TUHH
Technische Universität Hamburg-Harburg

ICS
Institute of Control Systems

Bachelor thesis

Development and Implementation of a Dynamic Kick for the NAO Robotic System

Author:
Felix Wege

Supervisor: M.Sc. Patrick Göttsch

Examiner: 1. Prof. Dr. Herbert Werner
2. M.Sc. Patrick Göttsch

March 1, 2017

ICS
Institute of Control Systems

Bachelor thesis
for Mr. Felix Wege

B — 1 — 2015
Student ID: 20712345

Study Course: ETBC

Title:
Development and Implementation of a Dynamic Kick for the NAO Robotic System

Project Description:
Currently the NAO robot of the TUHH's RoboCup team is restricted to performing a shot that is a frame based motion, designed by hand and executed by loading a series of joint angles from a file. This is undesirable because there has to be a separate file for each different kick and the motion cannot be adapted to changes, for example in target direction and velocity. The aim of this bachelor thesis is to incorporate a system that is able to perform a kick generated from a movement primitive (MP), comparing two framework approaches, similar to [1] and [2]. Generalised MPs allow the NAO to adapt to changes of either the position of the ball or other robots on the pitch while executing a kick. A challenge lies in ensuring the stability [3] and [4] of the NAO when standing on one foot during the kick.

Tasks:

1. Literature survey on utilisation of MP in kick motions
2. Design and simulation of control circuit for stabilization
3. Design and simulation of the 2 framework approaches for kick motions
4. Implementation on a NAO
5. Evaluation with current implementation

References:

[1] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and Generalization of Motor Skills by Learning from Demonstration," in *IEEE International Conference on Robotics and Automation*. IEEE, 2009.

[2] S. M. Khansari-Zadeh and A. Billard, "Learning Control Lyapunov Function to Ensure Stability of Dynamical System-based Robot Reaching Motions," *Robotics and Autonomous Systems*, vol. 62, no. 6, pp. 752–765, 2014.

[3] F. O. Poppinga, "Design und Implementierung eines Regelungsverfahrens zur Stabilisierung eines humanoiden Roboters bei verschiedenen Bewegungsabläufen aus Basis des NAO–Robotiksystems," TUHH, 2015.

[4] A. Böckmann and T. Laue, "Kick Motions for the NAO Robot using Dynamic Movement Primitives," in *20th RoboCup International Symposium*, 2016.

Supervisor: M.Sc. Patrick Göttsch

Examiner: 1. Prof. Dr. Herbert Werner
2. M.Sc. Patrick Göttsch

Start Date: 17.11.2016
Due Date: 18.01.2017

17.11.2016, Prof. Dr. H. Werner

Hereby I declare that I produced the present work myself only with the help of the indicated aids and sources.

Hamburg, March 1, 2017 Felix Wege

Abstract

In this thesis, a dynamic motion for kicking a ball is designed and implemented. The kick is implemented and tested on the NAO robotic system, a humanoid robot that is used in the RoboCup Standard Platform League. It is demonstrated that with the implemented kick, the NAO is able to kick the ball to cover a desired distance and from different ball positions. This is achieved by using Dynamic Movement Primitives to generate the motion, which allow to generalize a learned motion by changing scaling parameters. By applying gyroscope feedback to the ankle joints, the NAO can maintain balance during the kick.

Contents

List of Figures

Index of Abbreviations

CoM	Center of Mass
CoP	Center of Pressure
DoF	Degree of Freedom
DMP	Dynamic Movement Primitive
FSR	Force Sensitive Resistor
HULKs	Hamburg Ultra Legendary Kickers
IMU	Inertial Measurement Unit
LIPM	Linear Inverted Pendulum Model
MoMP	Mixture of Motor Primitive
MP	Motor Primitive
MRE	Magnetic Rotary Encoder
NAO	NAO Robotic System
ODE	Ordinary Differential Equation
PID	Proportional–Integral–Derivative
RoboCup	Robot Soccer World Cup
SPL	Standard Platform League
ZMP	Zero–Moment Point

1 Introduction

Humanoid robots—robots with a human–like shape—have long been a subject of science fiction and imagination. Industrial robots are common nowadays, but the notion of humanoid robots as our companion in everyday life remains not much more than wishful thinking. However, continuous research in the field of humanoid robotics steadily advances the capability of these robots, either by improving their hardware or software. This thesis focuses on the latter.

Besides developing humanoid robots that assist humans directly through completing given tasks, there is much to be learned by conducting research on them. For example, humanoid robots have provided new insights to biomechanics and neuroscience, [1]. Designing and working with humanoid robots has helped to understand human motions, e.g. balanced gait. This allowed for a further application: the development of better prostheses, [2].

As a measure to motivate research on humanoid robotics the Robot Soccer World Cup (RoboCup) was initiated and the first RoboCup games were held in 1997. A team of students of the TUHH competes in the Standard Platform League (SPL) of the RoboCup, where teams from different universities have to use robots of the same model, a toddler–sized humanoid robot by the name NAO. Thus, the SPL focuses on implementing Software that achieves cognition of pitch, ball, goal and other competitors; Software that controls the behaviour of individual NAOs and the whole team; and Software that realises motions such as walking, standing up and kicking. This thesis endeavours to implement a successful kicking motion.

Since the environment is often unpredictable, an important trait of any humanoid robot is flexibility. Robots that can handle changing environments by adapting their motions are desirable. Applied to kicking, there are different ways in which the environment can change, e. g. a varying ball position. Thus, a robot that can adapt its kick dynamically is superior to one that can not do so. Humanoid robots in fields of application outside of RoboCup also require this kind of flexibility and adaptability. Previously, key frames (see section 2.1) have been used by TUHH's RoboCup team Hamburg Ultra Legendary Kickers (HULKs) and they have proven to be successful at scoring goals, however, because of the disadvantage in flexibility and generalizability that accompany this method, a dynamic kick is to be implemented.

During a kick the NAO merely stands on one foot. Hence, not tipping over and maintaining balance is a formidable task, especially since the RoboCup SPL uses artificial turf in which the NAO Robotic System (NAO) sinks in a bit under its own weight, [3]. Also, collisions with other robots are a frequent disturbance which further stress the balance. To successfully execute a kick, a means to balance on one foot is to be implemented.

1.1 Chapter Outline

The implementation of a dynamic kick consists of two separate objectives: the actual kick motion and maintaining balance while kicking. This is reflected in the structure of this thesis.

Section 2 presents and explains different methods to design motions used by teams competing in the SPL or in humanoid robotics in general. The advantages and disadvantages of each method are discussed with respect to desired traits of a dynamic kick. Mainly, the concept of Dynamic Movement Primitives (DMPs)—a nonlinear system that generates a trajectory—is elaborated.

In section 3 the concept of static and dynamic balance is explained. Different ways to maintain a balanced pose are presented, among them a controller that uses the NAO's gyroscope to adjust ankle roll and pitch of the support foot.

Subsequently, section 4 presents the NAO that is used to implement and test the dynamic kick. It showcases essential features of the existing HULKs' software framework that are required for the implementation, e. g. forwards and inverse kinematics.

Thereafter, section 5 explains how a DMP is learned from a demonstrated motion. The learned kick motion is analyzed in MATLAB and then transferred to the NAO. In addition, the implementation of measures to maintain balance is described.

In section 6 the implemented kick is evaluated. Also, the balance during kicking is assessed. This is done with respect to the requirements defined in section 1.2.

At last, section 7 provides a review on the performed tasks. A final conclusion is drawn. Also, improvements to the implemented kick and future topics of research are pointed towards.

Before the next section begins, requirements that define a dynamic kick are defined.

1.2 Requirements for a Dynamic Kick

For a kick to be considered dynamic, it has to meet certain requirements. They are defined as follows:

1. **Adapt to Changes in Ball Position**
   The kick should be able to kick any ball that is within the reachable space of his feet without having to position precisely.

2. **Kick in Different Directions**
   A dynamic kick should also be able to kick the ball in any direction desired without reposition to save time.

3. **Adjust Distance Covered by Ball**
   It is vital for a successful robot football game that the robots are able to pass. For this, the distance covered by the ball needs to be controlled.

4. **Maintain Balance while Kicking**
   If the robot tips over, the kick is likely to miss and therefore, this should be prevented.

These requirements are used throughout this thesis to assess different approaches for kicking and balancing. They also serve as a basis for the evaluation of the implemented kick.

2 Kick Motions

In general, humanoid robots have a high number of Degrees of Freedom (DoFs). That is why designing and generating motion patterns for them is a complex task. This chapter introduces three means of modelling and designing motions for robots. It focuses on their applicability to kicking motions and discusses their advantages and disadvantages.

Greatly important for kicking motions is the trajectory of the foot that strikes the ball. This foot is the end effector of the kicking leg. It can be described by a set of joint angles as well as in task space by a position and orientation, [4]. Both the description with joint angles of position and orientation can be used to design motions. To design a motion by specifying joint angles can be less intuitive than using Cartesian coordinates, but they can be sent directly to the motors.

Approaches to design motions of robots can be distinguished into online and offline generated motions. Generating a motion *offline* means it is designed externally and prior to its execution. Therefore, it is not limited by the computational power of the robot and intuitive in its design process. However, it lacks adaptability because the robot will not be able to adapt to changes of the environment without aborting and restarting the execution to take new sensory data into account.

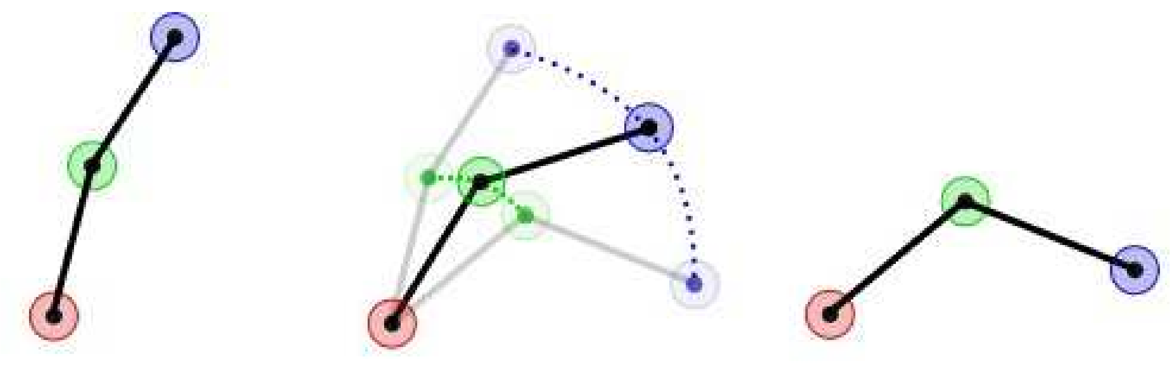
Conversely, *online* generation happens during the execution and can use incoming information to adjust the motion. This is especially advantageous in the RoboCup SPL because a successful kick has to consider the constantly changing game state on the football pitch as well as perturbations. Obviously, calculating the motion online is limited by the robot's computational power.

A compromise between online and offline generation can be made by using an offline generated motion that allows to be changed online, [5]. In the following sections, three different approaches to model motions of humanoid robots with varying degrees of online generation are explained and their suitability for a dynamic kick is discussed. The three approaches are interpolating key frames (section 2.1), trajectories modelled by splines (section 2.2) and Dynamic Movement Primitives (setion 2.3).

2.1 Interpolated Key–Frames

Key–frame based motion planning uses sets of joint angles for discrete points in time and interpolates between them. This method has been and is being used by a number of RoboCup SPL teams and humanoid robots in general, [6]. It is also commonly used in animation and filmmaking. Figure 2.1 depicts an exemplary sequence of two key frames 2.1a and 2.1c. Each frame describes the position of the limbs in terms of joint angles. To create a smooth motion, frames between two different key frames are interpolated for each step in time. Here, just one interpolated frame (2.1b) is shown.

Each individual key frame has to be set up manually, which is time–consuming. This deficiency can be overcome by deriving the key frames from kinaesthetic teaching, meaning a human teaches a robot a motion by physically guiding it. This still requires a separate



(a) First key frame (b) Interpolated in–between frame (c) Second key frame

Figure 2.1: A sequence of two key frames with one interpolated in–between frame (`http://www.real3dtutorials.com/images/img00073.jp`, 20.12.2016)

set of joint angles for each different motion, which is undesirable for it is impossible to have sets of key frames for each conceivable combination of ball and robot position as well as target direction. In practice, only a finite number of different motions can be implemented with key frames and for that reason the robot has to occupy a certain position relative to the ball and the target before it executes a kick.

Since the motion is fixed, the robot is unable to cope with disturbances and might lose its balance; stability is not guaranteed. Instead of specifying the motion by means of joint angles, one can define the position of the robot's limbs with Cartesian coordinates, as proposed by Czarnetzki et al. [7]. Thence, the calculation of required joint angles is underconstrained and additional constraints can be imposed. These constraint can be used to include a feedback controller that establishes stability. The respective joint angles can be calculated by inverse kinematics (see section 4.4). Furthermore, learning algorithms can be applied to optimize the key frames, [5].

2.2 Modelling a Trajectory with Curves

Instead of describing the motion via pre–recorded key frames, the desired trajectory of the end effector can be modelled. This could be achieved with a parametric curve or spline that is a polynomial derived from a set of control points or knots. Two suitable kinds of splines are presented here, but plenty of other ways to model a trajectory are conceivable. Both of them consist of a linear combination of control points. Again, since this motion is specified by Cartesian coordinates, inverse kinematics (see section 4.4) is necessary.

2.2.1 Bézier Curves

The Bézier curve for $n+1$ control points $P_i,\ i \in [0, n]$, is defined as

$$C_{B\acute{e}zier}(t) = \sum_{i=0}^{n} B_{i,n}(t) P_i. \tag{2.1}$$

Therein, $B_{i,n}(t)$ is the i–th Bernstein polynomial, which by definition is

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}. \tag{2.2}$$

A curve with $n+1$ knots is of degree n. Complex motions can either be described by a high order polynomial or a sequence of lower order polynomials. For the purpose of computational speed polynomials of high order are impractical. Choosing a degree of $n = 3$ is a reasonable choice since it still allows for smooth transitions, i. e. C^1 continuity, between two consecutive curves.

