoT is minimal, and fit a function through these points.
A typical obstacle crossing strategy entails a sequence of multiple steps. Starting from stable flat ground walking, the bipedal walker must first make an approach step to position the foot in front of the object. Then the front leg crosses the obstacle, followed by the back leg. The walker has now crossed the obstacle and must make a final recovery step to return to flat ground walking.
The only variable left to decide to design a strategy for obstacle avoidance is the step length of every step. We use the trajectory optimizer to find the optimal step length of all steps in the sequence. We extend the example problem to include an approach step. Therefore, we place the object at a distance of 0.8m. The rest of the problem remains the same.
We apply the trajectory optimizer sequences with varying step lengths of the approach step, front leg crossing and back leg crossing. Because of the way we implemented balance in this algorithm, the final state of the back leg crossing step is already present in flat ground walking, so that a recovery step is not necessary. The trajectory optimizer finds the optimal trajectory for the approach step, then performs the heel strike algorithm and uses the resulting state as initial conditions for the front leg crossing, which then leads to back leg crossing. We measure the cost of every step sequence. The optimal step length sequence is when this cost is minimal. We apply this method to a variation of objects in order to define the range of objects that the biped can cross.

3. Simulation Results

Using the equations described in the previous section, a MATLAB program is constructed that takes the following parameters as input:

- **Object**
  - H = Height of object
  - L = Length of object
  - D = Distance between stance foot and object
- **Configuration of the bipedal Walker**
  - x_0 = Initial state
  - x_F = Final state
  - t_F = Final time
  - $P_{5,F}$ = Step length
- **Mesh Settings**
  - nGrid = Number of grid points
  - S = Type of Spacing of Collocation Points

3.1. Trajectory Optimizer for a Single Step

Flat Ground We assume that an anthropomorphic walking pattern is symmetric and periodic, meaning that there is no difference between the left and the right leg motion pattern. Moreover, every step follows the same motion profile. We generate walking patterns for symmetric walking using the trajectory optimizer described in the previous chapter. For periodic and symmetric walking, the final state of the trajectory after heel strike must be equal to the initial state. Using this extra constraint and a mesh of 10 evenly spaced points, we generate a motion pattern for flat ground walking, see Figure 3.1. This figure shows a stop-action animation of the step, with frames spaced uniformly in time. The red dashed lines represent the stance leg and the blue solid lines represent the swing leg. The purple line reprsents the torso.
This simulation is run for a large range of step lengths and step times. This serves a triple purpose. First, through this method we find the energetically optimal step length and step time for the bipedal walker. Second, the final configurations found for symmetric walking are guaranteed to be in balance, because the motion is sustaining.

These configurations are therefore saved in a database to serve as the balance constraint for any following simulations. Finally, the motion trajectories are saved to a database to serve as initial guesses for the trajectory optimizer.

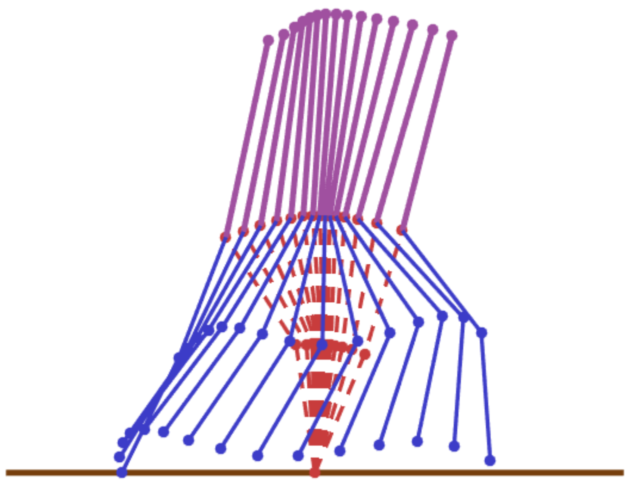


Figure 3.1.: A motion trajectory for flat ground walking.

Slope Walking A major advantage of the NMPC scheme is its versatility. To illustrate this, we also generate motion trajectories for slope walking. To do this, we change the ground trajectory to a constant slope and let the optimizer find a periodic and symmetric pattern. Motion trajectories are shown in Figure 3.2.

3.1.1. Example Problem

We investigate the behavior of the trajectory optimizer for an example problem. We place an object with a length and width of 25 cm in front of the bipedal walker at a distance of 20 cm from the stance foot. The initial and final state of the walker are constrained to the configurations shown in Figure 3.3. These configurations are the result of previous simulations, for which it has been shown that they adhere to the balance constraint $C3$. The step time is set to 1.3 s and the step length to 60 cm. This step time was proven empirically to give the the highest possible average COM velocity. We use a mesh with 10 collocation points, which are evenly spaced in time. The optimizer must be fed with an initial guess for the decision variables, for which we use the solution found for flat ground walking with the same step length.

The joint angles, angular velocities and torques corresponding to the trajectory are

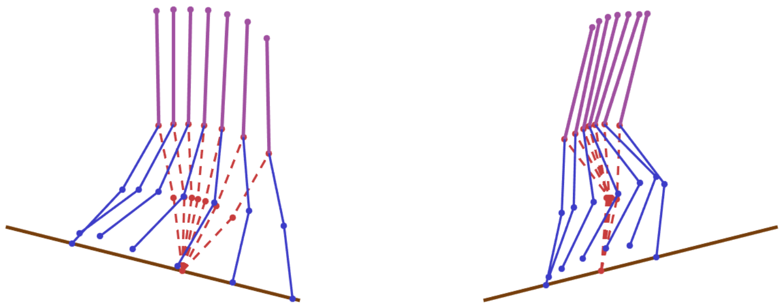
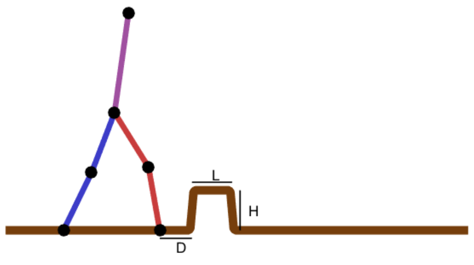
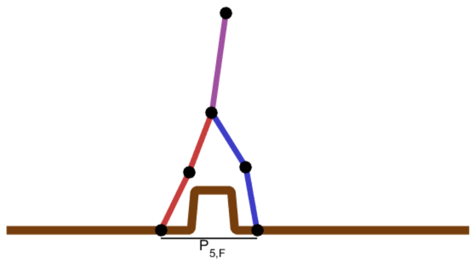


Figure 3.2.: (a) A motion trajectory for walking down a slope (b) A motion trajectory for walking up a slope.

