ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

AUTOMATIC CONTROL LABORATORY

Dimitri Eckert

A Parametrized, Dynamically Balanced Kick for NAO Robots

Semester Thesis

Automatic Control Laboratory
Swiss Federal Institute of Technology (ETH) Zurich

Supervision

Benjamin Flamm
Alexandros Tanzanakis
Prof. Dr. John Lygeros

July 2019

Acknowledgment

The author would like to thank his supervisors Benjamin Flamm and Alexandros Tanzanakis for the continuous support and for keep pushing him through difficulties. Special thanks goes to Nora Eckert for helping with the formatting of the images and to Karina Schmidlin for spell checking. Finally, the author would like to express his gratitude to Professor Lygeros for enabling the RoboCup Team at ETHZ.

Abstract

This thesis tries to generate a stable kick according to the ball position before the kick and the desired kick direction and speed at impact. The proposed algorithm takes this as an input. Next it generates a discrete trajectory for the foot which fulfills the constraints given from the input. Additionally, we introduce constraints ensuring that the foot does not collide with the supporting leg or the floor and does not try to reach a position which is unreachable due to the finite length of the leg. This trajectory is then followed by calculating the necessary joint angles while making sure that the horizontal contact forces and the contact moments at the supporting foot vanish. This last condition ensures stability. Since the nonlinear relation between the angular acceleration of the joints and the foot acceleration is linearized, some error is induced. This is compensated by updating the actual foot position and the necessary acceleration to stay on track at every time step.

Contents

Chapter 1

Introduction

1.1 RoboCup

RoboCup is an annually returning robotic soccer competition initiated in 1995. In the Standard Platform League all teams use the same hardware platform humanoid NAO from Aldebaran Robotics, allowing the teams to concentrate on software development rather than the mechanics of robots. The robots are fully autonomous. During a game, no remote control is allowed - neither by humans nor by computers.

1.2 Motivation

The kicking movement is probably the most important one in any soccer game. It is an absolute necessity for passing and for scoring a goal if one does not want to dribble the whole way there. Since the ball is able to move much faster than a player, being able to shoot fast, accurately and adaptable to different situations is a huge advantage in a soccer game. The current kick of the Nomadz RoboCup team of ETHZ does not fulfill these requirements. It is a hard-coded kick which is strong but can shoot only straight ahead with fixed strength. Hence a long process of alignment previous to the kick is necessary. During this time the opponent team has ample time to take the ball away from the robot. Also in order to be able to pass exact target positions have to be reached. This necessitates variable kick strength. Consequentially the aim of this thesis is to implement a kicking engine that produces kicks with variable kick strength and direction.

1.3 Structure of Thesis

This thesis is structured as follows: First a general background of the work done on the topic of robotic kicking by the Nomadz team and researchers is presented. Second the approach proposed in this thesis is outlined. Third the found results are described. And finally, the conclusion drawn from the results is summarized and an outlook on future work is presented.

Chapter 2

Background

2.1 Previous Work within the Nomadz Team

In this first part of this chapter work done in previous projects will be elaborated on.

Ivo de Concini and Konrad Schieban [1] implement a tool for visual feedback of the center of mass. They develop a parametrized kick without closed loop feedback. It consists of five stages:

1. Check Ball: Check if the Ball is in the reachable area
2. Move to Start Position: Move to a standard position from which it is possible to perform a stable kick
3. Shift Weight: Shift weight to the supporting leg
4. Swing Leg: Swing the leg back and forth again to kick the ball
5. Get Back: Get back to a stable position from which it is possible to perform another kick

First, they develop a static kick by recording joint angles at the desired positions. Then they create the complete motion by interpolating the angles between the recorded points. Then they parametrize this static kick. For this they look at the physical limits and derive the limits on the parameters thereof. They introduce timing parameters which define the time for one interpolation step. The roll joint in the hip is used to kick to the left and right (from back left to front right to kick to the right and vice versa) while it is made sure that the kicking foot does not touch the supporting leg. They implement three kicks: a customizable kick, a slow straight kick usable for passes and a kick that kicks straight forward with maximum power. They finally conclude that their kick is stable and reliable but for stronger shots and better stability closed loop feedback is necessary.

In the semester thesis of Dany Manickathu [2] inverse kinematics are used. From the desired limb position the needed joint position are calculated. Stabilization is done with a closed loop PID controller which is tilting and panning the torso such that the projection of the center of mass is within the supporting area. The supporting area is defined by the supporting foot when the robot is standing on one leg or both feet when both feet are on the ground. The PID parameter are retrieved experimentally. The derivative term is small due to big jumps which occur at transitions when the robot switches from one leg stand to two leg stand and vice versa. If it was big the jumps at the transition would lead to big jumps in the controller output. The strength is varied by changing the time for the part of the foot motion when the impact with the ball takes place. The problem of getting the center of mass within the supporting area of one foot is solved by commanding the feet to move to the opposite side. Since the feet stay in place due to friction with the floor the torso will shift to the desired side. The kick is split into intermediate positions. The

motion in between is interpolated using cubic splines. They manage to implement a straight kick which consists of five phases and takes about 2.2 seconds.

Martin Kälin [3] introduces a kick that happens while the ball is still moving. It is supposed to be able to kick to any position on the field. The ball trajectory is model based. For the design of this model the vicon system was used. This is a system of cameras recording the happening on the field from the ceiling and providing perfect knowledge of the surroundings. The resulting kick is suitable for distances of up to half a field in the simulation environment. For higher kick strength, the accuracy decreases significantly. Problems that arise in this project are that meanwhile the balls used in the matches of RoboCup have changed and that the approach was not stable on the real robots.

A different approach is tried out by Marc Heim [4]. He chooses to do no manual trajectory crafting. The kick should be able to adapt to different kick point positions, kick directions and kick speeds. This while maximizing stability by minimizing the torque on the supporting foot. Friction is supposed to compensate for any inaccuracies and disturbances. This approach is called zero torque policy. This makes use of the fact that if the moments on the supporting foot vanish the angular momentum of the whole body cannot change and the robot will be stable. Because the foot sensors are not accurate or partly broken they are not used. The model is linear and therefore easier to incorporate into optimization problems. There has no testing been done on the real robots and the precise range of possible kicking motions has not yet been defined.

These projects have never been completed up to a state where they are fully functional on the real NAO robots. The currently implemented kick is the hard-coded unparameterized kick done by Concini and Schieban. 2.1 This kick is stable and very strong. But the robot needs to perfectly align behind the ball before kicking since it can not adapt the direction of the kick.