In general, all knots except the first and last one do not lie on the curve, but they affect the shape of it. The tangent at the first knot is $P_1 - P_0$ and the tangent at the last knot is $P_n - P_{n-1}$. Let P_0, P_1, P_2, P_3 and Q_0, Q_1, Q_2, Q_3 denote two sets of control points. To create a composite Bézier curve the last knot of the first curve P_3 has to coincide with the first knot of the second curve Q_0. This enforces continuity, but not continuous differentiability, i.e. smoothness. The slope at the connection point $P_3 = Q_0$ is equal if $P_3 - P_2 = Q_1 - Q_0$ holds; this is true when P_2, $P_3 = Q_0$ and Q_1 are collinear and equidistant. This condition ensures smoothness, [8].

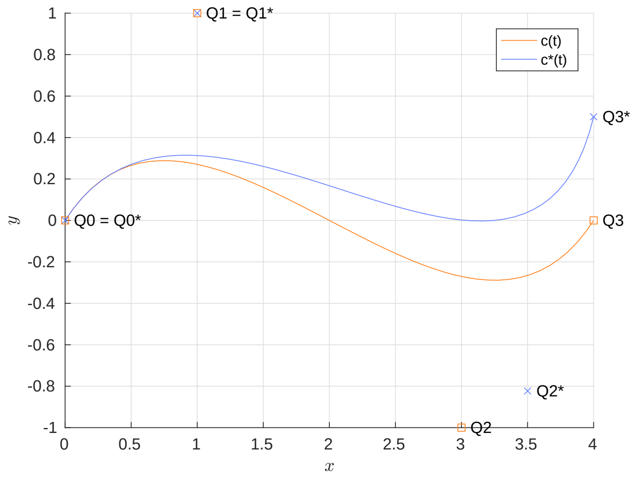


Figure 2.2: The Bézier curve $c(t)$ depicts the initial progression and the Bézier curve $c^*(t)$ depicts the adaptation to the new target position and direction.

When calculating the coefficients of a curve, all control points of this curve have to be known. However, if multiple curves are sequenced, only the curve that is currently used to perform a motion has to be known and the control points of future ones can be set online. Thus, the trajectory can be tuned by changing the control points and dynamic motions are possible. To satisfy the condition for smoothness the control points with index 0 and 1 have to meet certain conditions; they can not be set arbitrarily. The remaining control points, however, can be set unconditionally. The last one determines the final position while the penultimate determines the final direction of the trajectory. Figure 2.2 illustrates this circumstance. To adapt the curve $c(t)$ to a new final position, the control point Q_3^* is set accordingly. Then, Q_2^* can be set to determine the direction at the end of the trajectory. Connecting the resulting curve $c^*(t)$ to the following curve with continuous differentiability is possible for $Q_3 - Q_2 = Q_3^* - Q_2^*$. With this, a kick motion can be adapted dynamically to strike a ball at any reachable position in a desired direction.

2.2.2 B–Splines

With Bézier curves a motion can be adapted dynamically to new positions and directions. However, other requirements, e. g. the strength of the kick, need to be met (see section 1.2). To incorporate these, Wenk and Röfer presented a method that uses B–splines, [6]. B–splines are a more general form of Bézier curves and they allow for more control over the trajectory by the user. A B–spline of degree p is defined as follows, [9]:

$$C_B(t) = \sum_{i=0}^{n} N_{i,p}(t) P_i. \tag{2.3}$$

Here, the basis functions are recursively defined by

$$N_{i,j}(t) = \frac{t - t_i}{t_{i+j} - t_i} N_{i,j-1}(t) + \frac{t_{i+j+1} - t}{t_{i+j+1} - t_{i+1}} N_{i+1,j-1}(t). \tag{2.4}$$

and

$$N_{i,0}(t) = \begin{cases} 1 \text{ if } t_i \ \leq \ t < t_{i+1} \text{ and } t_i < t_{i+1} \\ 0 \text{ else} \end{cases} \tag{2.5}$$

The degree of the curve is $p = m - n - 1$. In addition to the control points $P_i, i \in [0, n]$, there now are internal knots $t_i, i \in [0, m]$.

Wenk and Röfer [6] defined 6 control points to determine the shape of their kick trajectory. With a degree of 3 there are $m = p + n + 1 = 10$ internal knots. They then formulated conditions for durations of segments between control points. Also, since the foot should not move at beginning and end of the kick, derivatives at this point should be zero. With enough conditions, all internal knots can be determined and the curve can be calculated. The resulting curve is continuously differentiable twice, which means there is no jerk. The control points are set at the beginning of the motions; trajectory modelling is not required beforehand.

Just as for the key frames, joint angles for the end effector to move along the desired trajectory can be calculated via inverse kinematics. For both the Bézier curves and the B–splines, all limbs that are not involved in the kicking motion can be used for balancing.

Using either the Bézier curves or B–splines for motion planning allows for adapting to changes in the environment but to do so, the motion has to be replanned. Furthermore, a controller is necessary to execute the motion with the capability to cope with disturbances and minimize the tracking error. While it is possible to use this approach for a dynamic kicking motion, there exist more suitable approaches.

2.3 Dynamic Movement Primitives

A motion pattern such a kicking a ball is a Motor Primitive (MP) in the sense that a robot accomplishes a given task with a sequence of motor commands, [4]. Such motions can be generated by means of a dynamical system. The planning and execution are embedded in the dynamical system; this means it generates a trajectory and follows it in the face of disturbances. Such a system is useful to model goal–orientated behaviour; one that has to fulfil a certain task, e.g. kick a ball. The time evolution for a given initial state is determined by the dynamical system. An estimation of a dynamical system can be created from single or multiple kinaesthetic demonstrations.

DMPs offer a way to achieve this. They are formalized in a second–order Ordinary Differential Equation (ODE) which has a weak nonlinearity that can be learned from kinaesthetic demonstration. DMPs allow online adaptation of the motion while ensuring stability. A major advantage is the generalizability in space and time by changing parameters. There are two variants of DMPs, one for discrete and one for rhythmic motions. A discrete motion has a point attractor whereas a rhythmic motion has a limit cycle attractor. Here, only the one that models discrete motions is explained since it is applicable to kicking motions. In theory rhythmic DMPs could be used for gait motions.

2.3.1 Discrete Dynamic Movement Primitives

Discrete DMPs consist of the *transformation system* 2.3.1 and the *canonical system* 2.3.1. Integrating the former generates a position whereas the latter acts as a clock. Learning a motion is achieved through a learnable forcing term. In the following sections, x, y and z do not correspond to spatial dimensions but instead represent the system state.

Transformation System

The system is described by the following equation [1]:

$$\tau\ddot{y} = K\ (g-y) - D\tau\dot{y} + (g-y_0)f. \tag{2.6}$$

Commonly, this is rewritten into a system of two first–order ODEs:

$$\begin{aligned}\tau\dot{z} &= K(g-y) - Dz + (g-y_0)f\\ \tau\dot{y} &= z.\end{aligned} \tag{2.7}$$

[1]Alternatively, $\tau\ddot{y} = \alpha_z(\beta_z(g-y)-\dot{y}) + x(g-y_0)f$ is an equally valid representation of the transformation system and all following equations could be expressed with this notation, [10]. However, the other representation is chosen because it can intuitively be interpreted as a mass–spring–damper system.

Here, y, z and $\dot{z}$ denote the position, velocity and acceleration of the system respectively and τ is a temporal scaling factor. Subsequently, the duration of the motion is normalized and therefore also labelled by τ. The nonlinearity (or forcing term) f allows the system to perform arbitrarily complex smooth movements while it converges towards the target state g, also known as the attractor. This is due to the factor $(g - y_0)$ which enables spatial scaling. Without the nonlinearity, the system is equal to the common mass–spring–damper system.

Customarily, g is called goal position but the term *goal* is ambiguous when referring to football so the final position shall be named *target.* K and D are positive constants that can be used to tune the damping ration. A critically damped system is desired since it returns to the attractor as fast possible and without oscillating. This is achieved for $D = 2\sqrt{K}$, with D being a damping constant and K acting as a spring constant, [11].

Canonical System

To enable temporal scaling, an *autonomous system*—one that does not explicitly depend on the independent variable—is necessary. Therefore, the canonical system with phase variable x for discrete motions is introduced [2].

$$\tau \dot{x} = -\alpha_x x \tag{2.8}$$

This ODE's closed–form solution is given as $x(t) = exp(-\alpha_x \frac{t}{\tau})$. Evidently, α_x is a decay factor that defines how fast the exponential decay is. The *phase variable* x converges monotonically to zero for any given initial value x_o; often, the x_0 is set to 1. Notably, the canonical system for rhythmic motions is periodic.

Nonlinearity

The nonlinearity (or forcing term) f that defines the motions' shape is defined as

$$f(x) = \frac{\sum_{i=0}^{N} \psi_i(x) w_i}{\sum_{i=0}^{N} \psi_i(x)} x, \tag{2.9}$$

wherein $\psi_i(x)$ are basis functions and w_i are associated weights. They are learned from a demonstrated motion and they determine the shape of the nonlinearity f, which in turn determines the shape of the motion that is reproduced by the transformation system. Utilizing the phase variable x instead of explicitly depending on the time t makes the system autonomous. Furthermore, multiplying by x makes the forcing term vanish at the end of the motion because it converges to zero.

The centers c_i are chosen to be equidistantly spaced. Because we use x instead of t and x is exponentially decaying, $c_i = \exp\left(-\alpha \frac{t_i}{\tau}\right)$ holds, where t_i are N equidistantly spaced points in time between 0 and τ. The N basis functions are almost unanimously implemented as

[2]The phase variable x as well as the state of the transformation system y and z are not to be associated with physical dimensions of space.

Gaussian functions in literature concerning DMPs, with σ_i^2 and c_i determining the width and center of the i–th basis function:

$$\psi_i(x) = \exp\left(-\frac{1}{2\sigma_i^2}(x - c_i)^2\right). \tag{2.10}$$

For a given demonstrated trajectory y_{demo} and its derivatives $\dot{y}_{demo}$ and $\ddot{y}_{demo}$ the appropriate nonlinearity f_{learn} can be computed by solving equation 2.6 for f; f_{learn} then approximates the necessary forcing term f so that y resembles the demonstrated trajectory.

$$f_{learn} = \tau\ddot{y}_{demo} + D\dot{y}_{demo} + K(y_{demo} - g) \tag{2.11}$$

With f_{learn} and $\psi_i(x)$ known, the according weightings w_i can be calculated by any regression algorithm.

Generalizability and other properties

DMPs can straightforwardly be generalized in time and space by changing few parameters; the system is spatially and temporally invariant. The temporal scalability is possible due to τ. For $0 < \tau < 1$, the is executed more quickly that is was learned and for $\tau > 1$ it is slowed down. Spatial scaling is done by changing the initial position y_0 or the final position g. The shape of the trajectory stays the same (at least for the same weightings) and since the forcing term vanishes for $t \to 0$, the final position is always reached. The motion can be scaled at the start or during runtime.

Besides the generalizability that is valuable for a dynamic kick DMPs possess other properties that are favourable. For example, the dynamics of the system automatically cope with external perturbations due to the intrinsic attractor behaviour, [11]. Therefore, the system is robust against perturbations. DMPs can easily be expanded to more than one dimension by simply describing each one with its own transformation system that share a canonical system for synchronisation purposes. With this, trajectories in a three–dimensional space can be generated conveniently. Describing the motion in task space has the advantage that spatial scaling for different ball positions is straightforward. The required joint angles that move the end effector along the generated trajectory are calculated by inverse kinematics (see section 4.4).

Additionally, new motions can be learned with ease. The weightings w_i can be learned from a demonstration by anyone without special training. It is not required to explicitly plan the motion. Furthermore, multiple DMPs can be joined together by sequencing them, allowing robots to perform even more complicated tasks [12]. DMPs can also summed together and weighted to create a mixture of motions, [4]. This, together with the generalizability, makes DMPs suitable for a robot designed to assist humans in daily routine.

2.3.2 Variations and Extensions for Dynamic Movement Primitives

Since their earliest proposal by Ijspeert et al. [13], DMPs have been applied in a variety of ways because of all the reasons stated above. Ijspeert et al. themselves used them for drawing two–dimensional shapes and for tennis swings. Additionally, they have been applied to grasping a bottle and pouring water out of it as well as pick–and–place motions, [10]. For a three–dimensional goal–oriented application such as a kick, the DMPs framework is a feasible approach to generate a dynamic kick. However, the original framework can be modified to better fit the task of kicking and overcome some deficiencies.

Third–Order Transformation System

When the target g is changed during execution a discontinuity in the acceleration occurs. Huge steps in acceleration are undesirable because the motor might not be able to follow motion. This deficiency can be overcome by filtering the target change. For this, the target g is considered to be a continuous variable and the changed target is denoted by g_0. The transformation system then becomes

$$\begin{aligned} \tau\dot{z} &= K(g-y) - Dz + (g-y_0)f \\ \tau\dot{y} &= z \\ \tau\dot{g} &= \alpha_g(g_0 - g). \end{aligned} \tag{2.12}$$

Essentially, the transformation system becomes a third–order system, [10]. Any sudden change in target position is smoothed. This does not change the generalizability or stability of DMPs.