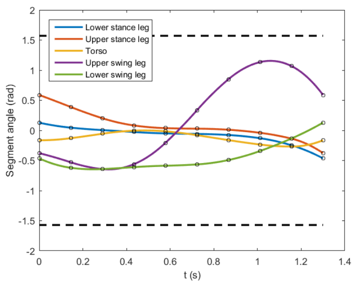


(a) Initial configuration

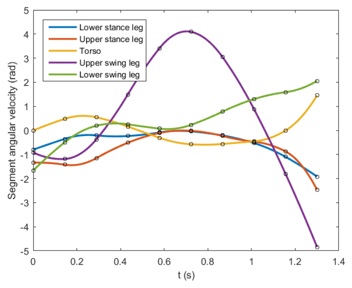


(b) Final configuration

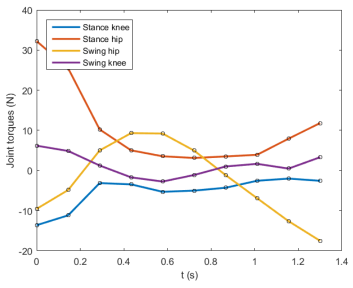
Figure 3.3.: The goal of the simulation is to find a trajectory that moves the bipedal walker between the configuration in (a) to the configuration in (b).



(a) Segment angles

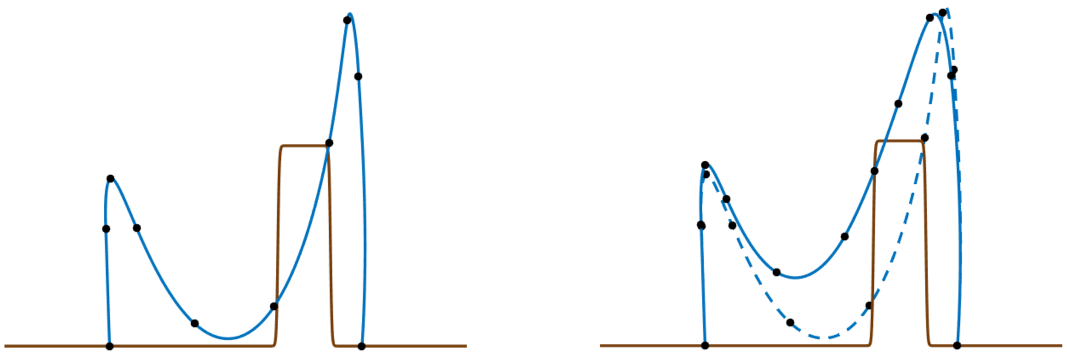


(b) Segment angular velocities



(c) Joint torques

Figure 3.4.: Angles, velocities and torques of the solution of the optimization probelem, including bounds. The maximum velocity is 10 rad/s and the maximum torque is 100 N. Both are out of bounds of the figure.



(a) Before smart meshing (b) After smart meshing

Figure 3.5.: Endpoint trajectories before and after smart meshing.

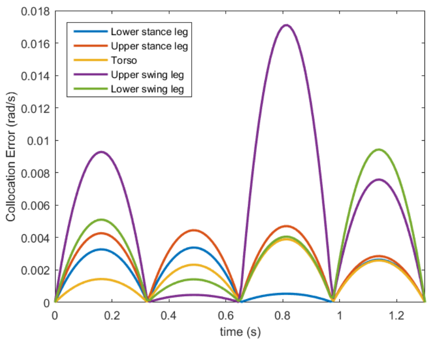
shown in Figure 3.4. As expected, all joint constraints are satisfied in the collocation points, see Figure 3.4.

Smart Meshing

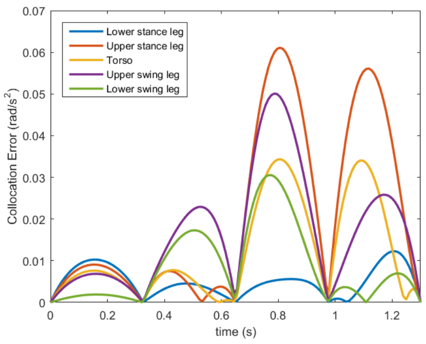
The trajectory of the endpoint is shown in Figure 3.5a. At the collocation points, the obstacle crossing constraint is satisfied, but the interpolated trajectory tunnels through the object. This issue is resolved by applying smart meshing. An extra collocation point is added in the segment that penetrates the object. This results in the updated trajectory shown in Figure 3.5b.
Figure 3.6 shows the error in the dynamics of the interpolated trajectory for the example problem, when a uniform mesh of five points is used, before and after smart meshing is applied. The error in the dynamics is given as the difference between the time derivative of the interpolated trajectory, $\dot{\boldsymbol{x}}$, and the output of the dynamics function. At the collocation points, the error is (almost) zero, because this is enforced by the dynamics constraint in the NLP. By implementing smart meshing on the third segment, the error in the dynamics between collocation points is decreased.

Influence of Mesh

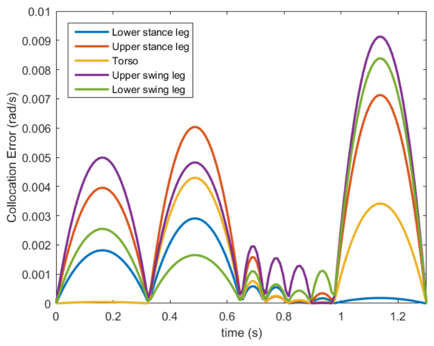
We investigate the influence of the chosen mesh on the solution. We vary the number of points in the mesh and record the computation time of the optimizer and the quality of the solution, which is defined by the cross-correlation with the global optimum. Figure 3.7 shows the cross-correlation of the solution with the global optimum and the associated cost for meshes ranging from 2 to 30 grid points when applied to the example problem. As expected, a denser mesh results in a more accurate and efficient solution.
The time necessary to run the NMPC is directly related to the number of grid points, see Figure 3.8. This figure is obtained by running the example problem, which produces



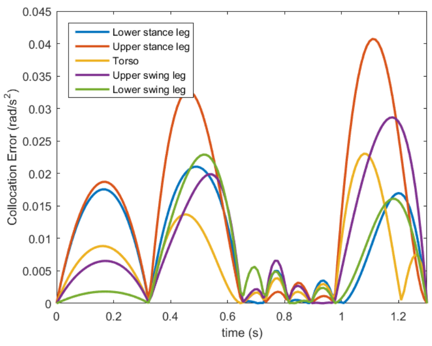
(a) Error in $\dot{\boldsymbol{q}}$ before smart meshing



(b) Error in $\ddot{\boldsymbol{q}}$ before smart meshing



(c) Error in $\dot{\boldsymbol{q}}$ after smart meshing



(d) Error in $\ddot{\boldsymbol{q}}$ after smart meshing

Figure 3.6.: The error in the dynamics for the example problem when a uniform mesh of five points is used, before and after smart meshing is applied.