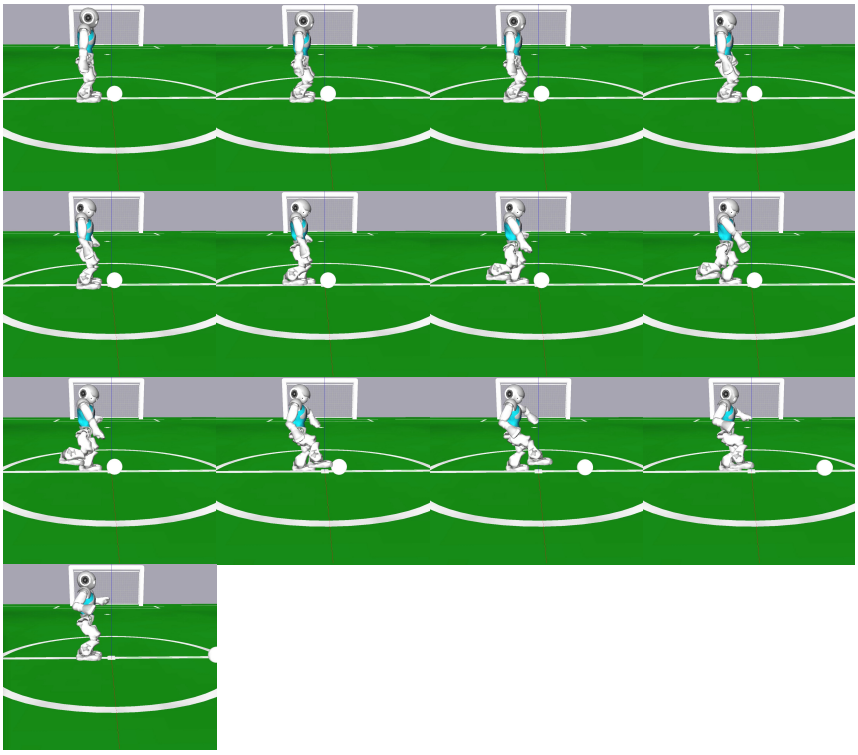

Figure 2.1: Currently implemented, hard-coded kick. The kick is strong and stable, but it cannot be adapted to different ball positions or kick directions and kick strengths.

2.2 Scientific Research

In the second part of this chapter approaches from scientific papers for balancing methods, trajectory generation and kicking motions of humanoid robots will be presented.

Kajita et al. [5] present a biped walking pattern generator that enables an additional Zero-Moment-Point (ZMP) control. This auxiliary ZMP is accomplished by an inverse system added to a pattern generator based on the ZMP preview control. A virtual time shifting of the reference ZMP is used in order to compensate the effect of the auxiliary ZMP. A simulation of a walking control on uneven terrain serves as a trial situation. By changing its walking speed the robot can walk successfully in the simulation.

In the paper of Vukobratović and Borovac [6] a general overview of the history, the concept and a special review concerning boundary cases is given. Boundary cases are present when the ZMP is close to the edge of the support polygon. Furthermore fictious cases are discussed, when the ZMP should be outside the support polygon. The exact difference between ZMP and the center of pressure is explained. Finally, they elaborate on some phenomena that need further work and that may yield a significant improvement in robot performance.

A comparison between different kick trajectory optimizations is done by Peña [7] They look at three different approaches: cubic splines, cubic Hermite splines and sextic polynomials. By means of the Webots simulator the trajectories are optimized and then compared to each other. They concluded that these joint interpolators are useful to get powerful kicks, while the cubic spline showed the fastest convergence.

Pholdee and Bureerat [8] use multiobjective meta-heuristics (MOMHs) in order to solve multi-objective time-jerk robot trajectory planning problems. Their goal is to minimize traveling time and jerk while following constraints on velocity, acceleration and jerk. Among the 8 MOMHs, the Two-Arch2 and DEMO are said to be the top two performers for such a problem.

The paper of Hehn and D'Andrea [9] presents an algorithm that allows the computation of interception maneuvers for quadrocopters. With Pontryagin's minimum principle it is shown that the interception maneuver minimizing the time to rest after the interception is identical to the time-optimal maneuver that leads the vehicle to the position at which it rests after the interception. This fact is taken advantage of to apply already developed, computationally efficient approaches for the calculation of interception maneuvers. A computationally lightweight trajectory generation algorithm is received, that allows to be used implicitly as a feedback law by working out the trajectory again at each controller update. In an experiment letting the quadrocopters inercept balls in mid-flight the effectiveness of the approach is demonstrated. The real-time trajectory generation allows for changes in the anticipated flight path of the ball at each controller update.

A dynamic kick with arbitrary direction is developed by Peña et al. [10] . Generated using cubic splines, the trajectories are carried out while dynamically balancing on one foot. They too use ZMP for balancing using a preview controller to minimize the ZMP error. For the optimization of the kick parameters they propose a framework based on the Webots simulator. With this approach, optimal parameters were found and then tested on the real robots for verification.

Xu and Mellmann [11] propose an online computed motion control able to kick a ball in different positions and into different directions. The kick develops as follows: First the weight is shifted to one leg, then the kicking foot is retracted, then the kick is executed and last the robot returns to the initial position. The reachable space is defined by some basic constraints that the robot needs to fulfill. The trajectory of the foot is chosen to be the shortest path in the reachable space. For simplicity, the rotation of the foot is neglected. The body inclination is then adapted such that static stability is given. The joint evolution is generated with inverse kinematics. The adaptivity

is proven in experiments.

Arne Böckmann and Tim Laue [12] use the zero torque policy for balancing. For simplicity, they decide that it is sufficient to keep the foot level to the ground and restrain the foot from rotating. Dynamic movement primitives are used to imitate arbitrary kicking movements. The whole process is split into two systems. The canonical system: variable s replaces t, goes from 1 to 0. The transformation system acts like a PD controller plus a forcing term that makes sure the shape of the trajectory is followed. If the goal position changes the shape "bends" to reach it. Since the velocity also matters the position, velocity and acceleration are taken into account. Problems arise when the target velocity changes. Then the shape changes extremely and becomes up to impossible. This is alleviated by multiply the forcing with a factor that gets bigger when the target velocity increases and gets smaller when the target velocity decreases. The dynamic movement primitives adapt very well to changes in the goal position or velocity without leading to discontinuities.

Chapter 3

Methods

3.1 Overview

In this chapter, the general setup of the proposed kicking engine is explained. 3.1

The input for the kicking engine is the position of the ball and the desired foot velocity (speed and direction) at impact with the ball. The output are the necessary joint angles to achieve this target with a physically meaningful motion while keeping balance.

This task is split into two parts. In the first part a trajectory is calculated that achieves the desired velocity at the target position while obeying constraints on the spatial reachability and the feasible accelerations. The second part takes this trajectory as an input and calculates the necessary joint angles are calculated which lead to the wanted foot motion and keep balance while abiding to constraints on the joint angles, the joint torques and the general dynamical behavior of the robot. For this Jacobians are necessary to change from the end effector (in this case the end effector is the kicking foot) space to the joint space. This mapping is nonlinear. Therefore, some error is introduced over a time step. To counteract this a feedback loop is introduced. From the obtained joint angles after the time step the resulting foot position is calculated. This is then used to recalculate the necessary foot acceleration to get back to the trajectory within the next time step.
Both optimization problems are solved using the Matlab-based modeling system for convex optimization CVX [13] .

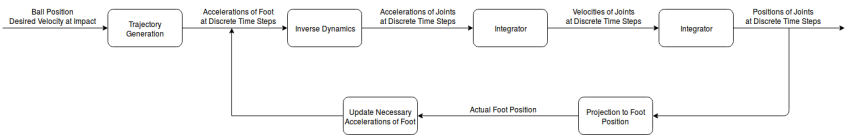

Figure 3.1: General Overview of the whole procedure

The following sections will elaborate on these intermediate steps.

3.2 Trajectory Generation

First a trajectory needs to be found. The input for this algorithm are the target position and velocity as well as the initial position and velocity. The whole trajectory from initial position to target position is discretized in to 10 time steps of length 0.1 seconds. The acceleration over one time step is assumed to be constant. Hence, the trajectory evolves over one step according to the

following equations

$$x(i+1) = x(i) + v(i) \cdot dt + 0.5 \cdot a(i) \cdot dt^2 \tag{3.1}$$

$$v(i+1) = v(i) + a(i) \cdot dt \tag{3.2}$$

Where i indicates the state before the time step and $i+1$ the state after the time step. x, v and a are the current position, velocity and acceleration of the foot in 3D space respectively. dt is the length of the time step.
This can be rewritten as follows

$$s(i+1) = A \cdot s(i) + b \cdot a \tag{3.3}$$

Where:

$$s = \begin{bmatrix} x \\ v \end{bmatrix} \tag{3.4}$$

$$A = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} \tag{3.5}$$

$$b = \begin{bmatrix} 0.5 \cdot dt^2 \\ dt \end{bmatrix} \tag{3.6}$$

This system is then solved using the matlab-based modeling system for convex optimization CVX.