Modified Transformation System

No motion will be generated if the initial and target position coincide; $y_0 = g$. For $y_0 - g \ll 1$ small changes in g lead to huge accelerations which could result in an infeasible motion. Moreover, if the sign of the target is changed, the trajectory is mirrored, [14]. All of these disadvantages can be prevented with subsequent alteration of the transformation system[3]:

$$\begin{aligned} \tau\dot{z} &= K(g-y) - Dz - K(g-y_0)x + Kf \\ \tau\dot{y} &= z. \end{aligned} \tag{2.13}$$

This formulation generates motions for $y_0 = g$ due to the fact that f is not multiplied by $(g - y_0)$; f is still multiplied by K to make spatial scaling possible. The $K(g - y_0)$ term avoids discontinuities in acceleration at the beginning of motions because it cancels out the initial $K(g - y)$. For learning from demonstration equation 2.11 has to be adjusted to fit this modified transformation system.

Moving Target

Using the formulation described in 2.7, the final velocity—that is, the velocity at the end of the imitated motion—can only be scaled if the duration is changed. Controlling this

[3]Interestingly, this approach was motivated by neurophysiology of spinal force fields in frogs under spinal cord stimulation.

velocity is important for kicking motions since is corresponds to the strength of the kick and therefore the distance the ball will cover. The transformation system can be altered to make independent scaling of $\dot{y}$ and τ possible.

Instead of using solely a target position g_p, one can expand the transformation system by additionally incorporating target velocity $\dot{g}_p$ and target acceleration $\ddot{g}_p$, thus modelling a moving target (as opposed to the aforementioned transformation systems with a fixed target):

$$\begin{aligned} \tau\dot{z} &= K(g_p - y) + D(\tau\dot{g}_p - z) + f + \tau^2\ddot{g}_p \\ \tau\dot{y} &= z. \end{aligned} \tag{2.14}$$

There are certain conditions initial and final values have to fulfil. These are as follows:

$$\begin{aligned} g_p(0) &= y_0, \quad & \dot{g}_p(0) &= \dot{y}_0, \quad & \ddot{g}_p(0) &= \ddot{y}_0, \\ g_p(\tau) &= y_1, \quad & \dot{g}_p(\tau) &= \dot{y}_1, \quad & \ddot{g}_p(\tau) &= \ddot{y}_1, \end{aligned} \tag{2.15}$$

where y_0, $\dot{y}_0$ and $\ddot{y}_0$ denote the initial position, velocity and acceleration while y_1, $\dot{y}_1$ and $\ddot{y}_1$ denote the final position, velocity and acceleration of the demonstrated motion respectively. The final acceleration should be zero, final position and velocity can be controlled and the initial values are determined when the kick starts. Since there are six conditions, the target can be described by a fifth–order polynomial and its first and second derivative, [15]:

$$\begin{aligned} g_p(t) &= a_5t^5 + a_4t^4 + a_3t^3 + a_2t^2 + a_1t + a_0 \\ \dot{g}_p(t) &= 5a_5t^4 + 4a_4t^3 + 3a_3t^2 + 2a_2t \\ \ddot{g}_p(t) &= 20a_5t^3 + 12a_4t^2 + 6a_3t + a_2 \end{aligned} \tag{2.16}$$

The coefficients $b_0, ..., b_5$ can be calculated by solving the system of equations 2.15 with the polynomials of 2.16. A MATLAB script that computes the symbolic solution can be found in appendix A. The coefficients are

$$\begin{aligned} b_0 &= y_0, \\ b_1 &= \dot{y}_0, \\ b_2 &= \frac{\ddot{y}_0}{2}, \\ b_3 &= -\frac{20y_0 - 20y_1 + 12\tau\dot{y}_1 + 3\tau^2\ddot{y}_0 - \tau^2\ddot{y}_1}{2\tau^3}, \\ b_4 &= \frac{30y_0 - 30y_1 + 16\tau\dot{y}_0 + 8\tau\dot{y}_1 + 3\tau^2\ddot{y}_0 - \tau^2\ddot{y}_1}{2\tau^4}, \\ b_5 &= -\frac{12y_0 - 12y_1 + 6\tau\dot{y}_0 + 6\tau\dot{y}_1 + \tau^2\ddot{y}_0 - \tau^2\ddot{y}_1}{2\tau^5}. \end{aligned} \tag{2.17}$$

Since the coefficients $b_0, ..., b_5$ depend on the temporal scaling factor τ they have to be calculated anew if τ is changed. Equation 2.11 has to be adjusted appropriately for learning a motion with a polynomial target function. The parameter $\dot{g}$ controls the final velocity. The learned forcing term has to calculated for the altered transformation system of 2.14. Because f_{learn} now also depends on the target velocity $\dot{g}$ the trajectory's shape changes if $\dot{g}$ is changed. The forcing term amplitude A is introduced to correct this:

$$A = \frac{\dot{g} - \dot{y}_0}{\dot{y}_1 - \dot{y}_0}. \tag{2.18}$$

With this, equation 2.14 can be modified to

$$\begin{aligned} \tau \dot{z} &= K(g_p - y) + D(\tau \dot{g}_p - z) + Af + \tau^2 \ddot{g}_p \\ \tau \dot{y} &= z \end{aligned} \tag{2.19}$$

and thus, f_{learn} is

$$f_{learn} = \frac{1}{A} \left[\tau^2 (\ddot{y} - \ddot{g}_d) + D\tau(\dot{y} - \dot{g}_d) + K(y - g_d) \right]. \tag{2.20}$$

The amplitude A is calculated from the desired and original final velocity. With this the duration, final position and velocity can be changed freely.

3 Maintain Balance

Being a humanoid robot, the NAO has a relatively high Center of Mass (CoM) compared to four–legged or wheeled robots. As a result walking or even standing upright with small disturbance proves to be a difficult task. Kicking further stresses the postural stability because the interaction with the ball and the resulting forces are unknown. Also, as the NAO is performing a kick, only one foot's sole has contact to the ground; it is called the *support foot*, [16]. It is important that this foot remains stationary, not only for the purpose of stability, but also because it is the reference coordinate system for kinematics.

At first, the axis for the NAO are defined. The x–axis is positive from back to front , the y–axis is from right to left and the z–axis is from bottom to top (see figure 3.1). Rotations about x–, y– and z–axis are called roll, pitch and yaw, respectively. Alternatively, they are referred to as rotations in the coronal, saggital and transverse plane.

Whilst standing on one foot, the robot can rotate as a whole about the edge of the support foot. The rotation about the x– and y–axis act as two additional DoFs. Because these DoFs are unpowered and can not be controlled, they are considered to be passive DoFs. To maintain balance and avoid tipping over, the foot should ideally remain flat on the ground at all times. If only one edge touches the ground, the robot will either fall over or swing back so that the whole sole has ground contact, depending on the forces acting on the robot. Of course, no other limb of the robot should interfere with the ground. To keep the body upright, only the sagittal and coronal plane are considered. It is assumed that there is no rotation in the transversal plane because the friction keeps the foot from rotating.

From observation of humans recovering from pushes, different balancing strategies were found, [17]. For standing or walking it is possible to take a step to change the *support polygon* and thereby maintain balance. For kicking however, this is not an option since the robot has to remain stationary. Thus, the biped NAO can only be balanced by utilizing internal joints. Since the kick motion only determines the joint angles of the kicking leg, all other angles can be used to maintain balance.

There are numerous ways to maintain balance and some shall be discussed here. Most of them consider the CoM's position, velocity and/or acceleration and rely on sensory feedback. The notion of static and dynamic balance is explained in the following sections. Then, different controllers are proposed to maintain balance and cope with disturbances.

3.1 Static Balance

When a robot is not moving or only executing slow motions with small momentum, a sufficient criterion for balance is that the projection of the CoM to the ground is within the support polygon, where the support polygon is the convex hull of all parts that contact the ground, [1]. What this means is that CoM projection should be inside the area that is formed by enclosing all points that contact the ground. This is illustrated in figure 3.2. For kicking motions, the support polygon is equal to the sole of the support foot.

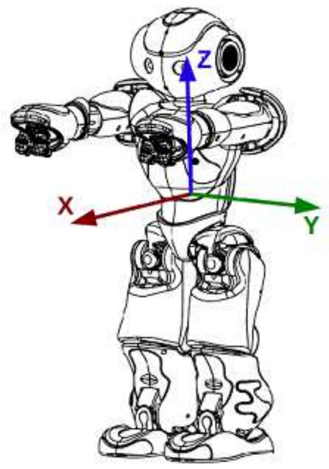


Figure 3.1: The axes as they are defined for the NAO (`http://doc.aldebaran.com/1-14/_images/torso_xyz_axis.png`, 30.12.2016)

3.2 Dynamic Balance

Because the idea of static balance only applies if a robot is standing still or in minuscule motion, it is most likely to be inapplicable to a kicking NAO because a fast kicking motion will result in dynamic forces that can not be coped with this way. Suppose the NAO is in a single–support phase—that is to say just one foot has ground contact—and is moving an arbitrary way. Then, at the contact surface, the ground reaction can be represented by three forces (F_x, F_y, F_z) and three moments (M_x, M_y, M_z) with the axis definition as in figure 3.1. These result from the distributed loads at the sole. If there is no sliding, the horizontal forces F_x and F_y compensate the vertical moment M_z due to static friction, [16]. To assess the stability, the horizontal moments M_x and M_y are to be examined. Horizontal moments can be compensated for by the point of application of the vertical force F_z and since the vertical load is unidirectional (its sign is the same over the entire surface), the point of application can only be in the support polygon. This point of application (p_x, p_y) on the surface of the foot where the moments M_x and M_y are zero is called the Zero–Moment Point (ZMP). It can be used to judge the stability of a robot. If the ZMP exists within the support polygon, the robot can be in dynamic balance. A ZMP outside the support polygon is a fictional ZMP, for the point of application of the ground reaction force can not exist outside of the contact area.

The point of application of the ground reaction force also is the Center of Pressure (CoP). If the robot is dynamically balanced, the ZMP and CoP coincide. This is illustrated in figure 3.2. The CoP always lies within the support polygon, but the necessary force application point of F_x in order to balance the robot can lie outside of the support polygon, constituting a fictional ZMP. However, the force can not actually have its point of application outside of the support polygon. In this case, the CoP ad the ZMP do not coincide and dynamic balance is not ensured.

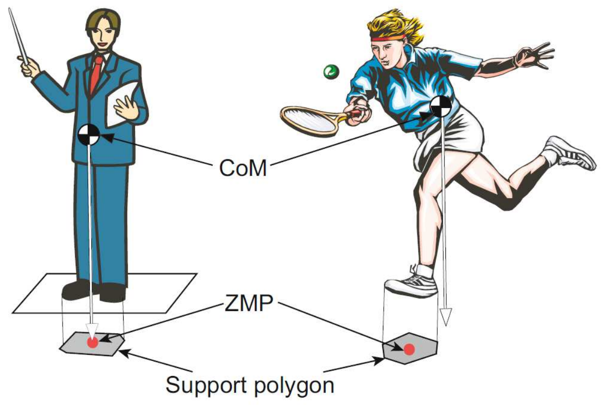


Figure 3.2: The left part depicts a sparsely moving human whose CoM projection is inside the area formed by its feet, coinciding with the ZMP. On the right, the CoM lies outside the support polygon during a rapid motion, but the ZMP lies within it.

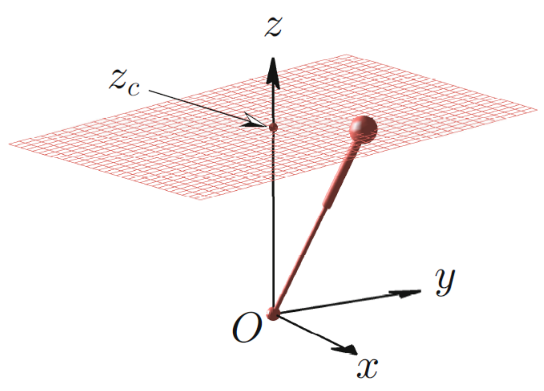


Figure 3.3: A three–dimensional inverted pendulum. The CoM is restricted to move in the xy–plane at a constant height.

3.3 Linear Inverted Pendulum Model

A Linear Inverted Pendulum Model (LIPM) can be used to model a humanoid robot standing in an upright pose. As a simplifying assumption, the robot's entire mass is assumed to be located in the CoM, which is at a constant height z_c. The legs that form the rod are assumed to be massless. The pendulum is assumed to have a rod of variable length so that the CoM can have a height above the ground when its position in the transverse plane changes. In addition, the pendulum is linked to the ground by a pivot that allows roll and pitch. The LIPM is described by the following equation:

$$\ddot{x} = \frac{g}{z_c} x. \tag{3.1}$$

The LIPM can be used to derive control policies that maintain balance.

3.3.1 Ankle Torque Control

For a two–dimensional LIPM with ankle torque τ_x the CoM dynamics can be described by a slightly altered LIPM–equation:

$$\ddot{x} = \frac{g}{z_c} x + \frac{1}{m z_c} \tau_x. \tag{3.2}$$

In equation 3.1, $z_c = const.$ is the CoM height, g is the gravitational acceleration and x and $\ddot{x}$ are the CoM position and acceleration, respectively. A PD–controller for the ankle torque proposed by Kajita et al. [1] is as follows:

$$\tau_x = -K_p x - K_d \dot{x}. \tag{3.3}$$

The feedback gains k_p and k_d can be tuned to achieve a desired frequency response. Integral feedback can be incorporated to reduce the steady state error, [18]. The dynamic behaviour is the same for y and $\ddot{y}$ instead of x and $\ddot{x}$ since the two coordinates are independent. A three–dimensional LIPM is a superimposition of two two–dimensional LIPM, one for the coronal plane and one for the sagittal plane. This allows for the dimensions to be controlled independently. This *ankle strategy* for balancing can be applied only if the torque of the motors can be set precisely. For the NAO the motors are controlled by sending desired joint angles and stiffnesses (see section 4.1). The torque can not be set directly and therefore, this approach is unsuited.