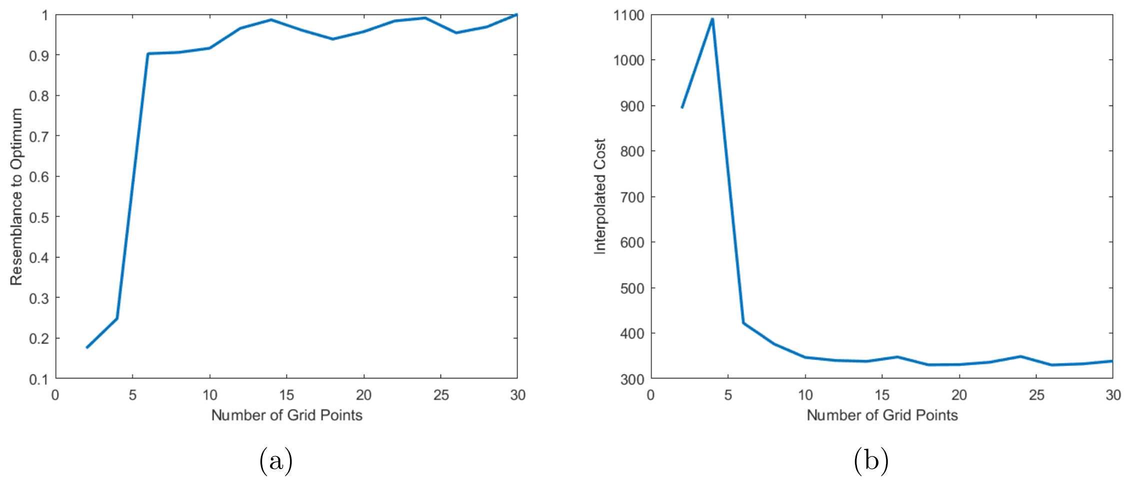


Figure 3.7.: The resemblance (a) and the associated cost (b) for meshes ranging from 2 to 30 grid points when applied to the example problem.

a trajectory of 1.3 seconds, for meshes ranging from 2 to 30 grid points. The time shown is the time necessary for the NLP solver to find the local optimum of the function. The program can therefore easily be made faster by reducing the number of grid points, which is relevant if the method is to be applied in a real-time situation. However, as seen in Figure 3.7, this will also decrease the accuracy of the solution. Selection of the optimal mesh density is therefore a matter of weighing the requirements in terms of accuracy and computation time. Both accuracy and cost stagnate after around 10 grid points, so this mesh is assumed to be sufficiently dense and is used in all further simulations.
The trajectory optimizer is designed for use in an NMPC, which updates the trajectory with the control rate. This means that collocation points that are later in time get treated much more extensively than those close to the current time. Since the execution time depends on the number of grid points, it is beneficial to reduce this number. This can be done by implementing a mesh of collocation points that is denser in the beginning than in the end. We run a simulation of the example problem for two different meshes: the blue line represent a mesh that is uniformly spaced and the red line a mesh for which the distance between points changes quadratically. Both meshes are initialized with the same guess. Figure 3.8 shows that the spacing of the grid points does not significantly influence the execution time of the optimizer.

3.2. Simulated Real-time Control

Real-time control is simulated using the MATLAB built-in function ODE45. At every discrete time step, the trajectory optimizer generates control torques for the prediction horizon, which ranges up to heel strike. The ODE45 solver uses the predicted control trajectory alongside the current state of the system to simulate the response of the sys-

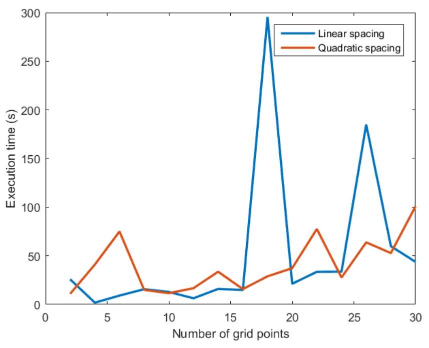


Figure 3.8.: The execution time of the optimizer for two different meshes as a function of the grid size.

tem to the control torque up to the next time step. Then the state is measured and fed back to the optimizer. This method is applied to the example problem presented in the previous section, using a control rate of 1.3/6 seconds, which corresponds to a time step of 0.1s. Figure 3.9 shows the endpoint trajectory in the sagittal plane at every time step. Here, the brown line represents the floor. The yellow is the previous optimizer prediction of the endpoint trajectory, whereas the black line shows the actual output of the system. The current position is marked with a big black circle. The orange trajectory shows the updated prediction corresponding to the current position.
The final motion trajectory is checked against all constraints to see if the real-time controller was successful. The maximum allowed constraint violation is set equal to the size of the constraint buffer, so that we are sure that the real-time trajectory adheres to the real constraints of the system. Figure 3.10 shows the maximum constraint violation for different control rates when the real-time controller is applied to the example problem. The maximum allowed constraint violation is shown in the thin blue line. For control rates under 0.02s, the constraint violation remains below the constraint tolerance. Therefore, we can conclude that the minimal control rate for the example problem is 0.02s or a frequency of 50Hz.

3.3. Step Sequence for Obstacle Crossing

For every step length, we plot the cost of transport, see Figure 3.12a. In this figure, every line represents a constant step time. We connect the minima for every step length to find the optimal step length, which is at 0.32m and 1.4 seconds. We plot the step time corresponding to the best solution for every step length in Figure 3.12b. We find a linear relationship between the step length and its optimal step time. This function is then used to design a foot placement strategy for a step sequence. We solve the NLP for a variation of step sequences. The step length of the approach step varies so that the end position ranges from 30 to 0 cm in front of the obstacle. The length of the front leg crossing step ranges so that the endpoint lies from 0 to 30 cm behind the obstacle.