To ensure that the solution has a better chance of being feasible some constraints have to be met. First the foot should not hit the floor. Therefore, the z coordinate can never be negative, where a safety margin of 5mm has been added. On the other side, there is a maximum height constraint. This was found heuristically by measuring the height of the foot when it pulls it as close to the torso as possible which was 100mm. Additionally one wants to prevent the kicking foot to hit the supporting leg. Thence, the two feet are constrained to have at least distance of 105mm in y direction between them. In order to account for the finite length of the leg the trajectory is bound to be at most 290mm – the length of the stretched ouleg - away from the initial position of the kicking hip. In order to prevent impossible accelerations, the maximum acceleration of the foot is set to be $1\,^{\mathrm{m}}/_{\mathrm{s}}$. This is a conservative estimate of what the robot is able to perform. A more sophisticated solution could be a next step in order to push the robot more towards the boundaries of what is feasible.

So as to find the fastest possible trajectory a binary search is used. It divides the possible number of time steps in to two intervals and tries to find a solution with the middle element of all possibilities. If a solution is found the higher interval is deleted and the algorithm starts anew with the remaining interval. If no solution is found the lower interval is deleted and the higher interval serves as the starting point of the new problem. 3.2

After three iterations, the fastest solution is chosen. Now the same thing has to be done for the way back to the initial position. Here the same approach is used the only difference is that the initial and target position and velocity are switched. It might seem easier to just reverse the found trajectory found for the first part. This is not always applicable, because for getting back first the foot has to decelerate before it can go back the same way. This is critical because in this part of the trajectory it is quite likely that one of the spatial constraints is violated.

3.3 Inverse Dynamics

Inverse dynamics is a method to calculate the necessary joint angle accelerations of a robot in order to satisfy a certain end effector trajectory, to fulfill constraints on external forces or achieve other

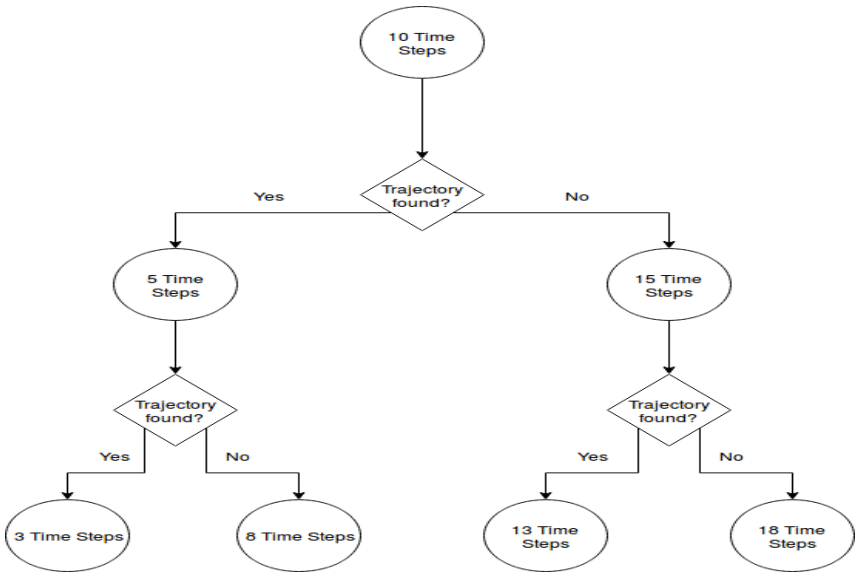

Figure 3.2: Procedure of the binary search.

objectives. As the name indicates as opposed to inverse kinematics it takes the dynamical behavior of the robot into account and therefore a precise modeling of the masses and inertias is necessary. The data for that is found on the official webpage of the manufacturer Softbank Robotics [14]. The project has been done for the V5 versions of the NAO robots.
The approach chosen here is mainly based on the course "Robot Dynamics" taught by Marco Hutter at ETHZ [15].

The Nao robots are composed of 25 rigid parts connected by 24 joints each of them actuated by an electrical motor except for the pelvis joint where both the joint of the right and the left leg are connected to one motor. This leaves us with 23 degrees of freedom (see 3.3).

We introduce a different coordinate system (CS) for every part of the robot. The root of these coordinate systems is placed in the joint connecting the part to the rest of the robot. The following values are all described in the parts own coordinate system.
The following values need to be calculated or looked up in the official documentation of Softbank Robotics:

$M \in \mathbb{R}$ mass [kg]

$P_{com} \in \mathbb{R}^3$ position of center of mass [m]

$I \in \mathbb{R}^{3\times3}$ inertia tensor $[\text{kg} \cdot \text{m}^2]$

$R \in \mathbb{R}^3$ rotational axis of the joint [dimensionless]

$T \in \mathbb{R}^{4\times4}$ transformation from one CS to another CS [mixed units]

3.3.1 Projected Newton-Euler

Since during the whole process of kicking the supporting foot stays on the ground and is supposed to stand still, the author chose for simplicity to use a fixed base model with the bottom of the

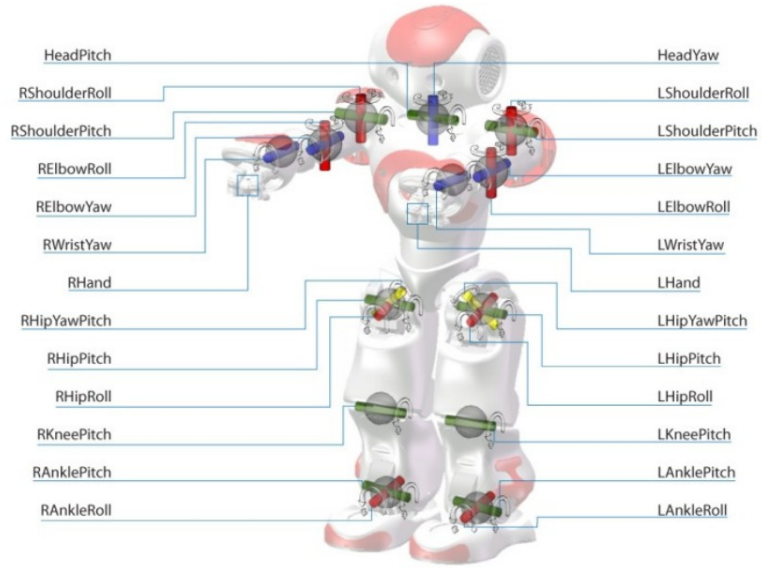

Figure 3.3: NAO robot with all its joints. The two diagonal joints LHipYawPitch and RHipYaw-Pitch are connected to one motor and therefore not independent.

standing foot as the origin of the inertial world frame. The modeling of the dynamical behavior was completed with the projected Newton-Euler method. In the projected Newton-Euler method the movements are all projected into the generalized coordinate space. Here the generalized coordinate space is the space of the possible angles of the joints of the robot. The big advantage of this approach is that every configuration (within the angular limits of the joints and the limits of the joint torques) is automatically possible.
The equation of motion has the following form:

$$M(q) \cdot \ddot{q} + b(q, \dot{q}) + g(q) = \tau \tag{3.7}$$

Where:

$M(q) \in \mathbb{R}^{n_q \times n_q}$ mass matrix [kg]

$q \in \mathbb{R}^{n_q}$ generalized coordinates [rad]

$\dot{q} \in \mathbb{R}^{n_q}$ generalized velocities [rad/s]

$\ddot{q} \in \mathbb{R}^{n_q}$ generalized accelerations [rad/s²]

$b(q, u) \in \mathbb{R}^{n_q}$ coriolis and centrifugal terms [kg·rad/s²]

$S \in \mathbb{R}^{n_\tau \times n_q}$ selection matrix of actuated joints [dimensionless]

$\tau \in \mathbb{R}^{n_\tau}$ joint torques [kg·rad/s²]

$n_q \in \mathbb{R}$ number of joints

$n_\tau \in \mathbb{R}$ number of actuated joints

For the case of the kicking robot the only forces acting on it are either from the impact with the ball and due to the lightness of the ball negligibly small or act on the standing foot and therefore do not show up in the equation of motion since the foot was chosen to be the fixed base. All the quantities in this formula need to be mapped from the intertial cartesian space to the generalized coordinate space and vice versa. This is done with jacobians. A jacobian J is a Matrix with the following properties:

$$m = J \cdot \ddot{q} \tag{3.8}$$

Where:

$m \in \mathbb{R}^n$ arbitrary value which is linearly dependent on $\ddot{q}$

If J maps from the joint velocities to the velocity of a point P in cartesian space then it is called a positional Jacobian J_{pos} and is dependent on the joint configuration q:

$$v_P = J(q) \cdot \ddot{q} \tag{3.9}$$

Where:

$v_P \in \mathbb{R}^3$ velocity of point P in cartesian space $[\mathrm{m/s}]$

With a positional jacobian J_{pos} a projection of an applied force at point P onto the generalized space can be done as follows:

$$F_{gen} = J_{pos}^T \cdot F_P \tag{3.10}$$

Where:

$F_P \in \mathbb{R}^3$ force in cartesian space at point P $[\mathrm{kg \cdot m/s^2}]$

$F_{gen} \in \mathbb{R}^{n_q}$ force in generalized space $[\mathrm{kg \cdot rad/s^2}]$

$J_{pos_P} \in \mathbb{R}^{3 \times n_q}$ positional jacobian for point P $[\mathrm{m/rad}]$

With rotational jacobians the same operations can be done for rotations.

3.3.2 Acceleration

For the calculation of accelerations one has to take the time derivative of the equation for velocities found above:

$$\dot{v}_{eff} = \frac{d}{dt}(J \cdot \dot{q}) = \dot{J} \cdot \dot{q} + J \cdot \ddot{q} \tag{3.11}$$

The time derivative $\dot{J}$ of the jacobian is non zero since the model configuration changes with q which changes with time. $a_{pure} = \dot{J} \cdot \dot{q}$ is equal to the acceleration of a point when there is no acceleration in the generalized coordinate space but the point still experiences acceleration due to the evolution of the whole robot with a constant generalized speed. This term accounts for the centripetal acceleration in the positional and for the coriolis acceleration in the rotational case.

3.3.3 Computing Jacobians

In order to be able to do the following computations one needs the affine transformation matrices that describe the transformation from the coordinate system of every body part to the inertial world coordinate system.
The absolute transformation matrices are computed according to the following pseudo code:

$T_{abs} = \mathbb{I}$
for $i \in jointsBetweenWorldAndBodypart$ **do**
  $T_{abs} \leftarrow T_{abs} \cdot computeJointTransform(q(i))$
end for
return T_{abs}

Where:

- T_{abs} is the transformation from the joints coordinate frame to the inertial world frame.
- i is the state index of the joint (position in the list q where the joints positional state is stored).
- computeJointTransform(α) gives back the homogeneous transformation of the joint currently examined dependent on the joint angle α. This is a rotation around a specified rotation axis and a translation from the root of the first coordinate system to the root of the second coordinate system e.g. for a rotation around the x-axis and a translation of 0.1 m in y-direction of the first coordinate system:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) & 0.1 \\ 0 & sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

Positional Jacobians

A positional jacobian delivers the speed of a point as a linear function on the joint speed. The velocity of a linkage k is given by

$$\dot{r}_{Ik} = \dot{r}_{I(k-1)} + \omega_{I(k-1)} \times r_{(k-1)k} \tag{3.13}$$

When denoting the base frame as 0 and the end-effector frame as $n+1$, the end-effector velocity can be written as

$$\dot{r}_{IE} = \sum_{k=1}^{n} \omega_{Ik} \times r_{k(k+1)} \tag{3.14}$$

Using n_k to represent the rotation axis of joint k such that

$$\omega_{(k-1)k} = n_k \cdot \dot{q}_k \tag{3.15}$$

and

$$\omega_{Ik} = \omega_{I(k-1)} + \omega_{(k-1)k} \tag{3.16}$$

the angular velocity of body k can be written as

$$\omega_{Ik} = \sum_{i=1}^{k} n_i \cdot \dot{q}_i \tag{3.17}$$

Substituting this in and rearranging terms results to

$$\dot{r}_{IE} = \sum_{i=1}^{k} n_k \cdot \dot{q}_k \times r_{k(n+1)} \tag{3.18}$$

Bringing this in matrix formulation yields the following equation:

$$\dot{r}_{IE} = \begin{bmatrix} n_1 \times r_{1(n+1)} & n_2 \times r_{2(n+1)} & ... & n_n \times r_{n(n+1)} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ ... \\ \dot{q}_n \end{bmatrix} \tag{3.19}$$

Rotational Jacobian

For rotation we receive the following equation:

$$\omega_{IE} = \sum_{i=1}^{n} n_i \cdot \dot{q}_i = \begin{bmatrix} n_1 & n_2 & ... & n_n \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ ... \\ \dot{q}_n \end{bmatrix} \tag{3.20}$$

Combining these two expressions yields the combined geometric Jacobian:
In this formulation, the rotation axis is given by

$$J = \begin{bmatrix} J_P \\ J_R \end{bmatrix} = \begin{bmatrix} n_1 \times r_{1(n+1)} & n_2 \times r_{2(n+1)} & ... & n_n \times r_{n(n+1)} \\ n_1 & n_2 & ... & n_n \end{bmatrix} \tag{3.21}$$

3.3.4 Optimization Problem

Now we write down the dynamics of the whole system as an equation of motion in the joint space as was shown in equation 3.7. This equation has to be fulfilled to make the whole movement physically feasible

$$M(q) \cdot \ddot{q} + b(q, \dot{q}) + g(q) = \tau \tag{3.22}$$

The calculation of these terms is rather tedious and is explained in the following section.