3.3.2 Gyroscope Feedback Control

As mentioned above, the NAO is assumed not to rotate in the transversal plane. To register motion in the sagittal and coronal plane the NAO's gyroscope proves to be useful. The gyroscope that is part of the Inertial Measurement Unit (IMU) measures the angular velocity ω^r_{Gyro} and ω^p_{Gyro} of the torso, where the superscript r and p stand for roll and pitch, respectively. For most poses, the CoM can be assumed to be located in the torso since its mass exceeds that of other limbs, therefore the measured angular velocity resembles the CoM's angular velocity. For the sagittal plane, ω^r_{Gyro} is proportional to the torso's velocity $\dot{x}$. Analogously, ω^p_{Gyro} is proportional to $\dot{y}$.

The angular velocity can be used for feedback control. To accomplish this, deviation from a desired velocity is fed back to a joint that corrects this error. For example, if there is a roll ω^r_{Gyro}, the ankle roll joint could tilt the foot relative to the rest of the body to compensate for this. Thus, the CoP can be shifted to that is lies in the support polygon. Alternatively, the hip roll joint could be used for such motion and self–evidently the same balancing approach can be applied to the coronal plane.

Ankle Controller

A balanced pose can be established if the body's velocity is kept at zero. The measured gyroscope data can act as the control error. A proportional controller is proposed to balance the NAO by adjusting the ankle roll and pitch, [19]:

$$\theta^r_{Foot} = \theta^r_{0,Foot} + K^r_P \, \omega^r_{Gyro}. \tag{3.4}$$

The gyroscope reading ω^r_{Gyro} is added to the ankle roll $\theta^r_{0,Foot}$ that the ankle of the support foot currently has. The actual joint angle then sent to the motor is θ^r_{Foot}. The same is done for the ankle pitch and the appropriate gyroscope reading. The constants K^r_P and K^p_P are proportional gains Figure 3.4 shows the design of the left leg of the NAO with all its joints. This design for a controller is justified by Hengst by solving the LIPM equation which leads to a hyperbolic cosine in the state space $(x, \dot{x})$, [20]. The hyperbolic cosine is then approximated by a straight line.

A P–controller only takes into account the currently measured gyroscope data. The steady–state error gets smaller the larger the gain is. But a large gain often leads to an

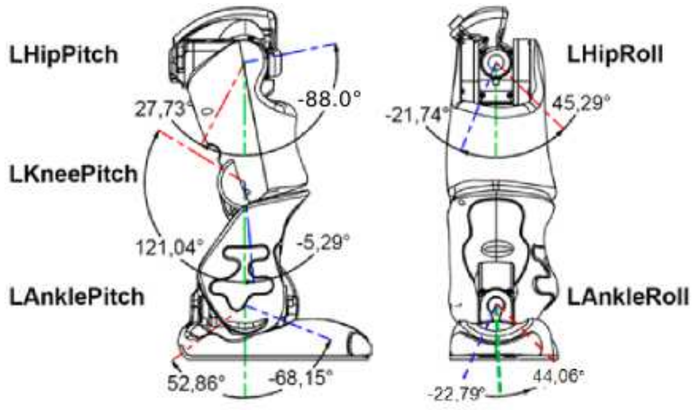


Figure 3.4: The 5 joints of the left leg of the NAO with their respective ranges and initial positions, [21].

undesired transient behaviour such as oscillating or large peak overshoot, [18]. This can be improved by adding integral feedback. Differential feedback on other hand can help to reduce the settling time. A Proportional–Integral–Derivative (PID) ankle controller with gyroscope feedback looks as follows:

$$\theta^r_{Foot} = \theta^r_{0,Foot} + K^r_P \left(\omega^r_{Gyro} + \frac{1}{T^r_I} \int_{t_0}^{t} \omega^r_{Gyro} \,\mathrm{d}\tau + T^r_D \frac{\mathrm{d}\omega^r_{Gyro}}{\mathrm{d}t} \right). \tag{3.5}$$

Hip Controller

A similar controller can be used to balance the NAO by adjusting the hip roll and pitch. This approach has been used for the NAO by Müller et al., [8]. PID controller that adjust given hip roll $\theta^r_{0,Hip}$ with gyroscope feedback

$$\theta^r_{Hip} = \theta^r_{0,Hip} + K^r_P \left(\omega^r_{Gyro} + \frac{1}{T^r_I} \int_{t_0}^{t} \omega^r_{Gyro} \,\mathrm{d}\tau + T^r_D \frac{\mathrm{d}\omega^r_{Gyro}}{\mathrm{d}t} \right). \tag{3.6}$$

Analogously, a controller for the hip pitch θ^p_{Hip} uses ω^p_{Gyro}. Both the ankle and the hip controller can be easily implemented on the NAO and either of them can contribute to maintain balance during kicking. Before the kick is further examined, some general concepts about the NAO, the existing software framework of the HULKs and related topics are covered.

4 NAO Robotic System Hardware and Software

This section introduces the NAO. Its hardware and parts of the HULKs' software framework that are relevant to the implementation are presented. Namely, kinematic matrices, coordinate transformations and forward and inverse kinematics are explained since they are frequently used in the final implementation. In addition to that, low–pass filtering of sensor readings is outlined shortly.

4.1 NAO Robotic System

The NAO is developed by French company Aldebaran–Robotics. It has a height of 57.4 cm and a weight of 5.4 kg; compared to other humanoid robots it is rather small. In 2008, it has replaced Sony's AIBO quadruped in the RoboCup SPL. Figure 4.1 shows two NAO robots competing in a match of football, one of which is about to kick the ball. Overall, the NAO was designed to provide a good performance and modularity while remaining affordable. It has 25 DoF in total and, as opposed to most other humanoid robots, it has hip joints with an inclination of 45° that make it's pelvis more similar to that of a human, [2]. Each joint has a coreless brushed DC motor that can be controlled independently except for the hip yaw–pitch, [21]. The torque of each motor can be limited by reducing the applied current to a percentage of its maximum value; this is called stiffness. The newest version has a 1.6 GHz CPU, 1 GB RAM and 2 GB flash memory. The NAO robotic system is designed to operate autonomously and can be programmed in C++ and in Python.

To perceive its environment, the NAO is equipped with several sensors, including 4 Force Sensitive Resistors (FSRs) underneath each foot, an IMU, composed of a gyroscope and an accelerometer and Magnetic Rotary Encoders (MREs) to measure the joint positions, [21]. Furthermore, the NAO is able to interact with humans and its surroundings visually and acoustically via video cameras, infra–red sensors, LEDs, loudspeakers and microphones.

4.2 Low–Pass Filter

To handle external disturbances while standing on one foot kicking, feedback control with sensor data is vital. The sensor data are obtained from methods of class `Blackboard`, which manages all sensor data of the NAO in the HULKs' code base. Sensor readings of any of the sensors mentioned in section 4.1 are noisy. Therefore, they are unfit to be used directly for closed–loop control. If noisy signals were used to control the motors it would cause vibrating. Since noise typically has a higher frequency than the measured value, a low–pass filter can be applied to smooth the data. For this an exponential moving average can be used, [8]. The filtered value $\tilde{x}_t$ for an arbitrary sensor at point in time t is recursively calculated by

$$\tilde{x}_t = \alpha\, x_t + (1 - \alpha)\, \tilde{x}_{t-1}. \tag{4.1}$$

The measured value x_t at point in time t is multiplied by a constant weighting α with $\alpha \in [0, 1]$. It is then added to the previously filtered value $\tilde{x}_{t-1}$, which is weighted by

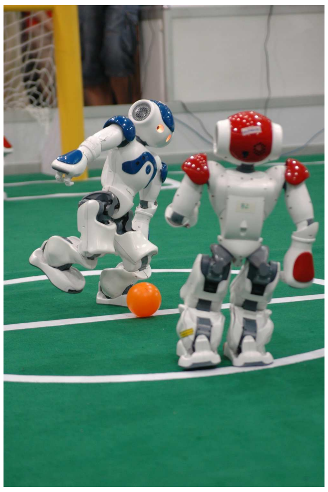


Figure 4.1: The blue NAO robot is about to kick the ball while the red one is approaching it (`http://www.ist.tugraz.at/staff/steinbauer/nao_game.jpg`, 15.12.2016)

$1 - \alpha$. Thus, the more recent a measured value is, the greater its impact on the filtered value. This filter can be easily implemented and is computationally inexpensive, but other low–pass filters could be used instead. Figure 4.2 showcases the influence of α for gyroscope roll measurements for an excerpt of a kick motion. Without any filtering, i. e. $\alpha = 1$, the signal is noisy. For decreased values α the currently measured value is trusted less. This causes the filtered signal to be smoothed; the hight–frequent noise is removed and the amplitude is reduced. However, an increasing delay comes along with that.

When looking at the gyroscope readings an interesting phenomenon is noticed: The values seem to be updated every 20 ms. But the motion cycle —requested the measured gyroscope values—has a frequency of 100 Hz. As to why this discrepancy exists is not known, but it appears as if the gyroscope only had a sample rate of 50 Hz.

4.3 Kinematic Matrices and Coordinate Transformations

The support foot serves as a reference coordinate system during a kick (see section 3). Positions and orientation defined with this coordinate system are *absolute*. Hence, the kicking foot's position should be calculated with DMPs relative to the support foot. However, the joint angles are calculated via forward kinematics (see section 4.4) from the kicking foot's position relative to the torso. To obtain them, a coordinate transformation is necessary. Another example for the necessity of coordinate transformations is the CoM that is calculated from joint angles in torso coordinates but is more useful in absolute coordinates for maintaining balance. This section briefly introduces concept of kinematic matrices and coordinate transformations.

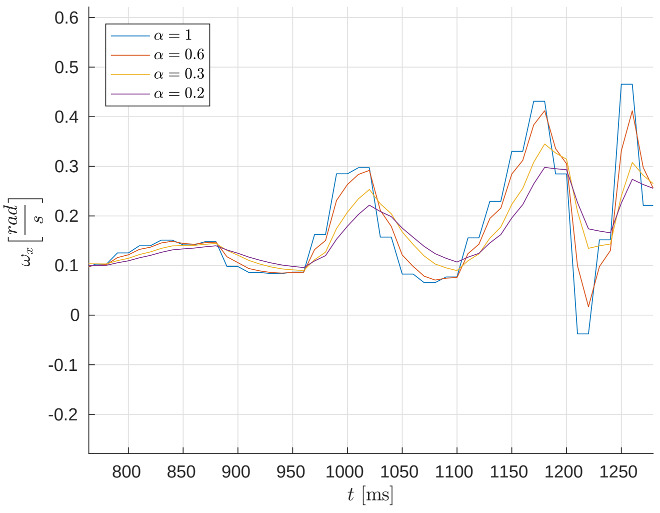


Figure 4.2: caption

Kinematic Matrices

A Kinematic matrix ${}^a\boldsymbol{T}_b$ that defines position and orientation of limb b in coordinates of limb a is written as follows:

$$ {}^a\boldsymbol{T}_b = \begin{bmatrix} {}^a\boldsymbol{R}_b & {}^a\boldsymbol{p}_b \\ 0 & 1 \end{bmatrix}. \tag{4.2}$$

In 4.2, ${}^a\boldsymbol{R}_b \in \mathbb{R}^{3\times3}$ is a rotation matrix, ${}^a\boldsymbol{p}_b \in \mathbb{R}^3$ is a position vector and ${}^a\boldsymbol{T}_b \in \mathbb{R}^{4\times4}$ is a kinematic matrix. The 1 and 0 are added to make the dimensions compatible. Kinematic matrices are represented as objects of class `KinematicMatrix` in the HULKs' framework. The nomenclature for them is $b2a$, meaning position and orientation of b relative to a is described. Not only are kinematic matrices useful for combining position and orientation, they also allow for straightforward coordinate transformations.

Coordinate Transformations

A coordinate transformation can easily be done by multiplying two kinematic matrices. Let ${}^c\boldsymbol{T}_a$ denote position and orientation of a in reference coordinate system c and ${}^a\boldsymbol{T}_b$ denote position and orientation of b in coordinates of a. To obtain ${}^c\boldsymbol{T}_b$, one has to multiply the given matrices:

$$ {}^c\boldsymbol{T}_b = {}^c\boldsymbol{T}_a {}^a\boldsymbol{T}_b. \tag{4.3}$$

With `KinematicMatrix` objects $b2a$ and $a2c$, this transformation would be equivalent to $b2c = a2c \cdot b2a$ in the HULKs' software framework. For a kinematic chain—a series of connected limbs—multiple transformations can be done by multiplying the respective kinematic matrices with matching sub– and superscripts.

4.4 Forward and Inverse Kinematics

The relationship between joint angles and kinematic matrices of a part of a mechanical system is quite important. In the HULKs' framework joint angles are represented by vectors of type *std::vector<float>*. Depending on whether a set of joint angles or kinematic matrix is given, the other one can be calculated accordingly by either forward or inverse kinematics, [1]. Since these methods are vital to the implementation of a kick, they shall briefly be explained here.

Forward Kinematics

Position and oriantation of a limb can be calculated from given joint angles by forward kinematics. For example, this is required when one wants to calculate the CoM for a given set of joint angles. In the HULKs' framework kinematic matrices for each limb can be calculated via forward kinematic by methods of class *ForwardKinematics* that have a vector of joint angles as input.

Inverse Kinematics

Inverse Kinematics on the other hand refers to calculate joint angles for a desired position and orientation of a limb. This can for example calculate the joint angles for a leg that correspond to a certain foot position. There exist analytical and numerical methods for this. The existing implementation for the NAO utilizes an analytical one. To calculate joint angles from a given kinematic matrix in torso coordinates methods of class *InverseKinematics* can be called. For the leg, there is a peculiarity since the NAO has only one motor to control both left and right hip yaw pitch (see section 4.1). Therefore, there are two methods for inverse kinematics for each leg, one that is called with a set hip yaw pitch and one without it.

5 Implementation

Prior to any implementation on the NAO, the DMPs' approach to generate trajectories is tested in MATLAB. Here, a kick is learned from a demonstrated kick motion. Different modifications are tested and assessed. Afterwards, the implementation on the NAO is described and important features are explained. This includes the implementation of DMPs and controllers utilizing gyroscope feedback as well as a way to hold the CoM above the support foot.