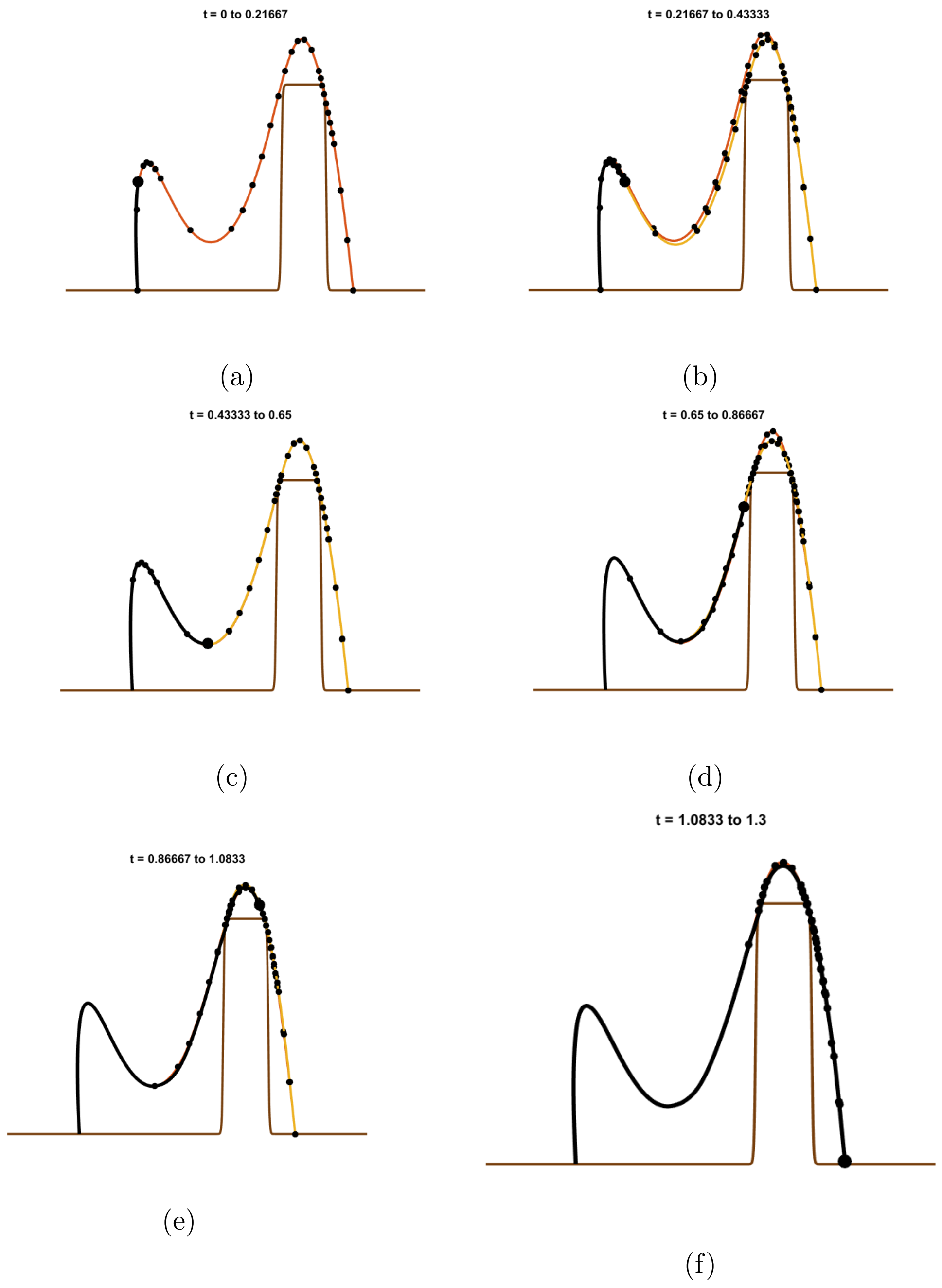


Figure 3.9.: the endpoint trajectory in the sagittal plane at every time step, when a control rate of 0.216 seconds is applied.

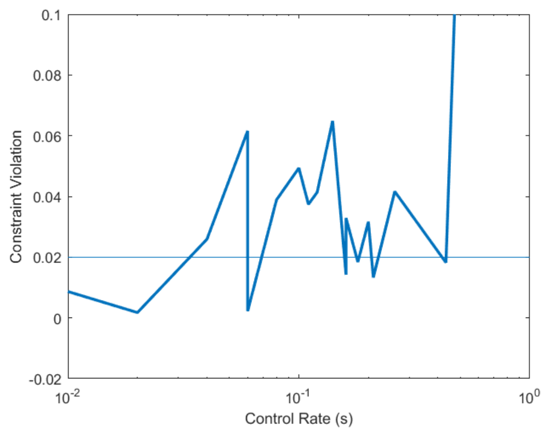


Figure 3.10.: The constraint violation of the real-time trajectory for different control rates.

The back leg crossing step length varies from 42 to 70 cm. We find the optimal solution where the foot is placed 22 cm in front of the obstacle, then 5 cm behind the obstacle and then 42 cm further. Figure 3.13 shows stop action animations for this sequence.

A simulation of real-time control is performed for this optimal solution. In contrast to the example problem presented in the previous section, the real-time controller fails to generate a motion trajectory that adheres to the constraints for this scenario, even for a very high control frequency. When a longer step time is enforced, the solution of the trajectory optimizer brings the stance knee very close to full extension. Hyperextension of the knee is not dealt with in the dynamics function, but is only inhibited by the joint constraints of the trajectory optimizer. Even though the smart meshing algorithm makes sure that the interpolated knee angle trajectory does not violate the constraint, the real-time trajectory can deviate from the optimal solution in such a way that the knee hyperextends. Moreover, the fully extended position is and unstable equilibrium, so that the further the knee overstretches, the more force is necessary to bring it back to a feasible position. The model predictive controller cannot correct for this, causing the real-time simulation to fail.
The step sequence algorithm is applied to objects ranging from 5cm to 50 cm in length and height. Figure 3.11 shows the range of objects for which the algorithm finds a solution. Green dots represent objects for which at least one step sequence has been found. Red crosses represent objects for which none of the combinations of step lengths produced a feasible solution. The trajectory for a certain object is also feasible for any smaller object, which is represented by the green rectangles. Similarly, if the optimizer cannot find a solution for an object, it will also fail for any larger objects, which is shown using the red squares.

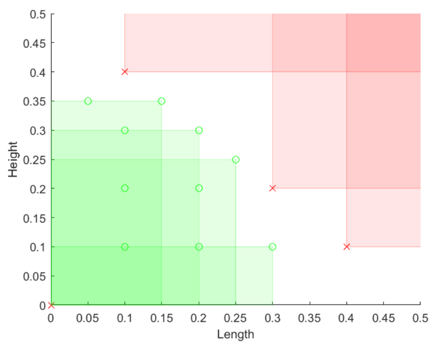
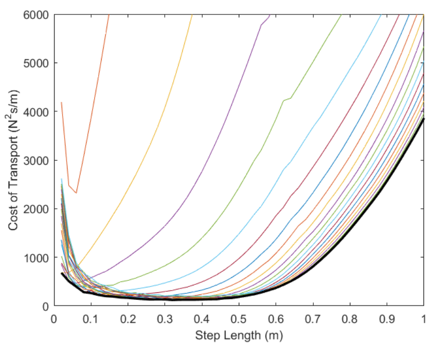
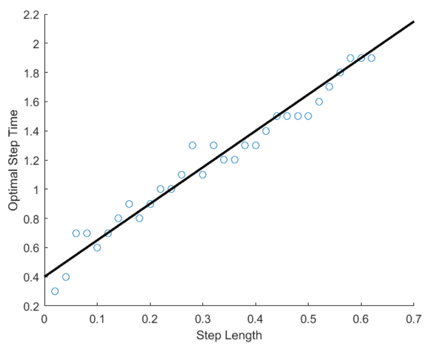


Figure 3.11.: The range of objects for which the step sequence algorithm finds a solution.

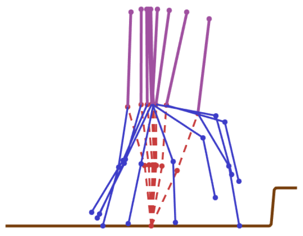


(a)

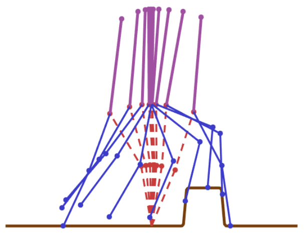


(b)

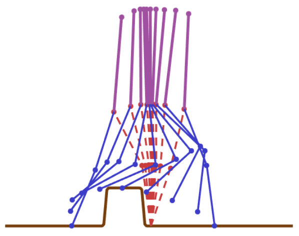
Figure 3.12.: The cost of transport and optimal step time for different step lengths in a flat ground walking scenario.