Gravity

The gravity Term $g(q)$ depends on the current state configuration q. The gravitational force just has to be projected onto the generalized space:

$$g(q) = \sum_{i=0}^{n_b} J_{com}^T \cdot a_g \cdot M_i \tag{3.23}$$

Where:

$a_g \in \mathbb{R}^3$ gravitational acceleration $[\mathrm{m/s^2}]$

$J_{com}^T \in \mathbb{R}^{3 \times n_q}$ positional jacobian of the individual center of mass of the robot part

Centrifugal and Coriolis Term

The coriolis and centrifugal term $b(q, \dot{q})$ depends on the joint angles q as well as the joint velocities $\dot{q}$. It is made up of the following three parts:

- The centrifugal term of the center of masses:
  This corresponds to the force that would be exerted on the center of mass when $\ddot{q}$ is zero but there remains some force to keep it on the specified trajectory. The term for the force F_c on an individual body part:

  $$F_c = J_{P_i}^T \cdot m_i \cdot \dot{J}_{P_i} \cdot \dot{q} \tag{3.24}$$

  Where:

  $J_{P_i} \in \mathbb{R}^{3 \times n_q}$ positional jacobian at center of mass of body part i

  $m_i \in \mathbb{R}$ mass of body part i [kg]

- The rotational pendant for F_c is the Euler Force F_eul that is induced by keeping the body on the trajectory induced by the constant angle velocities:

  $$F_{eul} = J_{R_i}^T \cdot I_i \cdot \dot{J}_{R_i} \cdot \dot{q} \tag{3.25}$$

  Where:

  $J_{R_i} \in \mathbb{R}^{3 n_q}$ rotational jacobian of body part i

  $I_i \in \mathbb{R}^{3 \times 3}$ inertia tensor of body part i $[\mathrm{m}^2 \cdot \mathrm{kg}]$

- The coriolis term which stems from the transformation of newtons laws from an inertial frame to a rotating frame:

  $$F_{cor} = J_{R_i}^T \cdot (\Omega_i \times I_i \cdot \Omega_i) \tag{3.26}$$

  Where:

  $\Omega_i \in \mathbb{R}^3$ angular velocity vector of body part i $[\mathrm{rad}/\mathrm{s}]$

  While Ω_i can be obtained by:

  $$\Omega_i = J_{R_i} \cdot \dot{q} \tag{3.27}$$

All three terms are then projected back onto generalized coordinate space and summed up for all body parts:

$$b(q, \dot{q}) = J_{P_i}^T \cdot m_i \cdot \dot{J}_{P_i} \cdot \dot{q} + J_{R_i}^T \cdot I_i \cdot \dot{J}_{R_i} \cdot \dot{q} + J_{R_i}^T \cdot (J_{R_i} \cdot \dot{q} \times I_i \cdot J_{R_i} \cdot \dot{q}) \tag{3.28}$$

Mass Matrix

The mass matrix $M(q)$ is responsible for the relation between the joint space acceleration and induced joint forces stemming from the inertia of the body parts. $M(q) \cdot \ddot{q}$ should give the joint force dependent on a given acceleration.
The basic formulas for positional and rotational acceleration to force relationships:

$$F = m \cdot a \tag{3.29}$$

$$\tau = I \cdot \alpha \tag{3.30}$$

These can then be used to get the expected resulting force of $F_a = M(q) \cdot \ddot{q}$ (already projected onto joint torque space):

$$F_a = \sum_{i=1}^{n_b} J_{P_i}^T \cdot M_i \cdot a_i + J_{R_i}^T \cdot I_i \cdot \alpha_i \tag{3.31}$$

Where:

$a_i \in \mathbb{R}^3$ acceleration of the body part i $[\mathrm{m/s^2}]$

$\alpha_i \in \mathbb{R}^3$ angular acceleration of the body part i $[\mathrm{rad/s^2}]$

$n_b \in \mathbb{R}$ number of body parts

a and α can be obtain with the help of the already used positional and rotational Jacobians:

$$F_a = M(q) \cdot \ddot{q} = \sum_{i=1}^{n_b} J_{P_i}^T \cdot M_i \cdot J_{P_i} \cdot \ddot{q} + J_{R_i}^T \cdot I_i \cdot J_{R_i} \cdot \ddot{q} \tag{3.32}$$

Which leads to:

$$M(q) = \sum_{i=1}^{n_b} J_{P_i}^T \cdot M_i \cdot J_{P_i} + J_{R_i}^T \cdot I_i \cdot J_{R_i} \tag{3.33}$$

Task related constraints

The task the robot needs to fulfill is to follow the precalculated trajectory with the kicking foot while balancing on the supporting foot and without violating the constraints on the joint angles and the joint torques. For this we use task-space dynamics control. We formulate our tasks in form of the following equations:

$$\dot{w}_{des} = \dot{J}_{kickfoot} \cdot \dot{q} + J_{kickfoot} \cdot \ddot{q} \tag{3.34}$$

Where $\dot{w}_{des}$ is the desired acceleration of the foot, $J_{kickfoot}$ is the jacobian mapping the joint angle velocities to the foot velocities of the kicking foot. This equation is obtained by taking the derivative of the foot acceleration in world space and in joint space.

$$w_{des} = J_{kickfoot} \cdot \dot{q} \tag{3.35}$$

Now for the balancing objective the task can be formulated as follows:

$$\sum_{i=1}^{n_b}((\dot{J}_i \cdot \dot{q} + J_i \cdot \ddot{q}) + g) \cdot m_i \times ri = 0 \tag{3.36}$$

$$\sum_{i=1}^{n_b}(\dot{J}_i \cdot \dot{q} + J_i \cdot \ddot{q}))_{x,y} \cdot m_i = 0 \tag{3.37}$$

Where g is the gravitational acceleration, m_i is the mass of the i_{th} body part and r_i is the position of the i_{th} body part in the world frame.
The first equation expresses the fact that for stability all the moments induced by all the body parts on the supporting foot need to sum up to zero. If they do not the robot is at risk of tipping over.
The second equation expresses that in x and y direction there can be no force otherwise the robot would slide. In z direction, a force can be counteracted by the floor and actually needs to otherwise the robot would "fall through" the floor due to gravity.
Of course, small moments and forces can be compensated by the foot surface and friction between the foot and the floor. The closer to zero they are, the more robust the stability of the robot is.

These three constraints have to be fulfilled for the robot to do the desired kick. For the purpose of having as small motions as possible the euclidean norm of all the joint accelerations is minimized.

Furthermore, since the joints are limited in their range the search space for the optimization problem is limited to be within those boundaries. The same goes for the possible joint torques which are limited by the motors used in the NAO robots.

This leaves us with this quadratic program:

$$M(q) \cdot \ddot{q} + b(q, \dot{q}) + g(q) = \tau \tag{3.38}$$

$$\dot{w}_{des} = \dot{J}_{kickfoot} \cdot \dot{q} + J_{kickfoot} \cdot \ddot{q} \tag{3.39}$$

$$\sum_{i=1}^{n_b}((\dot{J}_i \cdot \dot{q} + J_i \cdot \ddot{q}) + g) \cdot m_i \times ri = 0 \tag{3.40}$$

$$\sum_{i=1}^{n_b}(\dot{J}_i \cdot \dot{q} + J_i \cdot \ddot{q}))_{x,y} \cdot m_i = 0 \tag{3.41}$$

$$\|\tau_i\| <= \tau_{max_i} \qquad \forall i \tag{3.42}$$

$$q_{min_i} <= q_i <= q_{max_i} \qquad \forall i \tag{3.43}$$

$$\min_{\ddot{q},\tau}(\|\ddot{q}\|_2) \tag{3.44}$$

This program is solved at every time step for the whole trajectory before actually executing the first step. The resulting joint accelerations are integrated twice over the time step to receive the joint angles reached after the time step. These joint angles are then fed back to the robot for all time steps at once. This procedure is chosen since the computation time is - at least at the current state – too big to be carried out simultaneously.

3.3.5 Feedback Control

The jacobians and the other terms used above are calculated before each time step using the joint angles and joint velocities before the time step. But since the joint angles and the joint velocities change during a time step these terms change as well. This introduces an error since the found joint accelerations do not satisfy the constraints above for the changed configuration at the end of the time step.
To compensate for this, we introduce a feedback loop (see figure 3.1). From the calculated joint angles using jacobians – calculated with the new joint angles – the resulting foot position can be retrieved. This updated foot position is then used to recalculate the necessary foot acceleration to get to the desired foot position after the next time step.