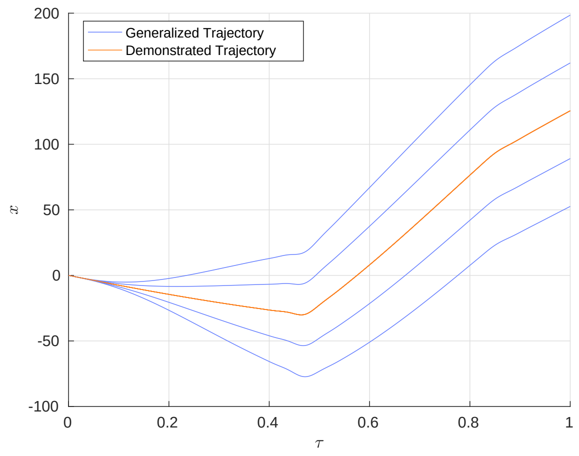
5.1 Dynamic Movement Primitives in MATLAB

To test DMPs, MATLAB scripts that implement the components are written. The solution of the canonical system (see section 2.8) and the basis functions (see section 2.10) is calculated. For trajectory learning, τ is set to the duration of the demonstrated motion. The time constant α_x is calculated so that $x(\tau) = 10^{-3}$ and the constants K and D can be set to arbitrary values so long as the system is still critically damped, [15]. The differential equation that is the transformation is solved by the Euler method. The weightings w_i are obtained from a demonstrated trajectory with locally weighted regression, [10].

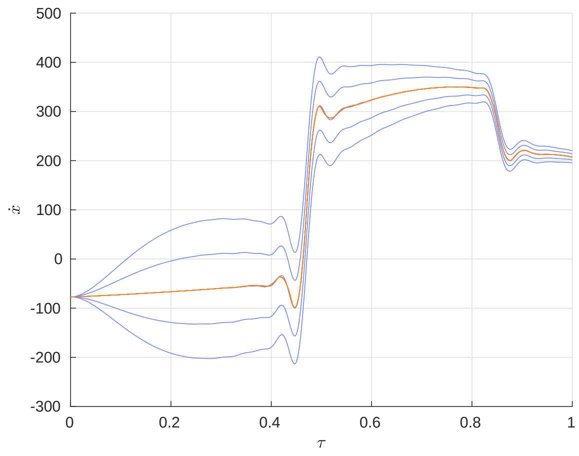
5.1.1 Foot Trajectory from Interpolated Key–Frames

A kick motion is learned from the existing key frame based motion previously used by the HULKs. To accomplish that a program that plays a `MotionFile` and interpolates between the key frames is written. Motion files consists of vectors of joint angles with a relative duration and a total duration. The elements of each vector correspond to joint angles with numbers 0 to 25. Also, the stiffness of the joints is specified. For each point in time the angles are interpolated linearly between the previous and next key frame. The position vector for the kicking foot can be calculated from these angles with forward kinematics. The calculated positions are relative to the torso. As explained in section 3 the end effector trajectory should be relative to the support foot coordinates; a coordinate transformation is necessary. The transformed position vectors are written into a `.txt` file for importing them to MATLAB.

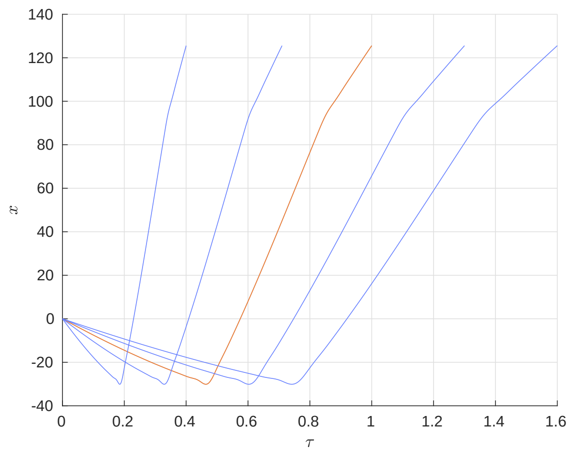
With this trajectory weightings can be learned. However, the whole kick motion begins and ends in the *ready pose*—where the NAO stands upright with bent knees—or close to it. If one were to scale the final position or velocity, one would not be able to scale it to meet requirements imposed on the kick since the point when foot and ball collide is not controlled directly. For that reason, only the part of the motion where the foot is lifted and swings forward is extracted. This allows a more usefully generalized kick. Sample code that computes the trajectory for the left kick that has been used thus far is given in appendix B.



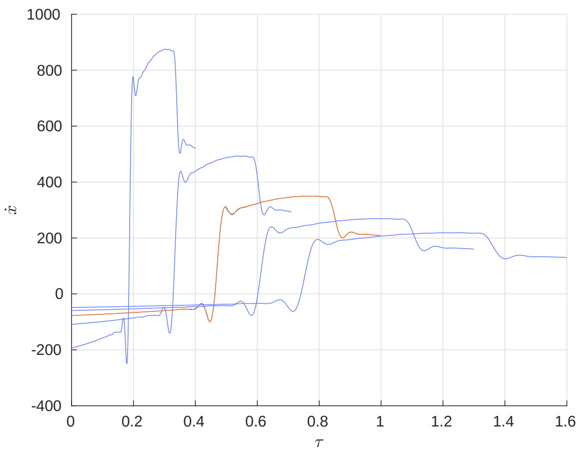
(a) Changing target, constant duration



(b) Changing target, constant duration



(c) Constant target, changing duration

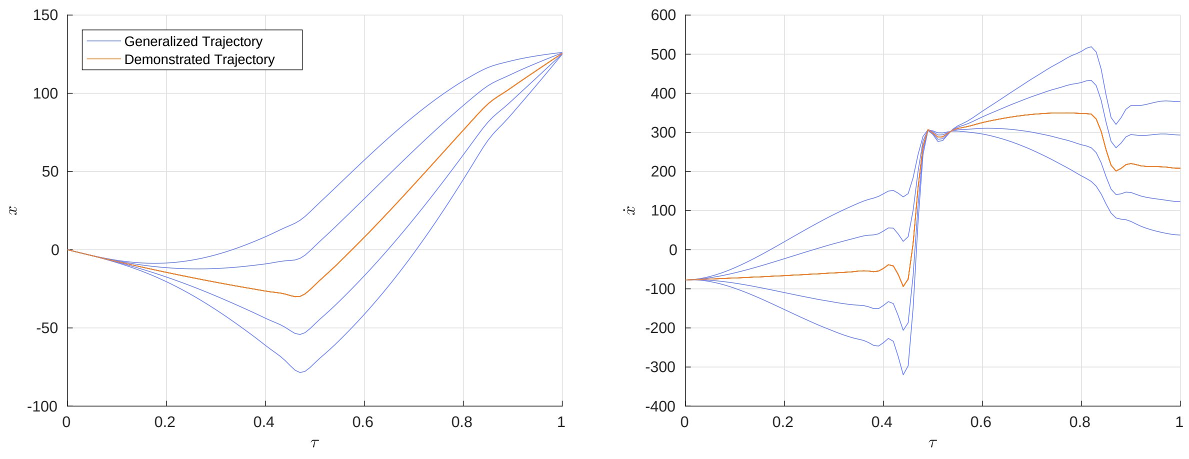


(d) Constant target, changing duration

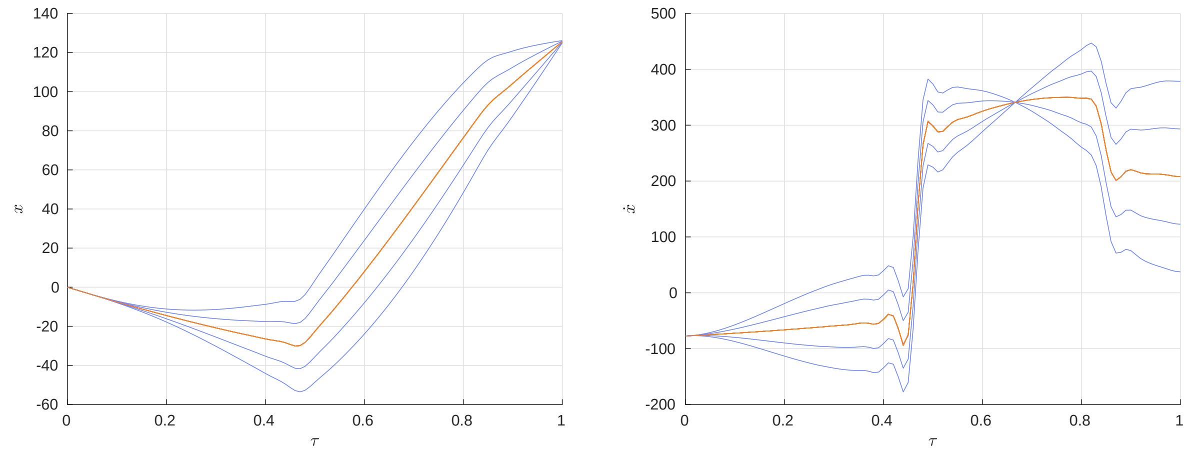
Figure 5.1: Position x (left) and velocity $\dot{x}$ (right) for different target positions $x(\tau)$ (top) and different durations τ (bottom). The transformation system has a fixed target g.

5.1.2 Transformation System and Modifications

In the following, the transformation systems as of 2.13, 2.14 and 2.19 are explored and their usefulness for kicking is assessed; generalizability in terms of final end effector position and velocity is tested. With a given trajectory as of section 5.1.1 the nonlinearity f_{learn} can be calculated; it has to match the transformation system though. The duration is normalized to 1. The target position, target velocity and duration are changed and the resulting $x(t)$ and $\dot{x}(t)$ are presented in figures 5.1 and 5.2. The position and velocity in y–direction can be found in appendix C. The z–direction is omitted because it is the least interesting for dynamic kicks.



(a) Constant target position, changing velocity (b) Constant target position, changing velocity



(c) Constant target position, changing velocity (d) Constant target position, changing velocity

Figure 5.2: Position x (left) and velocity $\dot{x}$ (right) for different target velocities $\dot{x}(\tau)$ without forcing term amplitude A (top) and with A (bottom). The transformation system has a moving target $g_p(t)$.

Modified Transformation System

At first, a transformation system as of equation 2.13 with constant targets is implemented. The system still reaches the desired state while having a shape resembling the original motion. Figure 5.1a and 5.1b show the behaviour under spatial scaling. As expected the final velocity stays constant.

Temporal scaling can be used to influence the final velocity. Figure 5.1d shows this correlation. Here, the duration τ is changed. Evidently, the relation between final velocity $\dot{x}(\tau)$ and temporal scaling τ is $\dot{x}(\tau) \propto \frac{1}{\tau}$. This means that if one wants to double the final velocity, the duration is to be halved. The shape of $x(t)$ stays the same; there is just a dilation because of the changed duration (see figure 5.1c).

Moving Target without Amplitude

Secondly, the transformation system with equation 2.14 is tested. Spatial scaling of x works in the same way as in the previous section and is therefore not illustrated here. It has direct control over the velocity $\dot{x}(\tau)$ at the end of the kicking motion. In figures 5.2a and 5.2c the position and velocity in x–direction for varying moving target velocities are shown. The desired final state is reached but apparently the shape of the trajectory deviates from the original one. Depending on the magnitude of the deviation this could be problematic; limited joint angles might not allow to follow the trajectory or limbs could collide. This shortcoming can be addressed by multiplying the nonlinearity by an amplitude that takes into account the desired and the original velocity.

Moving Target with Amplitude

With the forcing term amplitude of 2.18 in transformation system 2.19, the trajectory can maintain its original shape in the face of changing velocities. Figures 5.2b and 5.2d illustrate this. However, the trajectory still differs from the original one.

5.1.3 Assessment of the Different Transformation Systems

Transformation system 2.14 is clearly superior to 2.19 since the amplitude reduced the divergence from the original trajectory. When it comes to assessing the one with fixed target, the answer is not so obvious.

From figures 5.1 and 5.2 it can be concluded that the requirements of adapting to ball positions (1) and changing the kick strength (3) can both be fulfilled by both transformation systems. A dynamic kick should also be able to kick the ball in any direction as defined in requirement 2. The collision of ball and foot can be thought of as a collision of two spheres. [6]. When the foot kicks the ball, the tangent of its contour determines the direction of the kick. As can be seen in figure 5.3, any spatial scaling applied still results in a trajectory that resembles a forward kick. To kick a ball in a specified direction, the foot has to move in this direction when it hits the ball. This can not be done with the original DMPs formulation. A moving target as explained in section 2.3.2 is able to accomplish this by changing the velocities of the kick independently for each dimension. However, $\dot{y}_1$ in equation 2.15—for the transformation system for the y–direction—has to ba scaled with a large factor to have a significant impact on the kick direction since it is far smaller than $\dot{y}_1$. This causes large deformations of the trajectory even when a forcing term amplitude is used. Figure 5.4 showcases this problem. It is nontrivial to say whether a trajectory is still feasible with changed final velocities. Because of this, an omnidirectional kick is is not directly realizable with any of the DMP formulations mentioned.

Both equation 2.13 and 2.19 match some requirements of a dynamic kick by allowing to set final position and velocity and thereby kick a ball from any position with desired velocity. But neither is able to feasibly control the direction of the kick. Therefore, the simpler formulation as of equation 2.13 is chosen for the implementation on the NAO.