(a) Approach



(b) Front leg crossing



(c) Back leg crossing

Figure 3.13.: A stop action animation of the optimal sequence for the example problem

4. Discussion

The aim of this research is to design a NMPC that generates a movement trajectory that allows the wearer of an exoskeleton to step over an obstacle. In this thesis, we present a control strategy based on a direct collocation scheme. The NMPC scheme has been implemented in MATLAB. We generate motion patterns that agree with the constraints given in Section 1.3, which are tracked and updated using a feedback loop in a simulation of real-time behavior. The method successfully makes the bipedal walker step over an obstacle, thus providing proof of concept for NMPC for obstacle avoidance. The trajectory optimizer is used to design a strategy to make the bipedal walker navigate its environment fully autonomously. In addition to crossing over obstacles, the versatility of the method is illustrated by applying it to slope walking as well.
The results of our research are comparable to the work presented by Kalamian et al. [29]. We have designed a controller that can step over obstacles up to 35cm in height, where they manage to cross over an obstacle of 40cm. Even though the methods are comparable, there are a few differences. First and foremost, the method presented in this work allows the bipedal walker to traverse the obstacle while maintaining COM velocity throughout the entire sequence. The strategy presented by Kalamian et al. requires the walker to come to a full stop in front of the obstacle before crossing over it.
Kalamian et al. implements a balance using the zero moment point, along the entire trajectory, where we only impose it on the final state. Path constraints are computationally less efficient than equality constraints, which means that our method is potentially faster. However, Kalamian et al. do not present any data on the control rate of their controller. In contrast, using a neural network instead of numerically solving the dynamics equation probably makes their method faster. A big advantage of using this model compared to the neural network (NN) is that our dynamical model is easily changed in order for the method to be applied to a different system. When the system on which the controller is used changes, the NN has to be retrained, which is unfavorable.

Balance control Balance control was implemented using a final state that was drawn from a database that contained trajectories for flat ground walking. This implementation of balance control is overly constrictive, because the full final state is constrained, whereas there is a family of solutions that are in balance. A different balance protocol was tested using the extrapolated Center of Mass (xCOM), but this did not guarantee final configurations from which the bipedal walker was able to recover. This is because the walker tended to use very large deviations in the torso angle to change its xCOM. This way, the balance constraint was kept, but the walker was not able to recover to symmetric walking from the final position. We can conclude that the xCOM constraint on the final state is not sufficient to ensure dynamic balance. The method could be extended by constraining the torso angle further or by implementing an additional constraint that makes sure the walker remains upright. The vertical position of the COM can be constrained to remain above a certain threshold. Another solution is to use an altogether different balance metric, such as the angular momentum of the COM. It has been shown that this is very stable during normal walking in humans and may therefore be a valid option for a balance metric [30].

Velocity control The velocity of the bipedal walker is controlled by constraining the endpoint position at the final time to a certain step length. In section 3.3, we find that the optimal combination of step length and time for flat ground walking found by the trajectory optimizer is 0.32 cm and 1.4 s. This is equivalent to an average COM velocity of 0.23 ms^{-1}. The method favors trajectories in which the walker fully extends the stance leg and then lets the swing leg swing freely towards its end position, thus making optimal use of the passive dynamics of the system. However, by fully extending the stance knee, we run into trouble when implementing real-time control. Moreover, the walking speed is very low compared to normal walking. Velocity control may be improved by implementing a lower bound on the average COM velocity and let the trajectory optimizer free to choose the step time. Alternatively, it can be added to the objective function, following the work of Kalamian et al. [29].

Convex Approximation of the NLP In this research, we used sequential convex programming to deal with the non-convexity of the optimization problem. In order to solve the flat ground walking problem, the non-convex NLP was approximated by removing the foot and knee clearance constraints. It was shown that the solution to the approximated program adheres to the non-convex constraint and thus is also the optimum for the flat ground scenario. However, this problem was still not convex, because the highly non-convex dynamics were still present in the problem. Therefore, we cannot guarantee that the solution is the global optimum. A more accurate solution could be found by adding a simplification step to the sequence by linearizing the dynamics at every grid point, so that the entire problem is convex. If a very dense grid is used, the linearized dynamics represent the actual dynamics accurately. The solution to this problem is guaranteed to be the global optimum. It can then be used as an initial guess to the NLP with non-linear dynamics.

Hyperextension of the Stance Knee The NMPC method is shown to produce a feasible real-time control for the example problem. However, it fails when the average velocity becomes too low and the solution brings the stance knee close to full extension. This may be solved by implementing a tighter knee extension bound, so that the solution from the trajectory optimizer does not get as near to full extension. Another solution is to model a knee stop into the dynamics, that makes sure that the knee does not overstretch. In doing this, care must be taken that the dynamics function remains continuous and smooth, because discontinuities or undefined gradients impede convergence of the numerical solver.

4.1. The Next Step: Moving towards Implementation on a Device

The final goal for exoskeleton technology is to allow SCI patients to autonomously navigate daily life. For this, the method must be implemented on an actual device. This research provides a solid proof of concept for the NMPC, but there are still steps to be taken before the method can be implemented in real-time. The minimal control rate for the example problem was found to be 50Hz in the simulations. The example problem lies on the edge of the range of feasible objects, so that the trajectory tends to get close to its bounds and constraints. In the neighborhood of constraints, the necessary control rate is higher, because then it is of paramount importance that the trajectory is followed closely. For smaller objects, the trajectories are less constrained, so that we expect the minimal control rate of 50Hz to be applicable to all obstacles. However, in real life, we need to deal with perturbations, noise and other sources of uncertainty, so we expect that the necessary control rate for real life application is higher.
For meshes of 10 grid points, which was determined to be sufficiently accurate, the trajectory optimizer has a computation time of around 20s. Therefore the algorithm must be sped up with two orders of magnitude. There are several solutions to realize this. First a bigger database for initial guesses can be made, so that the optimizer is started closer to the global minimum and will need fewer iterations to reach a solution. Second, the fmincon function can be sped up considerably by feeding it analytical gradients of both the objective and the constraint functions. These gradients are now determined numerically, which is computationally expensive. Finally, it can be investigated if the NLP can be simplified using linearization of the dynamics.
For the exoskeleton to work fully autonomously, it should respond to the environment. In the current setup, we assume the height map to be known a priori, but for implementation on the device, this should be replaced with a machine vision scheme. Using for instance a depth camera, a 3D map of the environment can be generated, to which the bipedal walker can respond. The step sequence for obstacle avoidance solves the optimization problem hundreds of times in order to find the optimal foot placement strategy. Therefore it cannot be used in real time. The simulations can be used by finding optimal strategies for a range of objects and extracting a function describing the optimal strategy with respect to the obstacle parameters. Alternatively, a lot of great work has been done on foot placement algorithms that may be applied to the system [17, 31–33].
Finally, the dynamical model and method used in this research make some assumptions and contain simplifications. The method considers only single support phase, using an impulsive heel strike algorithm. In order to design truly anthropomorphic motion trajectories, it is necessary to extend the method to also incorporate a double support phase. The dynamical model neglects any ankle strategy. It should be investigated to what extent adding feet in the dynamical model increases the computational effort required for the NMPC.