Chapter 4

Results

4.1 Trajectory Planning

The proposed linear program is able to calculate trajectories to achieve various different target positions and velocities (see figure 4.1, 4.2 and 4.3). The binary search makes an adjustment of the kick time possible. The possible range of kicking times - the time the foot takes from the initial position until impact - ranges from 0.3 seconds up to 2 seconds. This is a whole order of magnitude that can be won in kicking time. Meanwhile the positions further away from the initial position and bigger velocities remain reachable. The needed computation time is for the tested inputs between 25 and 186 seconds.

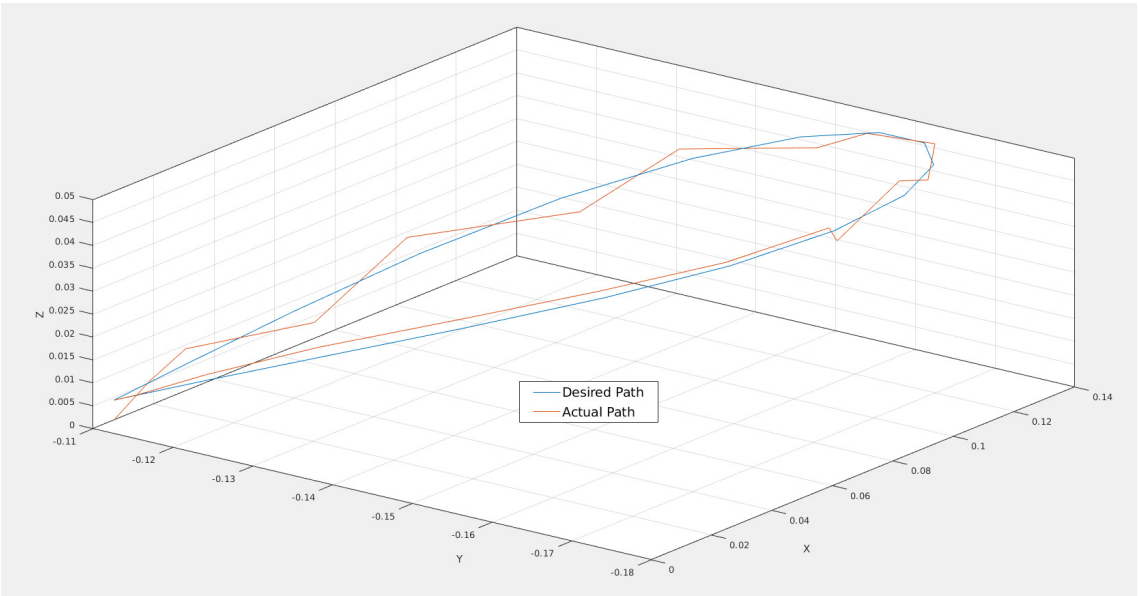

Figure 4.1: In blue the desired trajectory, in orange the actually achieved trajectory. The units are all in meters here. The target position is 8 cm in x-direction, -6cm in y-direction and 4 cm in z-direction from the initial position. The target velocity is at 0.3 $^{m}/_{s}$ in x-direction. The error in position is growing over time and gets bigger with increased velocity. Because of the feedback loop the error remains within the range of few milimeters.

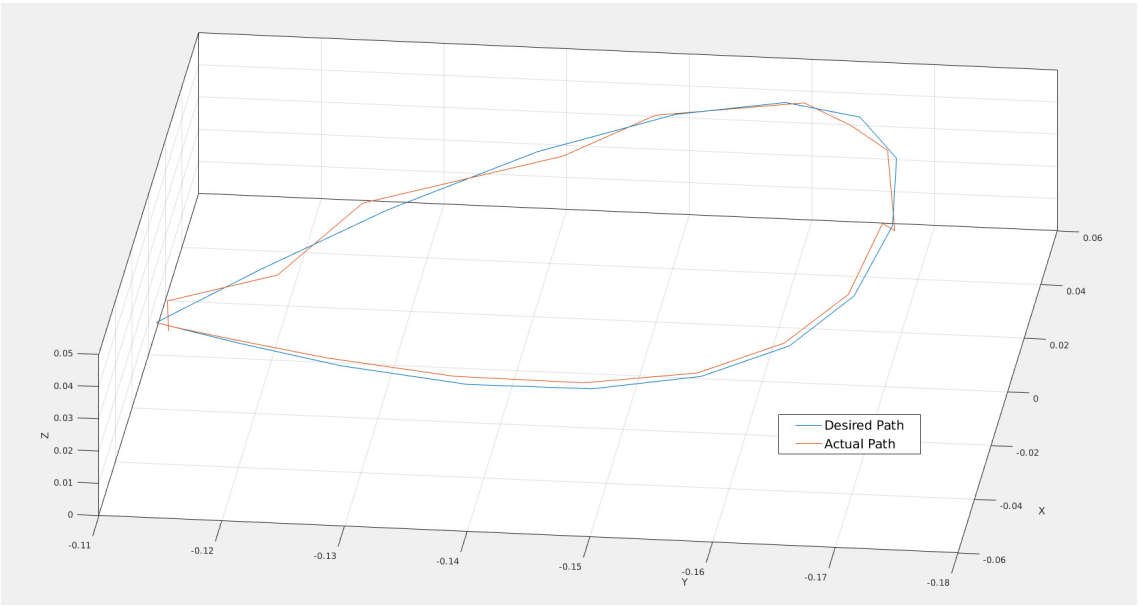

Figure 4.2: In blue the desired trajectory, in orange the actually achieved trajectory. The units are all in meters here. The target position is 0 cm in x-direction, -6cm in y-direction and 4 cm in z-direction from the initial position. The target velocity is at 0.3 $^{m}/_{s}$ in x-direction. The error in position is growing over time and gets bigger with increased velocity. Because of the feedback loop the error remains within the range of few milimeters.

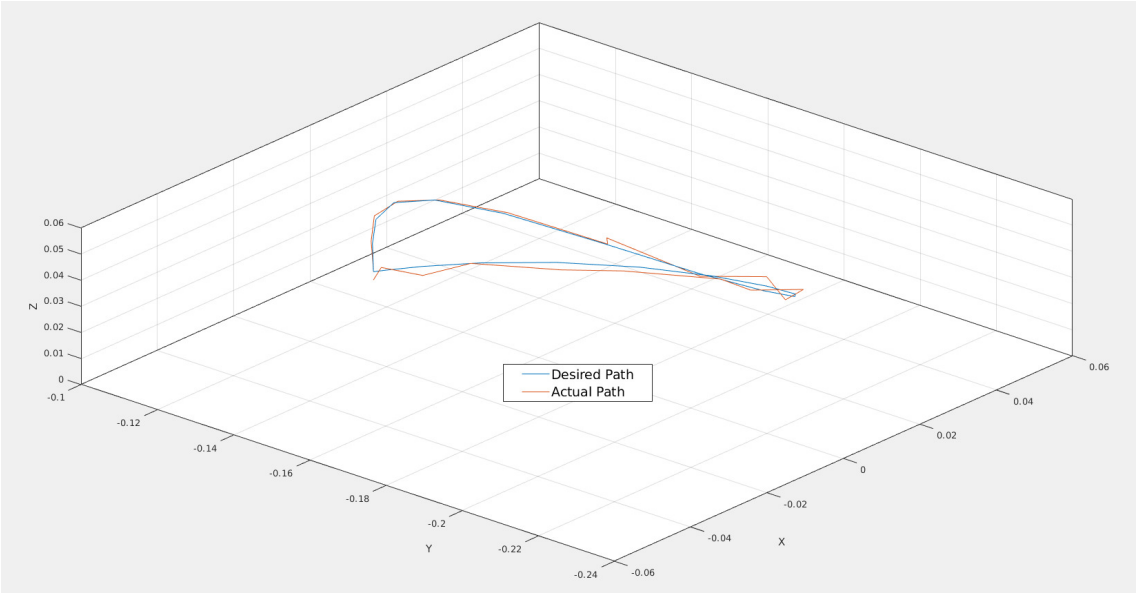

Figure 4.3: In blue the desired trajectory, in orange the actually achieved trajectory. The units are all in meters here. The target position is 0 cm in x-direction, -6cm in y-direction and 4 cm in z-direction from the initial position. The target velocity is at 0.3 $^{m}/_{s}$ in y-direction. The error in position is growing over time and gets bigger with increased velocity. Because of the feedback loop the error remains within the range of few milimeters.