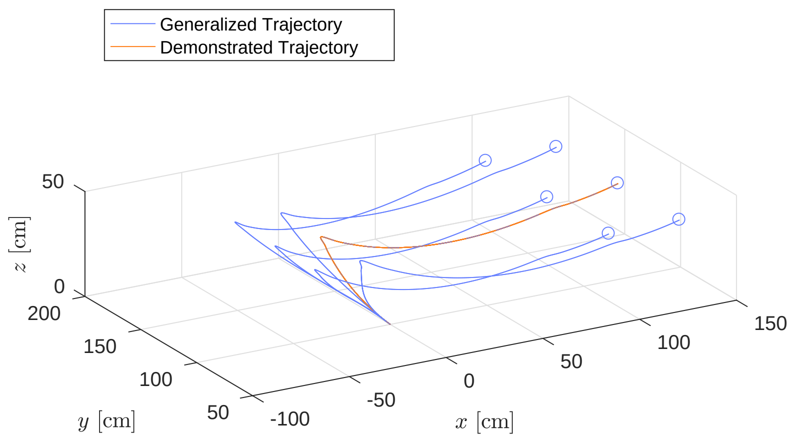


Figure 5.3: With DMPs the kick motion can be adapted to different ball positions. The motions bends naturally to adjust the reach back motion for different ball positions.

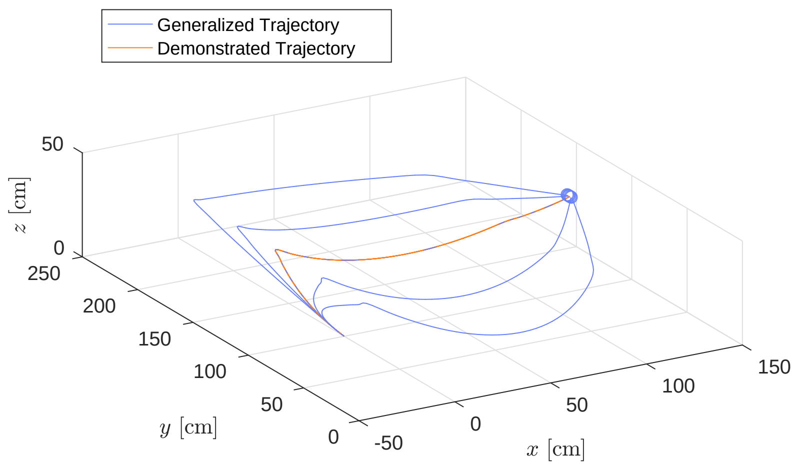


Figure 5.4: The ability to scale the final velocity of x and y independently different kick angles can be achieved. However, the motions bends and can become inexecutable.

5.2 Implementation on the NAO Robotic System

With the results from the previous section 5.1.3, a dynamic kick is implemented on the NAO in C++. It kick is implemented as the `Kick` module in the HULKs' module architecture. Each module has a `cycle` that is executed periodically, [22]. The motion modules all have a frequency of 100 Hz and thus, the `cycle` time is 10 ms. During each `cycle`, the `kick` module checks for a `MotionRequest`. A `MotionRequest` describes the desired properties of the kick that is to be executed. If a `MotionRequest` is sent from the `Brain` module, a kick motion starts. At this point a decision regarding which type of kick to execute is made depending on ball position and desired direction and strength of the kick. The parameters that generalize the kick motion are set to match the demands of the `MotionRequest`. One of the feet is assigned the role of the kicking foot, the other one will be the support foot. This decision is based on the ball position relative to the torso. The weightings for the left and right that were calculated in MATLAB (see section 5.1.1) are stored in a `json` file. The same is true for the duration and final velocity of the demonstrated trajectory. They are required to calculate the scaling parameters.

If a kick has been initialized, the next kicking foot position is calculated during each `cycle`. Then, the joint angles are calculated with methods of the `InverseKinematics` module. They are adjusted so that the CoM is at a desired position and a control signal is applied to either the ankle or the hip joints of the support foot. The learned motion described in section 5.1.1 does not return the initial position. Thus, motion phases are formed around the motion generates from DMPs.

5.2.1 Kick Motion Phases

The kick is divided into motion phases based on similar to [8]. In each phase, a part of the kick is executed while a controller aims to maintain balance. There are six phases in total. This allows precise adjustments to the individual phases.

1. **Attain Balanced Pose**
   The torso is shifted so that the CoM ground projection is at a specified point within the support foot's sole.

2. **Lift Kicking Foot**
   The single–support phase is commenced by lifting the kicking foot.

3. **Swing Leg to Kick**
   The actual kick motion that strikes the ball. The final position and velocity of this phase are those that are adapted to different situations.

4. **Retract Kicking Leg**
   The leg that kicked is retracted. This compensates the angular momentum that results from swinging the leg to some extent.

5. **Extend Kicking Foot**
   The foot is extended to establish double support again.

6. **Center CoM**
   The torso is shifted back so that the CoM ground projection is between the two feet. After this, the NAO is back in its initial pose again.

Phase 2 and 3 are covered by DMPs and are only mentioned to conceptually separate the lifting and swinging. For the CoM shifts 1 and 6 the position relative to the support foot is interpolated for a given duration and joint angles are calculatped with inverse kinematics (see section 4.4). The leg retracting 4 and extending motion 5 is generated from interpolating joint angles that specify desired poses. The first four phases happen sequentially. It has turned out to be successful to shift the CoM back while simultaneously extending the leg.

With the results from the previous assessment (see section 5.1.3), a dynamic kick is implemented on the NAO. The DMPs framework with a transformation system 2.13 is implemented in a method of class `Kick`. When called, it calculates the kicking foot's position for the current `cycle` and updates the `KinematicMatrix` $kicking2support$ from the calculated position of the foot. It used a transformation system with a vector representing the three spatial dimensions. For vector and matrix operations the C++ library "Eigen" is used. The orientation is set to an identity matrix which means the "toes" face forwards. As explained in section 4.3 the trajectory is calculated relative to the support foot but to calculate the joint angles for the current cycle they need to be relative to the torso. The required coordinate transformation is done by multiplying two `KinematicMatrix`, i. e. $kicking2torso = support2torso \cdot kicking2support$.

For the interpolation, an `Interpolator` class is used. An object of that class is constructed with an initial vector, a final vector and a duration. A method `step` calculates intermediate values in each cycle, hence creating a motion in a manner very much resembling key frames (see section 2.1). The `Interpolator` class can also interpolate the position of the CoM. This is used for the CoM shift of phases 1 and 6.

As the robot shifts the CoM, the torso has a certain velocity and therefore a momentum. When the motion stops abruptly, this momentum can cause the NAO to tip over, especially since the support foot is rather small along the y–axis. To reduce the risk of tipping over the `step` method of the `Interpolator` is called with a time step with varying size. Depending on the progress $\frac{t}{T}$, where t is the current time and T is the duration of the phase, the time step is adjusted. A sinusoidal time step $\mathrm{d}t_{adjusted} = 1 + cos(\pi \frac{t}{T})$ accomplished this and makes sure that the size of the time step is smaller as the motion nears its end.The same adjustment is done to the interpolation in phase 5, where the kicking leg is extended again. If the leg is extended at constant speed, the collision upon touching the ground can severely impair the ability to balance. Therefore, the size of the time step is decreased as the foot nears the ground. This way the speed diminishes as the foot draws near the ground and contact is made more smoothly.

5.2.2 Balance Center of Mass

With the kick motion settled, a way to maintain balance is implemented. As a first measure to maintain balance while kicking the torso is shifted during phase 1 so that the CoM is above the support foot. During the rest of the kick the CoM is held at this position until finally being moved back to the initial position in phase 6. The CoM is calculated from the current joint angles as measured by the MREs and masses of all limbs. Since the CoM is calculated relative to the torso but required to be relative to the ankle of the support foot, a coordinate transformation is required. The transformation matrix is acquired from angle measurements of the IMU, which describe the inclination of the NAO. The measured roll and pitch of the torso are used to project the CoM to the xy–plane that is the ground. Alternatively, the transformation matrix could be obtained from the current joint angles, but they do not take the torso inclination into account, [17]. The IMU angle data are low–pass filtered as described in section 4.2 with $\alpha = 0.2$.

During the kick, the CoM changes because the kicking leg is moving. To keep the CoM above the support foot, the angles of the support leg are recalculated. The kinematic matrix *kicking2support* has to stay the same in order to properly kick the ball. A method `balanceCoM` is implemented to do this. It iteratively calculates joint angles for both legs that result in the CoM being at a desired position. The error of the desired and current *com2support* position is used to adjust the torso position, [8]. With the CoM projection lying within the support polygon, static balance is established (see section 3.1).

5.2.3 Gyroscope Feedback Control

Holding the CoM projection within the support polygon can balance the NAO if it is almost standing still. This is not the case during kicking. Therefore, a means to establish dynamic balance (see section 3.2) has to be implemented.

The ankle controller is implemented as a method of the `Kick` class. In each `cycle`, after all joint angles are calculated and then corrected by the `balanceCoM` class explained in 5.2.2, a P–controller as of equation 3.4 is implemented to adjust the ankle roll and pitch. The gyroscope readings are low–pass filtered (see section 4.2) to deal with noisiness.

The gains for the controller have to be determined. To do this, the Ziegler–Nichols method is applied. For the Ziegler–Nichols method, the proportional gain K_P of a closed–loop system without integral or derivative feedback is gradually increased until the system becomes unstable. At this point there will be an oscillation. The critical period P_{cr} is measured and the critical gain K_{cr} at which this oscillation occurs is noted. The gains are then calculated from the following table, [18].

	K_P	T_I	T_D
P	$0.5 \cdot K_{cr}$	∞	0
PID	$0.6 \cdot K_{cr}$	$0.5 \cdot P_{cr}$	$0.125 \cdot P_{cr}$

For the ankle roll controller the critical gain is $K_{cr} = 0.23$ and the critical period is $P_{cr} = 7.8\,\text{ms}$. The gains for P– and PID controller are calculated according to table 5.2.3. To address integrator wind up the control signal is limited to a set interval. The pitch controller is equivalent to the one currently used in the HULKs' walking engine and the proportional gain is set to the same value.

The hip controller that was described in section 3.3.2 can be implemented in a similar fashion. However, it was found that using the hip joints to maintain balance often causes the legs to collide while kicking. Because of this, the idea of adjusting hip roll and pitch joints to maintain balance is discarded.

6 Evaluation

The implemented kick is evaluated with respect to generalizability and stability. Different ball positions relative to the support foot are tested. Also, the distance the ball covers is measured for different parameters and the stability during the kick is evaluated. Because the NAO is (almost) symmetrical in its joints and links, measurements are only taken for the left foot.

6.1 Kick Strength and Temporal Scaling

At first, the scalability of the kick strength is tested (see requirement 3). The distance covered by the ball can be controlled indirectly by increasing or decreasing the duration, which in turn leads to a decreased or increased velocity of the foot. The simulations in MATLAB (see 5.1.2) suggested that the correlation between final foot velocity $\dot{x}(\tau)$ and duration τ is $\dot{x}(\tau) \propto \frac{1}{\tau}$. With this in mind, kicks are executed with different durations and the distance covered is measured. The mean value and standard deviation for the distance covered in x– and y–direction are calculated. The measurements for 10 kicks for each chosen duration τ is portrayed in figure 6.1, whereby τ is changed in steps of 50 ms between 150 ms and 500 ms. It can be seen that the distance covered does indeed correlate with the duration, although the distance covered in not necessarily inversely proportional to τ. For $\tau = 150\,\text{ms}$ the distance covered is smaller than the distance covered for $\tau = 200\,\text{ms}$. At this point, the motion becomes too fast for the motors the execute it. Therefore, the minimum duration is set to 200 ms. The maximum duration is set to 500 ms because at that point the mean distance covered is only about 70 cm and at that point, dribbling is more desirable than kicking.

In an idealized case there is no distance covered perpendicular to the kick direction. But a real kick will have such a deviation from the desired behaviour. To measure the accuracy, the error of distance the ball covered in y–direction is measured, where y is the axis perpendicular to the kick direction x. The mean and standard deviation of the angle α that results from this deviation is calculated.

The empirical relationship between distance covered in centimetres for $200\,\text{ms} \leq \tau \leq 500\,\text{ms}$ is approximated by $x = 0.0015\tau^2 - 1.6467\tau + 510$. One can observe that the standard deviation increases the greater the distance covered is.

The angle deviation is depicted in figure 6.2. An angle of $\alpha = 0$ indicates a kick in x–direction. The mean angle is close to zero, but the standard deviation is high. Unexpectedly, the angle deviation does not decrease for shorter distances covered.

Both the error of the distance covered and the kick angle can be the result of the ball being kicked at different segments of the trajectory. Also, the quality of the kick depends strongly on the exact ball position.

The exact interaction with the ball as well as friction the ball experiences as it rolls on the artificial turf are unknown. But with the empirical relationship, a `MotionRequest` sent by the `Brain` can be properly executed by translating the specified desired distance

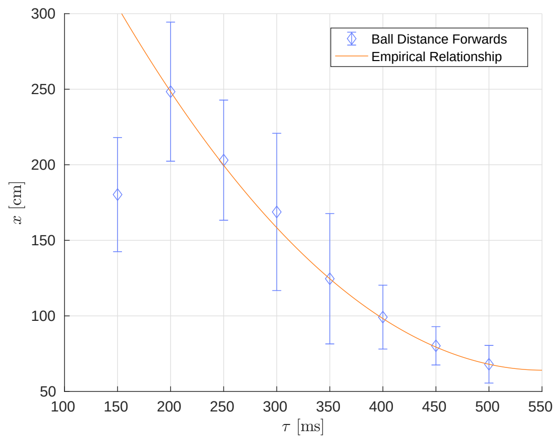


Figure 6.1: The mean and standard deviation of the distance covered by the ball of forward kicks with different durations.

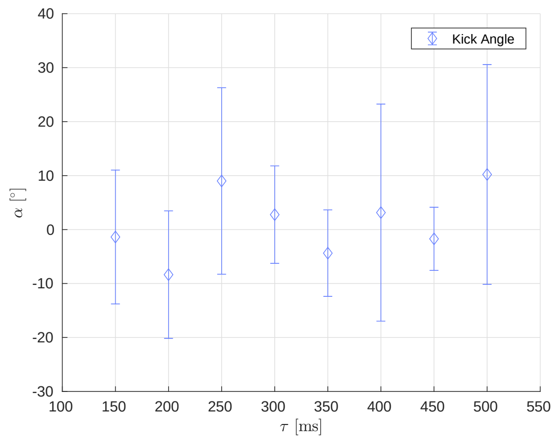


Figure 6.2: The mean angle of the forward kicks with different durations and their standard deviation.

covered to a necessary duration. Albeit both the covered distance and angular deviation are high, this makes it possible to pass the ball to a team member if that robot's position is known and within reach.