Bibliography

[1] S. B. Jazayeri, S. Beygi, F. Shokraneh, E. M. Hagen, and V. Rahimi-Movaghar, “Incidence of traumatic spinal cord injury worldwide : a systematic review,” *Eur Spine J*, vol. 24, no. 5, pp. 905–918, 2015.

[2] N. A. Silva, N. Sousa, R. L. Reis, and A. J. Salgado, “From basics to clinical: A comprehensive review on spinal cord injury,” *Progress in Neurobiology*, vol. 114, pp. 25–57, 2014.

[3] H. Herr, “Exoskeletons and orthoses: classification, design challenges and future directions.,” *Journal of neuroengineering and rehabilitation*, vol. 6, p. 21, 2009.

[4] J. L. Contreras-Vidal, N. A. Bhagat, J. Brantley, J. G. Cruz-Garza, Y. He, Q. Manley, S. Nakagome, K. Nathan, S. H. Tan, F. Zhu, and J. L. Pons, “Powered exoskeletons for bipedal locomotion after spinal cord injury,” *Journal of Neural Engineering*, vol. 13, no. 3, pp. 031001–0310017, 2016.

[5] A. Young and D. Ferris, “State-of-the-art and Future Directions for Robotic Lower Limb Exoskeletons,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. PP, no. 99, pp. 1–1, 2016.

[6] T. Yan, M. Cempini, C. M. Oddo, and N. Vitiello, “Review of assistive strategies in powered lower-limb orthoses and exoskeletons,” *Robotics and Autonomous Systems*, vol. 64, pp. 120–136, 2015.

[7] K. A. Strausser and H. Kazerooni, “The development and testing of a human machine interface for a mobile medical exoskeleton,” *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4911–4916, 2011.

[8] T. A. Swift, “Control and trajectory generation of a wearable mobility exoskeleton for spinal cord injury patients.,” *University of California at Berkeley*, 2011.

[9] R. J. Farris, H. A. Quintero, and M. Goldfarb, “Performance evaluation of a lower limb exoskeleton for stair ascent and descent with Paraplegia,” *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, pp. 1908–1911, 2012.

[10] Y. Sankai, “HAL: Hybrid assistive limb based on cybernics,” in *Springer Tracts in Advanced Robotics*, vol. 66, pp. 25–34, 2010.

[11] A. Esquenazi, M. Talaty, A. Packel, and M. Saulino, "The ReWalk Powered Exoskeleton to Restore Ambulatory Function to Individuals with Thoracic-Level Motor-Complete Spinal Cord Injury," *American Journal of Physical Medicine & Rehabilitation*, vol. 91, no. 11, pp. 911–921, 2012.

[12] P. D. Neuhaus, J. H. Noorden, T. J. Craig, T. Torres, J. Kirschbaum, and J. E. Pratt, "Design and Evaluation of Mina a Robotic Orthosis for Paraplegics," *Proceedings of the 2011 International Conference on Rehabilitation Robotics*, 2011.

[13] D. Sanz-Merodio, M. Cestari, J. C. J. Arevalo, and E. Garcia, "A lower-limb exoskeleton for gait assistance in quadriplegia," *2012 IEEE International Conference on Robotics and Biomimetics, ROBIO 2012 - Conference Digest*, pp. 122–127, dec 2012.

[14] J. Gancet, M. Ilzkovitz, E. Motard, Y. Nevatia, P. Letier, D. De Weerdt, G. Cheron, T. Hoellinger, K. Seetharaman, M. Petieau, Y. Ivanenko, M. Molinari, I. Pisotta, F. Tamburella, F. S. Labini, A. D'Avella, H. Van Der Kooij, L. Wang, F. Van Der Helm, S. Wang, F. Zanow, R. Hauffe, and F. Thorsteinsson, "MINDWALKER: Going one step further with assistive lower limbs exoskeleton for SCI condition subjects," *Proceedings of the IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics*, vol. 2010, pp. 1794–1800, 2012.

[15] L. Wang, E. H. F. Van Asseldonk, and H. Van Der Kooij, "Model predictive control-based gait pattern generation for wearable exoskeletons," *IEEE International Conference on Rehabilitation Robotics*, 2011.

[16] N. Thatte and H. Geyer, "Toward Balance Recovery With Leg Prostheses Using Neuromuscular Model Control," *IEEE Transactions on Biomedical Engineering*, vol. 63, pp. 904–913, may 2016.

[17] A. C. Hildebrandt, D. Wahrmann, R. Wittmann, D. Rixen, and T. Buschmann, "Real-time pattern generation among obstacles for biped robots," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2015-Decem, pp. 2780–2786, 2015.

[18] C. Zhou, X. Wang, Z. Li, D. Caldwell, and N. Tsagarakis, "Exploiting the Redundancy for Humanoid Robots to Dynamically Step Over a Large Obstacle," pp. 1599–1604, 2015.

[19] Y. Guan, E. S. Neo, K. Yokoi, and K. Tanie, "Stepping over obstacles with humanoid robots," *IEEE Transactions on Robotics*, vol. 22, no. 5, pp. 958–973, 2006.

[20] K. Nishiwaki, J. Chestnutt, and K. Satoshi, "Planning and Control of a Humanoid Robot for Navigation on Uneven Multi-scale Terrain," *Springer Tracts in Advanced Robotics*, vol. 79, pp. 401–415, 2014.

[21] O. Stasse and B. Vanderborght, "Strategies for Humanoid Robots to Dynamically Walk Over Large Obstacles," vol. 25, no. 4, pp. 960–967, 2009.

[22] M. Arbulú, A. Kheddar, and E. Yoshida, "An approach of generic solution for Humanoid stepping over motion," pp. 474–479, 2010.

[23] M. P. Kelly, "An introduction to trajectory optimization: how to do your own direct collocation," pp. 1–44.

[24] D. M. Bramble and D. E. Lieberman, "Endurance running and the evolution of Homo," *Nature*, vol. 432, no. 7015, pp. 345–352, 2004.

[25] M. Diehl, H. G. Bock, H. Diedam, P.-b. Wieber, P.-b. W. Fast, and D. Multiple, "Fast Direct Multiple Shooting Algorithms for Optimal Robot Control To cite this version : Optimal Robot Control," 2009.

[26] A. E. Bryson, Y.-C. Ho, and G. M. Siouris, "Applied Optimal Control: Optimization, Estimation, and Control," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 6, pp. 366–367, 1979.

[27] 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 Science and Systems (RSS)*, 2017.

[28] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Society for Industrial and Applied Mathematics, jan 1996.

[29] N. Kalamian and M. Farrokhi, "Stepping of biped robots over large obstacles using NMPC controller," *Proceedings - 2011 2nd International Conference on Control, Instrumentation and Automation, ICCIA 2011*, pp. 917–922, 2012.