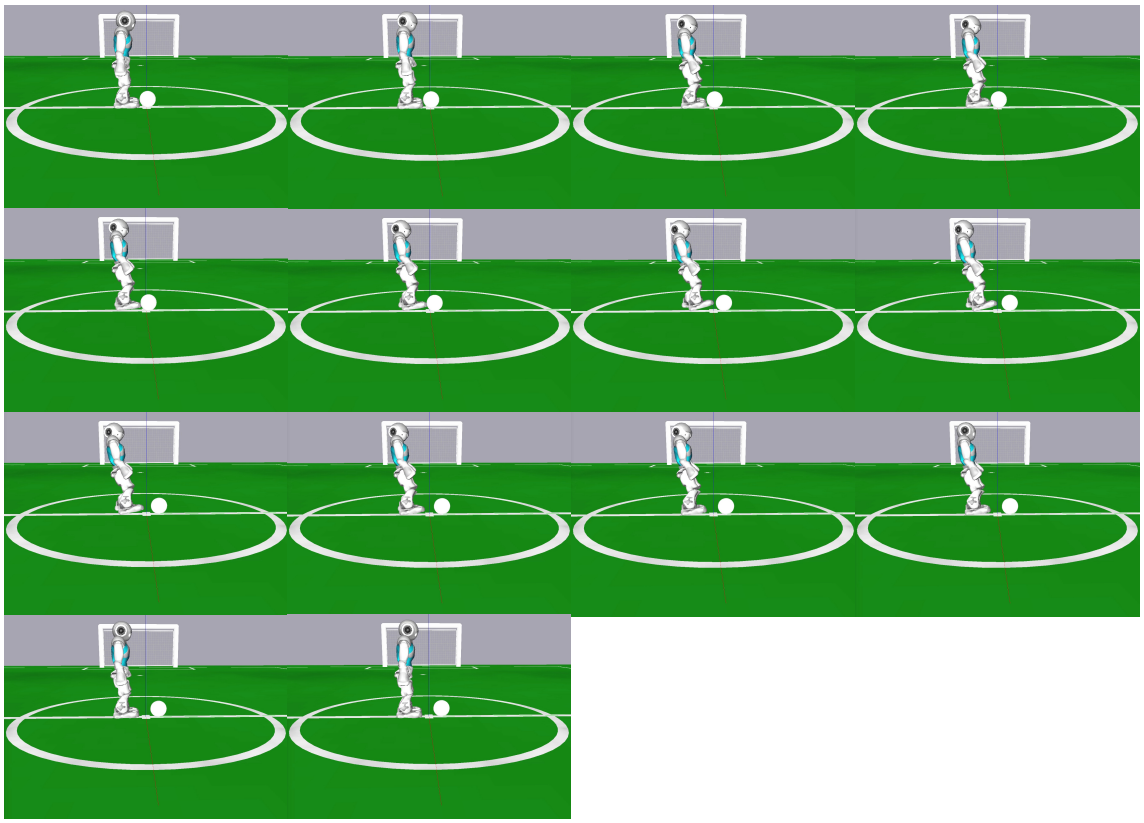

Figure 4.4: A kick simulated in SimRobot [16]. The target position is 3 cm in x-direction, 0 cm in y-direction and 0 cm in z-direction from the initial position. The target velocity is at 0.2 $^{m}/_{s}$ in x-direction.

4.2 Trajectory Following

The proposed quadratic program is able to produce joint angle accelerations that follow the commanded foot trajectory exactly, but only for an infinitesimal time step. Due to the above-mentioned error that occurs due to the time dependence of the equations in the quadratic program the open loop inverse dynamics are not able to follow the trajectory in a meaningful way. If the target state is set equal to the initial state the algorithm is able to hold this position. This makes sense since if there is no movement, the configuration stays the same and the terms dependent on the joint angles and velocities as well. If the target is moved further away the algorithm tries to follow the desired trajectory but the error is in the same order of magnitude as the whole movement. Thus, the output joint angles are not usable. When the feedback loop is introduced the error drops significantly. It gets at worst as big as one order of magnitude smaller than the movement. Since most kicks are in the range of some centimeters the error never gets worse than some millimeters. This can be seen by comparing the commanded trajectory and the actual trajectory that results from mapping the joint angles back to the cartesian foot frame (see figure 4.1, 4.2 and 4.3).
Letting the robots perform the computed kick in the simulation or on the real robots shows that the kicks also show the same performance in reality as in the model.

4.3 Balancing

The balancing equation in the inverse dynamics could not yet be fully implemented due to a lack of time. Therefore, the balancing relies on the static stability of the robot and the compensation of inertial and gravitational moments and forces on the standing foot. The reactionary moment of