6.2 Ball Position and Spatial Scaling

With the DMP framework, the kick should be able to kick the ball from different positions and thus meet requirement 1. The generalized trajectories for different end effector positions are depicted in figure 5.3. To test this for the robot, the ball is put in front of one foot with an offset in $x-$ and/or $y-$direction. The distance from the center of the ball to the ankle is measured by hand. Then, the parameters that determine the final end effector position are set so that its trajectory should traverse the ball. A straight forward kick is performed and the process is repeated multiple times for different ball positions. The results for 4 different y_{Ball} positions relative to the support foot are depicted in figures 6.3 and 6.4. The mean distance covered is approximately the same for ball positions ranging between 60 cm and 150 cm. The same holds for the kick angle α. For $y_{Ball} = 200$ cm the distance covered decreases drastically. This can be explained with the limited range of the joints.

There are physical limitations to the generalizability of the kick. The joints operate in a limited range and the dimensions of limbs are constant. Thus, only a confined area is reachable by the foot; it is called the end effector workspace, [23]. The workspace boundary is denoted work envelope. This limits the spatial scaling capabilities of the DMPs. Upper and lower bounds for the parameters that control the trajectory are introduced to avoid this. With the simplifying assumption made in section 5.1.3, the foot is assumed to kick the ball with the center of the tip of the foot if the motion is scaled correctly. The scaling parameters that specify the ankle position of the kicking foot are only compatible with

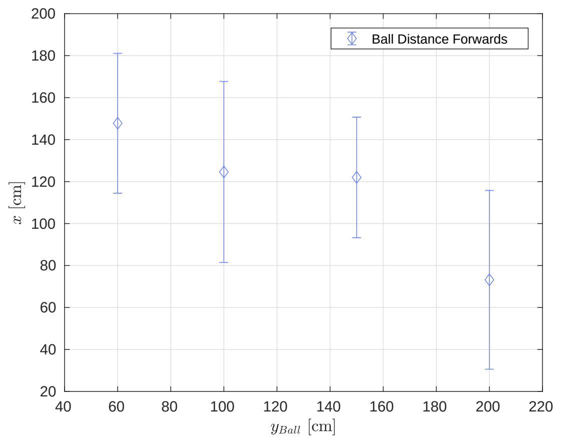


Figure 6.3: The mean and standard deviation of the distance covered by the ball of forward kicks with different initial ball positions relative to the support foot.

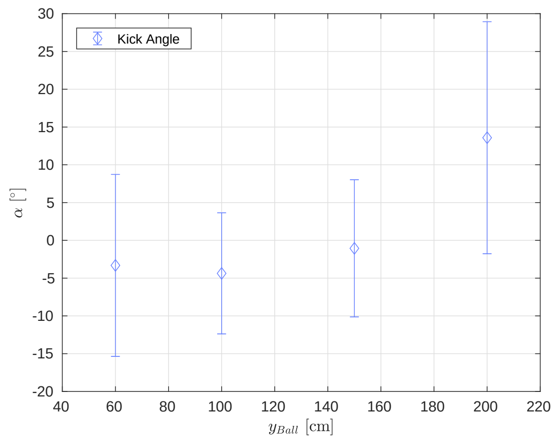


Figure 6.4: The mean angle of the forward kicks with different initial ball positions and their standard deviation.

the ball position given by `MotionRequest` if the relevant foot length—the distance from its toe to its ankle—is subtracted from the ball position.

To estimate the work envelope, a MATLAB script is written. It calculates the end effector position in the workspace through nested for–loops. Each loop increments a joint angle of the kicking leg. For each combination of angles, the position is calculated by forward kinematics (see section 4.4). To avoid dependency on too many joint angles, the workspace of the kicking foot is calculated relative to the torso and not the support foot. It is assumed that the boundaries derived are still applicable since the CoM—which is held above the support foot—most of the time lies within the torso since it has a relatively high mass and the limbs are symmetrical. The official ball used in the SPL of the RoboCup 2017 has a diameter of 100 mm, [3]. The effect of striking the ball at different heights is not investigated here. With this in mind, the workspace at a height about 35 mm to 45 mm is used to estimate boundaries for spatial scaling in the transversal plane. The actual kicking height is set to 40 mm. The points of the end effector form a three–dimensional scatter plot, but only the x–y–plane is taken into account here. Figure 6.5 displays the workspace of the left foot with at the defined height.

Figure 5.1a suggests that the when kicking the foot swings forward for about 150 cm regardless of the spatial scaling. If this swing distance is reduced by the limited workspace of the foot, the kick strength is impaired. This explains the reduced kick strength observed for the outermost ball position. With the workspace calculation, limits that determine minimum and maximum spatial scaling are set. They are set as absolute values of positions of the ankle relative to the support foot. For the left foot, these boundaries can be seen in figure 6.5. The frontal limit is approximated by a quadratic function. To retain the full kick strength, the limit for the maximum y_{Ball} position is set accordingly. The boundaries for the right foot are obtained by mirroring the boundaries for the left foot.

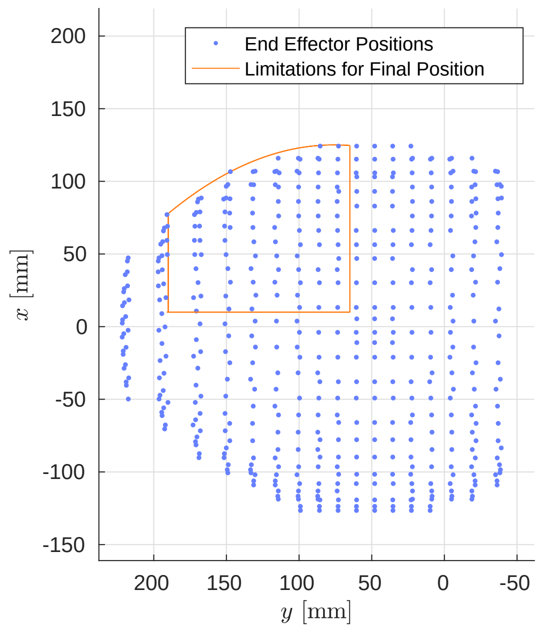


Figure 6.5: The workspace of the left foot to estimate reachable space.

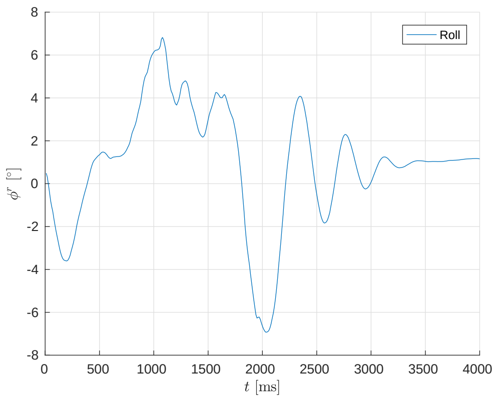
6.3 Balance While Kicking

The balance during kicking is assessed by performing kick with varying temporal and spatial scaling. During previous tests it was observed that the saggital plane is almost never an issue, whereas balance in the coronal plane is harder to maintain. This is due to the dimensions of the feet. The length in x–direction of about 15 cm far exceeds the width in y–direction of about 7 cm and therefore the NAO is more likely to tip over about the x–axis.

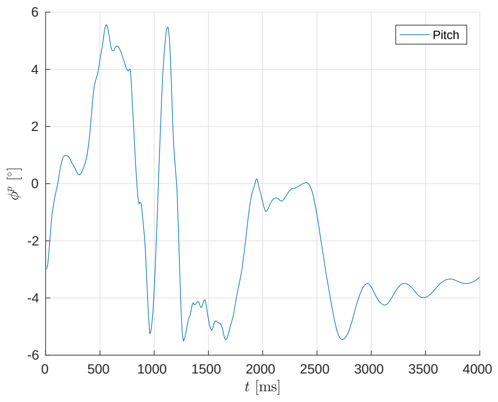
While testing the implemented kick, it has been observed that the NAO is seemingly not symmetrical, for maintain balance during right kicks appears to be more difficult than during left kicks. When right kicking, the NAO would frequently fall over to the left. If one consults the manufacturer's documentation, one finds out that the joint ranges of the left and right leg are slightly different. This can potentially explain the observed discrepancy. To cope with this, the desired CoM that is held during the right kick (see section 5.2.2) is adjusted manually.

The implemented PID–controller was tuned in compliance with the Ziegler–Nichols method. However, the PID-controller produces large overshoot and is often in saturation. On the other hand, the P–controller tuned with the Ziegler–Nichols method provides more promising results. This might indicate that the critical period is erroneous. In the remainder of this section, only the P–controller is tested.

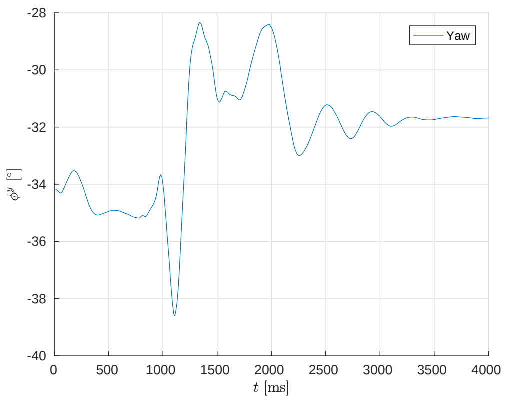
For each foot, 50 kicks are performed. The location of the NAO on the pitch is changed to compensate variances of the artificial turf. With the adjusted desired CoM position for the right kick as it was described above, the NAO did tip over 7 times across the 50 kicks.



(a) Roll angle during a left kick



(b) Pitch angle during a left kick



(c) Yaw angle during a left kick

Figure 6.6: Roll, Pitch and Yaw during a Kick

The results for left kicks is even better with the NAO only tipping over 5 times out of 50. It occurred not once that the balance is the sagittal plane could not be maintained. For reasons states above, the coronal plane is more problematic.

Figures 6.6a and 6.6b show the roll and pitch of the NAO's torso during a kick. It can be observed that as the actual kick motion (see section 5.2.1, phase 3) happens at about 1000 ms, there is a sudden change of the pitch. This is caused by the moment induced from the rapid foot motion. The robot swings after kicking. This is partially due to the backlash of the joints. The change from the initial pitch of $-3°$ to $5°$ can be explained by the fact that the CoM is not only shifted in $x-$ but also in y–direction. In the *ready pose*, the NAO is inclined slightly backwards and leans forward a bit when initializing a kick so that the ground projection of the CoM lies closer to the center of the support polygon. The P–controller is able to ensure that both the roll and pitch return to their original value.

In section 3, it was assumed that the NAO does not rotate about the z–axis. This is confirmed for the implemented kick. Figure 6.6c shows the yaw angle while kicking.

Although there is a divergence between the initial and final yaw angle, the magnitude small enough to be neglectable.

7 Conclusion and Outlook

This thesis examined the applicability of DMPs for kicking motions on the NAO. A kick motion was learned and could be generalized to new situations. Thus, much time is saved because careful positioning—as it was required with the previous kick based on key frames—is no longer necessary. The NAO is also able to engage in team behaviour because the controllable kick strength makes passing the ball to team members possible. A kick motion generated from DMPs could also be incorporated into the walking engine of the HULKs' framework for flexible in walk kicks.

The deviation of the distance covered in kick direction and the spurious distance covered in the direction perpendicular to the desired one is rather large. Among other factors, it is caused by irregularities of the artificial turf and of the ball. Even with a model for the rolling friction and other properties, the variance is hard to reduce.

As of now, the generalizability of the kick direction is not possible. This can be done by using a Mixture of Motor Primitive (MoMP) as it was proposed by Mülling et al., [4]. The MoMPs consists of a weighted sum of MPs such as DMPs. Due to this fact, a forward kick combined with a side kick could control the direction of the kick by adjusting their weight. A new kick motion can be incorporated easily simply by deploying new weightings.

The trajectory that was learned was used mainly because it was available. The kick can be further improved by optimizing the demonstrated trajectory with learning algorithms, [5].

During testing the NAO could maintain balance most of the time with a simple P–controller. Using the Ziegler–Nichols method for tuning the ankle controller did not function successfully. The PID–controller could be improved by optimizing the gains with the help of learning algorithms, [19].

To further improve the ability to not tip over while executing a kick, a ZMP preview controller can be implemented. It generates a trajectory for the CoM so that the current and future ZMPs are within the support polygon. ZMP preview control has been successfully implemented by other RoboCup SPL teams, [15].

The computational time is within the limits imposed by the NAO. However, the kick is not implemented in an optimal way and the required time during each cycle can potentially be reduced.