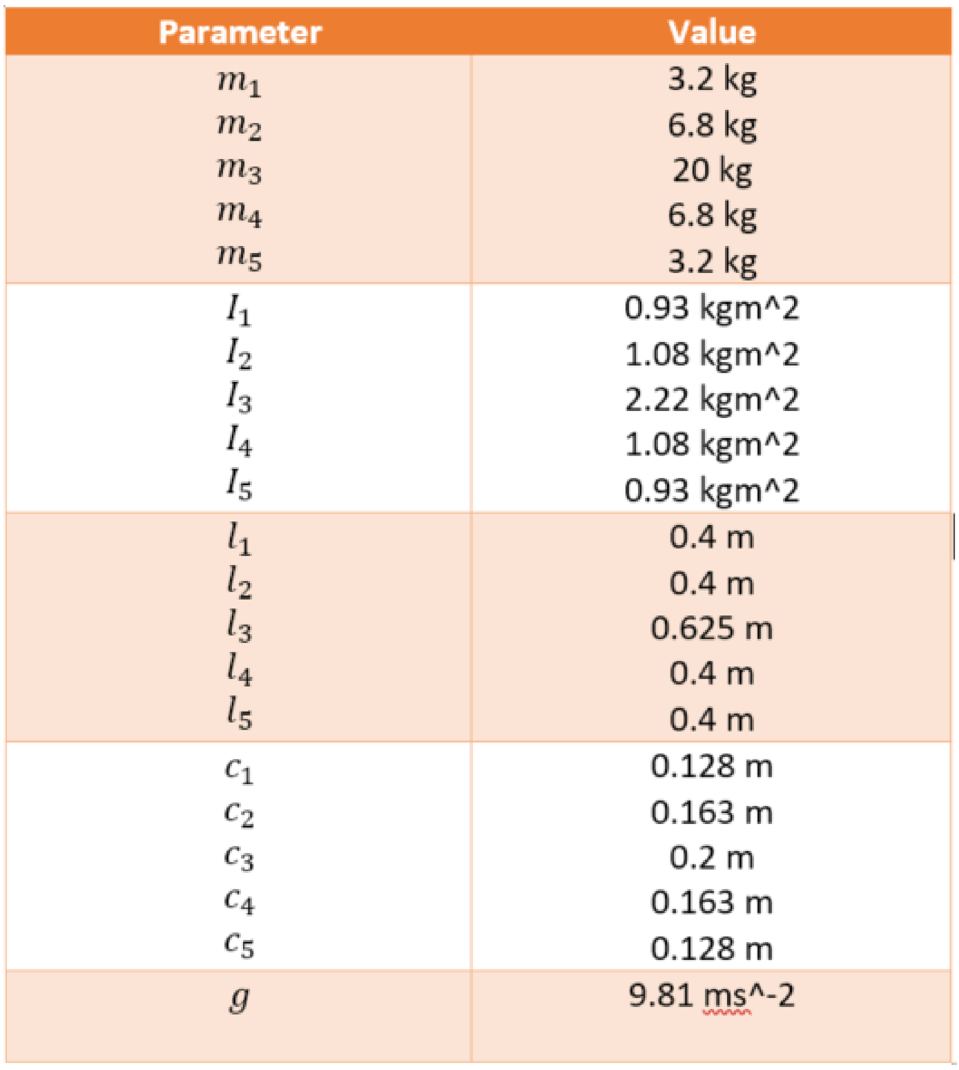
[30] J. H. Jung, L. van Opheusden, P. Barralon, and J. F. Veneman, "Real time Computation of Centroidal Momentum for the Use as a Stability Index Applicable to Human Walking with Exoskeleton," 2017.

[31] M. X. Grey, A. D. Ames, and C. K. Liu, "Footstep and Motion Planning in Semi-unstructured Environments Using Possibility Graphs," 2016.

[32] A. Stumpf, S. Kohlbrecher, D. C. Conner, and O. V. Stryk, "Open Source Integrated 3D Footstep Planning Framework for Humanoid Robots," *2016 IEEE-RAS International Conference on Humanoid Robots (Humanoids 2016)*, 2016.

[33] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous Robots*, vol. 40, no. 3, pp. 429–455, 2016.

A. List of Parameter Values



Parameter	Value
m_1	3.2 kg
m_2	6.8 kg
m_3	20 kg
m_4	6.8 kg
m_5	3.2 kg
I_1	0.93 kgm^2
I_2	1.08 kgm^2
I_3	2.22 kgm^2
I_4	1.08 kgm^2
I_5	0.93 kgm^2
l_1	0.4 m
l_2	0.4 m
l_3	0.625 m
l_4	0.4 m
l_5	0.4 m
c_1	0.128 m
c_2	0.163 m
c_3	0.2 m
c_4	0.163 m
c_5	0.128 m
g	9.81 ms^-2

B. Derivation of the Dynamics

B.1. Dynamics

In this seciton, we derive the equations of motion of the system. Using the Newton-Euler method, we split up the kinematic tree at each successive joint i of the system. Euler's second law of motion states that the rate of change of the angular momentum about a point is equal to the sum of the external torques about that point:

$$\sum \boldsymbol{\tau}_i = \dot{\boldsymbol{L}}_i. \tag{B.1}$$

We define the set $\mathcal{C}_i$, which contains the indices of the children of i and consists of all bodies that are lower in the kinematic tree than joint i. The angular momentum $\boldsymbol{L}_i$ about joint i is the sum of the influences $\boldsymbol{L}_{ij}$ of all children j of the joint, so that we can write:

$$\sum \boldsymbol{\tau}_i = \sum_{j \in \mathcal{C}_i} \dot{\boldsymbol{L}}_{ij}. \tag{B.2}$$

We start on the right hand side of the equation. The angular momentum resulting

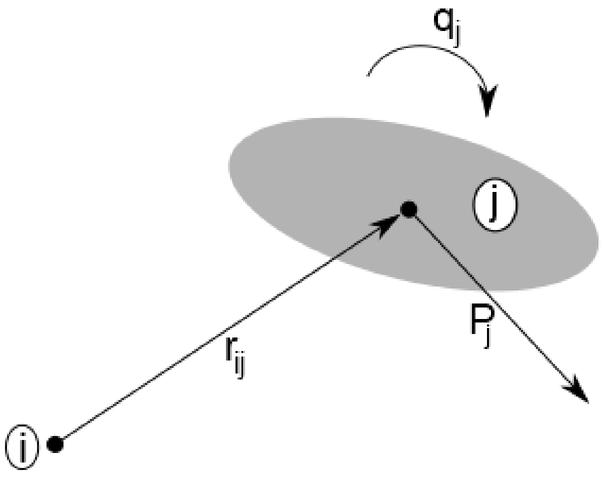


Figure B.1.

from the motion of body j is the sum of both the orbital momentum and the spin momentum. The orbital momentum relates to the rotation of j around i. It is given as the cross product of the vector $\boldsymbol{r}_{ij}$ pointing from i to the center of mass of j and the linear momentum $\boldsymbol{p}_j$ of j, see Figure B.1. The spin momentum, which relates to the rotation of j around its own center of mass, is the product of the angular velocity of the body and the moment of inertia around its rotation axis. We are working with a 2D model, so that the moment of inertia is a scalar and the angular velocity is a vector, perpendicular to the sagittal plane, with magnitude q_i. The influence of the motion of body j on joint i is given as:

$$\boldsymbol{L}_{ij} = \boldsymbol{r}_{ij} \times \boldsymbol{p}_j + I_j \dot{\boldsymbol{q}}_j. \tag{B.3}$$