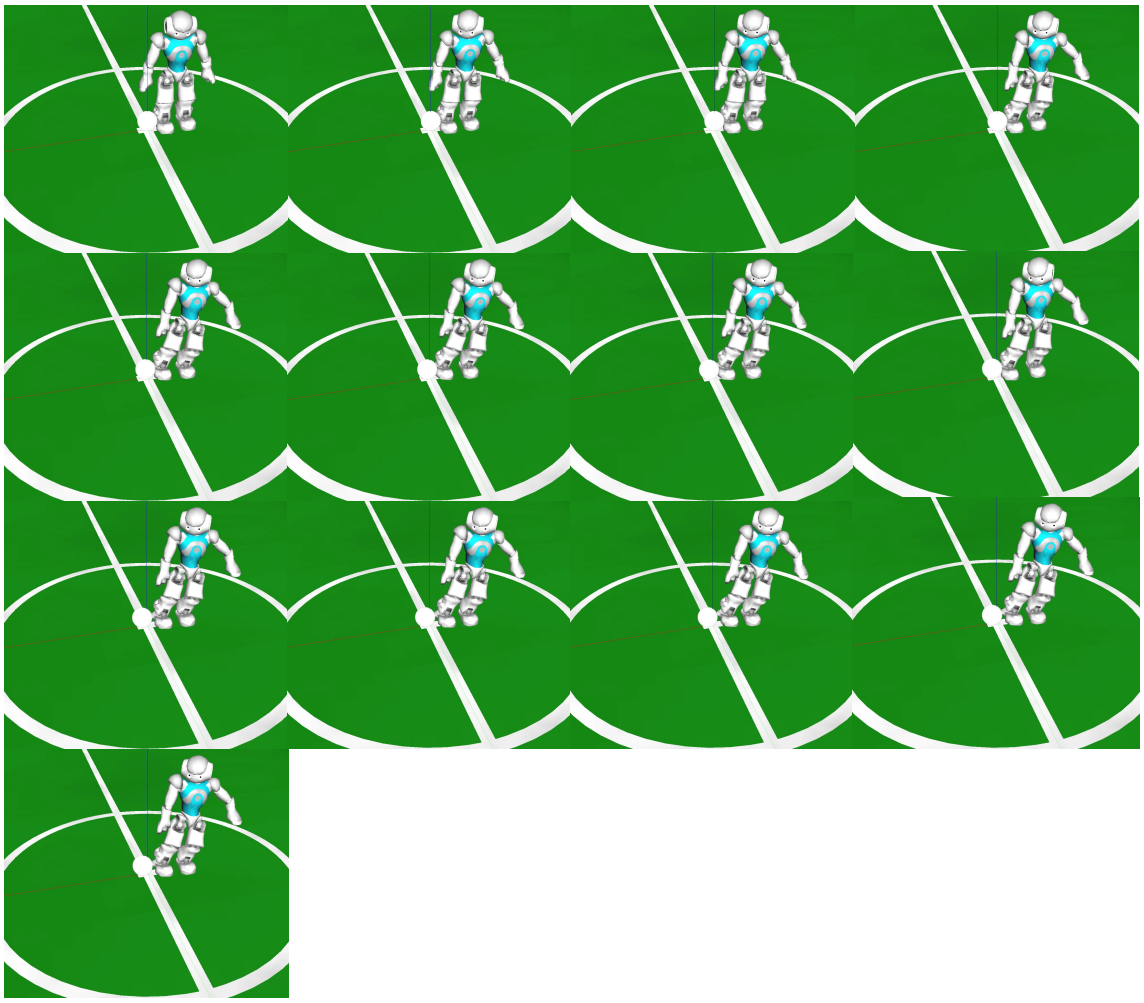

Figure 4.5: A kick simulated in SimRobot [16]. The target position is 0 cm in x-direction, 3 cm in y-direction and 0 cm in z-direction from the initial position. The target velocity is at 0.2 $^{m}/_{s}$ in y-direction.

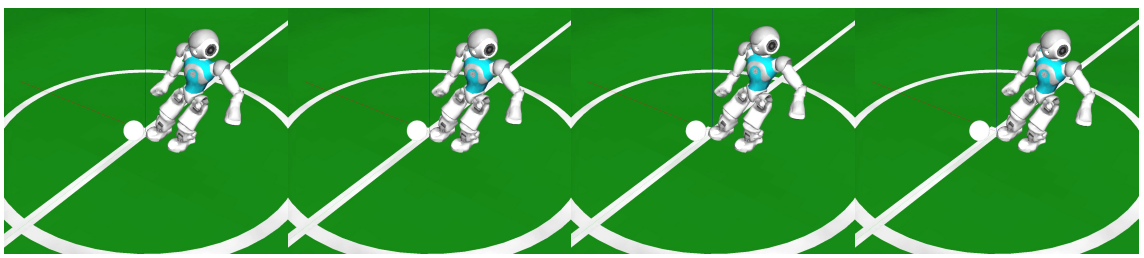

Figure 4.6: A kick simulated in SimRobot [16]. The target position is 2 cm in x-direction, 2 cm in y-direction and 0 cm in z-direction from the initial position. The target velocity is at 0.1 $^{m}/_{s}$ both in x-direction and y-direction.

the area of the foot and reactionary forces of the friction between the foot and the floor have to come up for this compensation. This is given for very small movements with target position of up to 3 centimeters away from the initial position and target velocities of up to 0.2 meters per second. For a real soccer match much bigger and faster movements are needed in order to perform useful kicks. This means that the implementation of the balancing constraint is absolutely crucial.

Chapter 5

Conclusion

The parts implemented parts of the proposed approach have shown to fulfill the requirements satisfyingly. The algorithm is able to achieve various different positions and impact velocities. This leads to a far bigger flexibility of the robot. Thence the time-consuming task of alignment will be reduced or even become needless. This can lead to a big time saving which is very crucial in a RoboCup match.
The adaptability of the impact velocity enables far more precise shots then the hard-coded kick that is implemented at the moment. This can also be quite useful for ball passing and a more advanced gameplay.
The error occurring in the following of the trajectory is so small that the necessary functionality is given. Due to the naturally occurring errors due to the inhomogeneous grass floor, the inaccuracies of the ball detection and other disturbances the remaining error from the proposed algorithm is negligible.
The needed computation time is at the current state too big for the program to be used online on the robot. The efficiency needs to be improved in order to actually bring an improvement for the robots.

Chapter 6

Outlook

The next steps are first of all bringing the balancing into a working state. If this is achieved the computation needs to be reduced at least to a few seconds for the calculation. Lower would be much better.
Given that both of these tasks are done, the code would need to be embedded into the rest of the framework. One major part of this would be to implement the interfaces. This includes defining how the robot decides upon where the ball is and where it should go and how it tells this to the algorithm proposed here. But also, how the algorithm transmits the joint angles to the robot would need to be resolved. Furthermore, the behavior of the robot would need to be adapted. The approach of the ball would stop as soon the ball is in reach. The alignment could be omitted and the robot could directly shoot.

Bibliography

[1] Ivo de Concini and Konrad Schieban. Parametrized open-loop kicking for Aldebaran's NAO H25, June 2012.

[2] Dany Manickathu. RoboCup: Generating Kicking Motions for the NAO, April 2015.

[3] Martin Kälin. Targeted and Coordinated Passing for a Bipedal Robot, October 2016.

[4] Marc Heim. Stable Ball Passing for a Bipedal Robot, October 2017.

[5] Shuuji Kajita, Mitsuharu Morisawa, Kensuke Harada, Kenji Kaneko, Fumio Kanehiro, Kiyoshi Fujiwara, and Hirohisa Hirukawa. Biped walking pattern generator allowing auxiliary zmp control. pages 2993 – 2999, 11 2006.

[6] Miomir Vukobratović and Branislav Borovac. Zero-moment point - thirty five years of its life. *International Journal of Humanoid Robotics*, 1(1):157–173, 2005.

[7] Pedro Pena. An omni-directional kick engine for nao humanoid robot, 04 2019.

[8] Kritsada Wichapong, Nantiwat Pholdee, Sujin Bureerat, and Thana Radpukdee. Trajectory planning of a 6d robot based on meta heuristic algorithms. *MATEC Web of Conferences*, 220:06004, 01 2018.

[9] Markus Hehn and Raffaello D'Andrea. Real-time trajectory generation for interception maneuvers with quadrocopters. pages 4979–4984, 10 2012.

[10] Pedro Pena, Joseph Masterjohn, and Ubbo Visser. *An Omni-Directional Kick Engine for Humanoid Robots with Parameter Optimization*, pages 385–397. 01 2018.

[11] Yuan Xu and Heinrich Mellmann. Adaptive motion control: Dynamic kick for a humanoid robot. volume 6359, pages 392–399, 09 2010.

[12] Arne Böckmann and Tim Laue. Kick motions for the nao robot using dynamic movement primitives. pages 33–44, 11 2017.

[13] Inc CVX Research. Cvx. http://cvxr.com/cvx/. Accessed July 5, 2019.

[14] SoftBank Robotics. Nao documentation. http://doc.aldebaran.com/. Accessed July 5, 2019.

[15] Marco Hutter. Robot dynamics, master course. https://rsl.ethz.ch/education-students/lectures/robotdynamics.html. Accessed July 5, 2019.

[16] Dr. Thomas Röfer. Simrobot - robotics simulator. http://www.informatik.uni-bremen.de/simrobot/. Accessed July 5, 2019.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

A Parametrized, Dynamically Balanced Kick for NAO Robots

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Eckert	Dimitri

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Muttenz, 05.07.2019	D. Eckert

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.