A Symbolic Solution for Coefficients

This MATLAB script symbolically solves the system of equations with initial and final position, velocity and acceleration of the moving target introduced in 2.3.2 for the coefficients $b_0, ..., b_5$.

```
% symbolic variables
syms b0 b1 b2 b3 b4 b5 y0 y0d y0dd y1 y1d y1dd tau;

% system of equations
eq1 = y0 == b0;
eq2 = y0d == b1;
eq3 = y0dd == 2*b2;
eq4 = y1 == b5*tau^5 + b4*tau^4 + b3*tau^3 + b2*tau^2 + b1*tau + b0;
eq5 = y1d == 5*b5*tau^4 + 4*b4*tau^3 + 3*b3*tau^2 + 2*b2*tau + b1;
eq6 = y1dd == 20*b5*tau^3 + 12*b4*tau^2 + 6*b3*tau + 2*b2;

% solve for coefficients; display in command window
coefficients = solve(eq1, eq2, eq3, eq4, eq5, eq6, b0, b1, b2, b3, b4, b5);
[coef.b0; coef.b1; coef.b2; coef.b3; coef.b4; coef.b5]
```

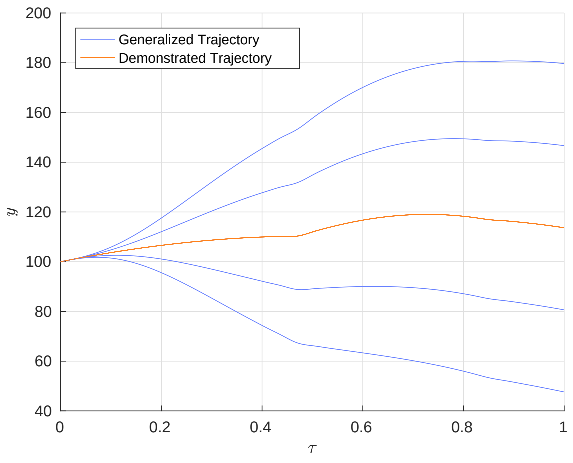
B Foot Trajectory from Key Frames

The following C++ code calculates the trajectory for the kicking foot from a given motion file. It is implemented for the left foot only but can easily be adapted for the right.

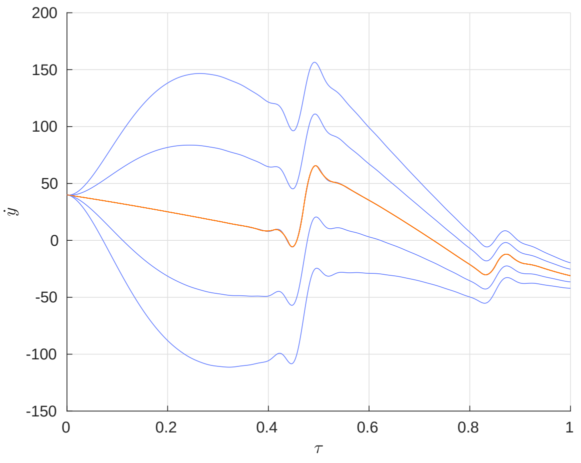
```
#include <iostream>
#include <fstream>
#include "MotionFile.hpp"
#include "ForwardKinematics.h"
#include "NaoProvider.h"

int main() {
    // load and play motion file; open file for output
    mFile::MotionFile LKickFile;
    LKickFile.loadFromFile("kick_L.motion2");
    LKickFile.play();
    std::ofstream LFile;
    LFile.open("kick_L_trajectory.txt");
    // for the duration of the motion get angles for each cycle
    while(LKickFile.isPlaying()){
        mFile::JointValues values = LKickFile.cycle();
        std::vector<float> kickingAngles(JOINTS_L_LEG::L_LEG_MAX);
        std::vector<float> supportAngles(JOINTS_R_LEG::R_LEG_MAX);
        // leg angles from whole body angles
        for(int i = 0; i < kickingAngles.size(); i++){
            kickingAngles[i] = values.angles[JOINTS::L_HIP_YAW_PITCH+i];
            supportAngles[i] = values.angles[JOINTS::R_HIP_YAW_PITCH+i];
        }
        // kinematic matrices and coordinate transformation
        KinematicMatrix kickingFoot2Torso = ForwardKinematics::getLFoot(kickingAngles);
        KinematicMatrix supportFoot2Torso = ForwardKinematics::getRFoot(supportAngles);
        KinematicMatrix kickingFoot2supportFoot = supportFoot2Torso.invert() * kickingFoot2Torso;
        Vector3<float> LKickPosition = kickingFoot2supportFoot.posV;
        LFile << LKickPosition.toString();
    }
    LFile.close();
}
```

C Supplementary Figures Illustrating Generalizability for the Learned Motion

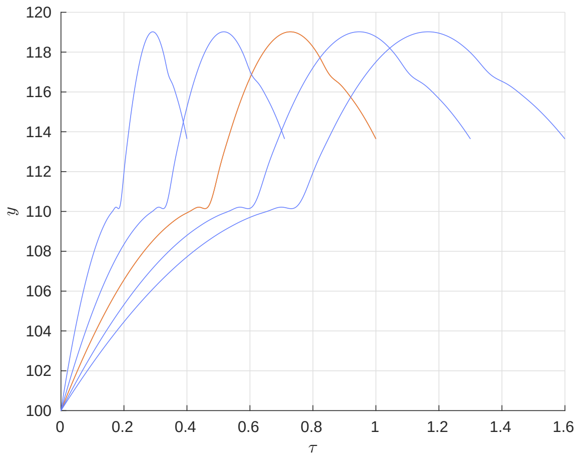
Figures C.1 and C.2 show the position and velocity of the left foot for kicks with varying durations and velocities. Figure C.1 uses the modified transformation system (see equation 2.13) while figure C.2 uses the one with a moving target without and with an amplitude for the nonlinearity (see equations 2.14 and 2.19).



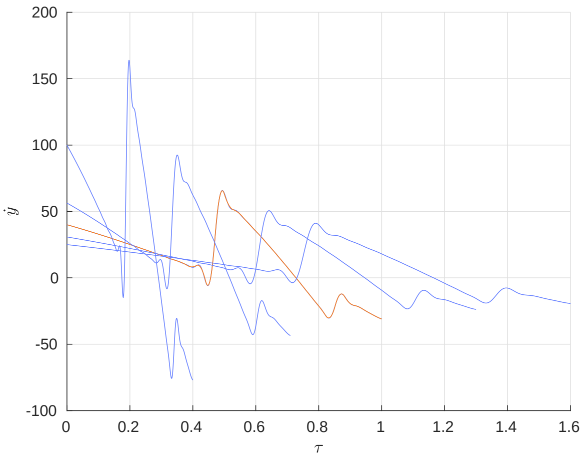
(a) Changing target, constant duration



(b) Changing target, constant duration

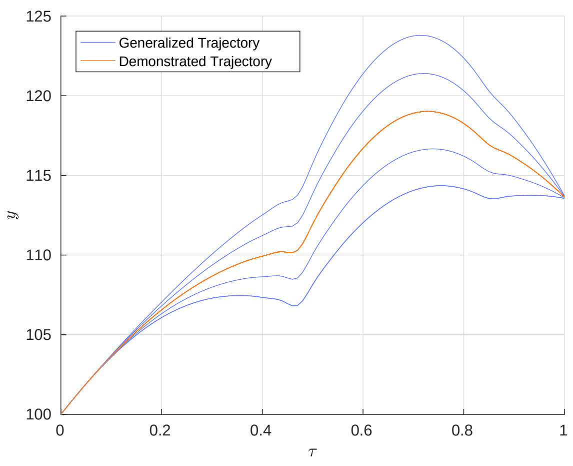


(c) Constant target, changing duration

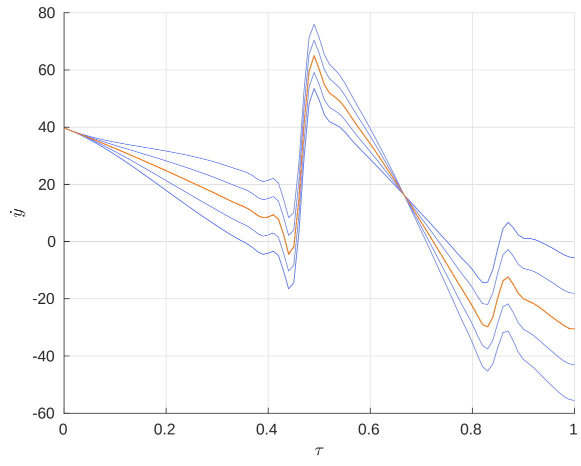


(d) Constant target, changing duration

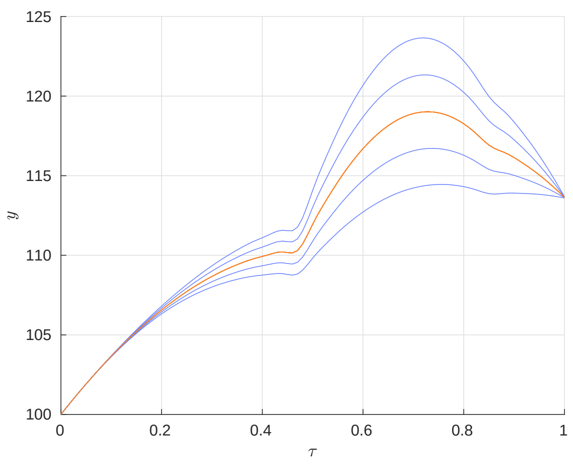
Figure C.1: Position y (left) and velocity $\dot{y}$ (right) for different target positions $y(\tau)$ (top) and different durations τ (bottom). The transformation system has a fixed target g.



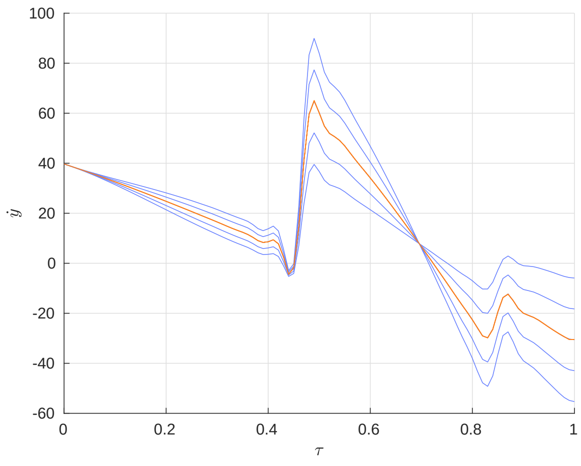
(a) Constant target position, changing velocity



(b) Constant target position, changing velocity



(c) Constant target position, changing velocity



(d) Constant target position, changing velocity

Figure C.2: Position y (left) and velocity $\dot{y}$ (right) for different target velocities $\dot{y}(\tau)$ without forcing term amplitude A (top) and with A (bottom). The transformation system has a moving target $g_p(t)$.

D Content of the CD

1. This thesis in a digital format
2. MATLAB code for Dynamic Movement Primitives with the learned trajectory and foot workspace calculation
3. The source code of the implemented kick on the NAO

References

[1] S. Kajita, H. Hirukawa, K. Harada, and K. Yokoi, *Introduction to Humanoid Robotics.* Springer, 2014.

[2] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, "Mechatronic design of NAO humanoid," in *IEEE International Conference on Robotics and Automation.* IEEE, 2009, pp. 769–774.

[3] RoboCup Technical Commitee, "RoboCup Standard Platform League (NAO) Rule Book," preliminary 2017 rules, as of November 1, 2016.

[4] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *The International Journal of Robotics Research*, vol. 32, no. 3, pp. 263–279, 2013.

[5] M. Antonelli, F. D. Libera, E. Menegatti, T. Minato, and H. Ishiguro, "Intuitive Humanoid Motion Generation Joining User-Defined Key-Frames and Automatic Learning," in *RoboCup 2007: Robot Soccer World Cup XI.* Springer, 2008, pp. 13–24.

[6] F. Wenk and T. Röfer, "Online Generated Kick Motions for the NAO Balanced Using Inverse Dynamics," in *RoboCup 2012: Robot Soccer World Cup XVI.* Springer, 2013, pp. 25–36.

[7] S. Czarnetzki, S. Kerner, and D. Klagges, "Combining key frame based motion design with controlled movement execution," in *RoboCup 2008: Robot Soccer World Cup XII.* Springer, 2009, pp. 58–68.

[8] J. Müller, T. Laue, and T. Röfer, "Kicking a ball–modeling complex dynamic motions for humanoid robots," in *RoboCup 2010: Robot Soccer World Cup XIV.* Springer, 2011, pp. 109–120.

[9] E. W. Weisstein. (2016, November 15) B–Spline. From *MathWorld*–A Wolfram Web Resource. `http://mathworld.wolfram.com/B-Spline.html`.

[10] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, *Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors*, ser. Neural computation. MIT Press, 2013, vol. 25, no. 2.

[11] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and Generalization of Motor Skills by Learning from Demonstration," in *IEEE International Conference on Robotics and Automation.* IEEE, 2009.

[12] B. Nemec and A. Ude, *Action sequencing using dynamic movement primitives*, ser. Robotica. Cambridge University Press, 2012, vol. 30, no. 05.

[13] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *IEEE International Conference on Robotics and Automation.* IEEE, 2002.

[14] H. Hoffmann, P. Pastor, D.-H. Park, and S. Schaal, "Biologically–inspired dynamical systems for movement generation: automatic real–time goal adaptation and obstacle avoidance," in *IEEE International Conference on Robotics and Automation.* IEEE, 2009.

[15] A. Böckmann and T. Laue, "Kick Motions for the NAO Robot using Dynamic Movement Primitives," in *20th RoboCup International Symposium*, 2016.

[16] M. Vukobratović and B. Borovac, "Zero–moment point—thirty five years of its life," *International Journal of Humanoid Robotics*, vol. 1, no. 01, pp. 157–173, 2004.

[17] F. O. Poppinga, "Design und Implementierung eines Regelungsverfahrens zur Stabilisierung eines humanoiden Roboters bei verschiedenen Bewegungsabläufen aus Basis des NAO–Robotiksystems," TUHH, 2015.

[18] H. Werner, *Introduction to Control Systems*, 2015, lecture Notes.

[19] F. Faber and S. Behnke, "Stochastic optimization of bipedal walking using gyro feedback and phase resetting," in *2007 7th IEEE-RAS International Conference on Humanoid Robots.* IEEE, 2007, pp. 203–209.

[20] B. Hengst, "Reinforcement learning inspired disturbance rejection and Nao bipedal locomotion," in *15th IEEE RAS Humanoids Conference*, 2015.

[21] Aldebaran Robotics, *NAO Software Documentation*, 2016, November 30, `http://doc.aldebaran.com/1-14/index.html`.

[22] N. Riebesel, A. Hasselbring, L. Peters, and F. Poppinga, "Team Research Report 2016," 2016, HULKs.

[23] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control.* John Wiley & Sons, 2006.