In order to find the equations of motion, we need the rate of change of the angular momentum, so we take the time derivative of Equation (B.3):

$$\dot{\boldsymbol{L}}_{ij} = \dot{\boldsymbol{r}}_{ij} \times \boldsymbol{p}_j + \boldsymbol{r}_{ij} \times \dot{\boldsymbol{p}}_j + \dot{I}_j \dot{\boldsymbol{q}}_j + I_j \ddot{\boldsymbol{q}}_j. \tag{B.4}$$

The moment of inertia of a 2D rigid body is constant, so that the equation simplifies to:

$$\dot{\boldsymbol{L}}_{ij} = \dot{\boldsymbol{r}}_{ij} \times \boldsymbol{p}_j + \boldsymbol{r}_{ij} \times \dot{\boldsymbol{p}}_j + I_j \ddot{\boldsymbol{q}}_j. \tag{B.5}$$

In the dynamical model, the positions of joint i and the center of mass of body j are defined by the vectors $\boldsymbol{P}_i$ and $\boldsymbol{G}_j$ respectively, see Figure B.2. Then,

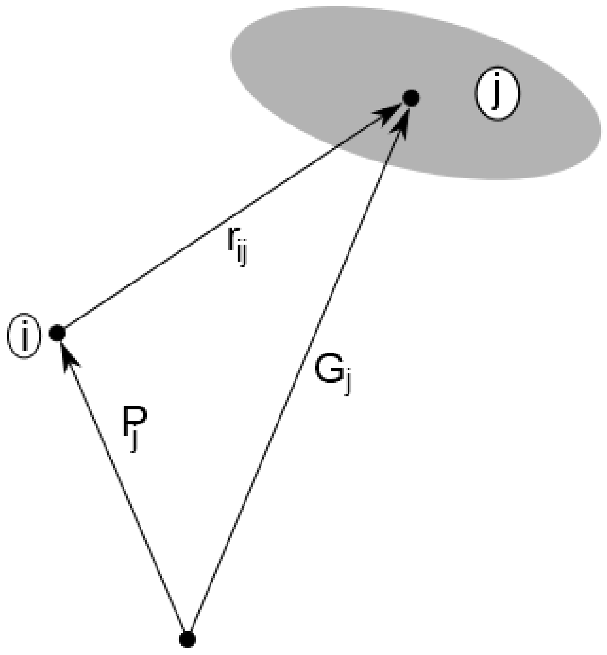


Figure B.2.

$$\begin{aligned} \boldsymbol{r}_{ij} &= \boldsymbol{G}_j - \boldsymbol{P}_i, \\ \boldsymbol{p}_j &= m_j \dot{\boldsymbol{G}}_j. \end{aligned} \tag{B.6}$$

We express the angular momentum balance relative to joint i. Therefore, its position is fixed in this frame and $\dot{\boldsymbol{r}}_{ij}$ becomes:

$$\dot{\boldsymbol{r}}_{ij} = \dot{\boldsymbol{G}}_j. \tag{B.7}$$

We also determine the time derivative of the linear momentum, which is given by:

$$\dot{\boldsymbol{p}}_j = m_j \ddot{\boldsymbol{G}}_j. \tag{B.8}$$

Filling in Equations (B.6), (B.7) and (B.8) in (B.5) results in:

$$\dot{\boldsymbol{L}}_{ij} = \dot{\boldsymbol{G}}_j \times (m_j \dot{\boldsymbol{G}}_j) + (\boldsymbol{G}_j - \boldsymbol{P}_i) \times (m_j \ddot{\boldsymbol{G}}_j) + I_j \ddot{\boldsymbol{q}}_j. \tag{B.9}$$

We see that the first term drops, so that we arrive at the following expression:

$$\dot{\boldsymbol{L}}_{ij} = (\boldsymbol{G}_j - \boldsymbol{P}_i) \times (m_j \ddot{\boldsymbol{G}}_j) + I_j \ddot{\boldsymbol{q}}_j \tag{B.10}$$

On the left hand side of Equation (B.1), we have to account for two torques. First, we express the control torque $\boldsymbol{u}_i$ as a vector with the same direction as $\boldsymbol{q}_i$ and magnitude u_i. Second, we need to account for gravity working on all children of joint i, resulting in the total torque:

$$\sum \boldsymbol{\tau}_i = \boldsymbol{u}_i + \sum_{j \in \mathcal{C}_i} \boldsymbol{\tau}_{ij}. \tag{B.11}$$

The gravitational torque $\boldsymbol{\tau}_{ij}$ for body j is found by taking the cross product of $\boldsymbol{r}_{ij}$ with vector $\boldsymbol{g}$, which always points downwards, and multiplying with the mass, see Figure B.3:

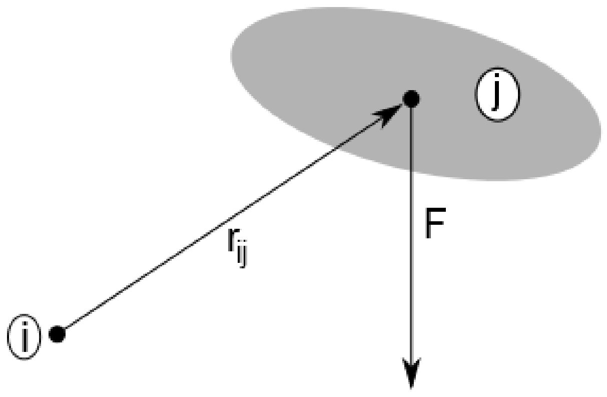


Figure B.3.

$$\boldsymbol{\tau}_{ij} = (\boldsymbol{G}_j - \boldsymbol{P}_i) \times (m_j \boldsymbol{g}) \tag{B.12}$$

We combine this with (B.11) and (B.10) to arrive at the full expression of the angular momentum balance for joint i:

$$\boldsymbol{u}_i + \sum_{j \in \mathcal{C}_i} (\boldsymbol{G}_j - \boldsymbol{P}_i) \times (m_j \boldsymbol{g}) = \sum_{j \in \mathcal{C}_i} \Big((\boldsymbol{G}_j - \boldsymbol{P}_i) \times (m_j \ddot{\boldsymbol{G}}_j) + I_j \ddot{\boldsymbol{q}}_j \Big) \tag{B.13}$$

This expression is set up for all five degrees of freedom of the system. These equations are then combined and solved for the joint accelerations $\ddot{\boldsymbol{q}}$, so that the dynamics of the system are written in their most general form as:

$$\boldsymbol{\mathcal{M}}(\boldsymbol{q})\ddot{\boldsymbol{q}} = \boldsymbol{\mathcal{F}}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}). \tag{B.14}$$