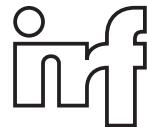


Nao Devils Dortmund

## Team Report 2016

---

Matthias Hofmann, Ingmar Schwarz, Oliver Urbann, Fabian Rensen, Aaron Larisch, Arne Moos, Janine Hemmers



Robotics Research Institute  
Section Information Technology

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Team Description . . . . .	1
1.2	Software Overview . . . . .	2
1.3	Recent Changes . . . . .	2
1.4	Getting started . . . . .	3
<b>2</b>	<b>Motion</b>	<b>4</b>
2.1	Walking . . . . .	4
2.1.1	Dortmund Walking Engine . . . . .	5
2.1.2	The ZMP/IP-Controller . . . . .	7
	Sensor Feedback Sources . . . . .	8
	Integration of the Measurements, a Sensor Fusion . . . . .	9
	Experimental Sensor Feedback Methods . . . . .	11
	Results and future work . . . . .	11
2.1.3	ZMP Generation . . . . .	12
2.1.4	Swinging Leg Controller . . . . .	13
2.1.5	Reactive Stepping . . . . .	13
2.2	Path Planning . . . . .	15
2.3	Kicking . . . . .	16
2.3.1	Precise Kick Motion . . . . .	16
2.3.2	Instant Kick . . . . .	18
2.3.3	Preview Kick . . . . .	18
2.4	Special Actions . . . . .	19
2.5	Handling the Outdoor Competition . . . . .	19
2.5.1	Controller . . . . .	19
2.5.2	Gravity and Inertia Compensation . . . . .	20
	General Considerations . . . . .	20
	Offsets . . . . .	20
2.5.3	Sensor Feedback . . . . .	21
	Observer . . . . .	21
	Orientation Controller . . . . .	22
	Linear Acceleration . . . . .	22
2.5.4	Acceleration . . . . .	22

<b>3 Cognition</b>	<b>23</b>
3.1 Image Processing . . . . .	23
3.2 Line and goal detection . . . . .	24
3.3 White goal detection . . . . .	24
3.4 Localization . . . . .	27
3.5 Distributed World Modeling . . . . .	28
3.6 Handling the Outdoor Competition . . . . .	29
3.6.1 Handling Varying Lighting Conditions . . . . .	29
3.6.2 Black-White Ball Detection . . . . .	30
3.6.3 Using Modified Drivers . . . . .	31
<b>4 Behavior</b>	<b>34</b>
4.1 Teamplay . . . . .	34
Team Strategy . . . . .	34
HeadControl . . . . .	36
4.2 Path Planning . . . . .	37
<b>5 Conclusion and Outlook</b>	<b>39</b>
<b>A Walking Engine Parameters</b>	<b>40</b>
A.1 File walkingParams.cfg . . . . .	40
A.2 File NaoNG.m . . . . .	43

# Chapter 1

## Introduction

Competitions such as the RoboCup provide a benchmark for the state of the art in the field of autonomous mobile robots and provide researchers with a standardized setup to compare their research. Additionally the RoboCup *Standard Platform League* does not only provide researchers with a common setup, but also with the same hardware platform to use. This renders increased importance to publications of those teams, since extensive documentation and especially releasing source code allows other researchers to compare results and methods, reuse and improve them, and to further common research goals.

In the course of this report some of the points of the robot software and current the research approach of the RoboCup team *Nao Devils* are described. An overview about the Nao Devils software is given in section 1.2.

This report is organized as follows. Chapter 2 describes the entire motion. Chapter 3 focuses on the perception processes of the Nao while section 4 presents the concepts and ideas for the implementation of the robots behavior. In each of those chapters 2, 3 and 4 the solutions currently in use are described in detail and further references supplied when appropriate, but also the current research is presented. Chapter 5 summarizes our current research topics and the development process for 2017.

### 1.1 Team Description

The *Nao Devils Dortmund* are a RoboCup team by the Robotics Research Institute of TU Dortmund University participating in the *Standard Platform League* since 2009 [1] as the successor of team BreDoBrothers, which was a cooperation of the University of Bremen and the TU Dortmund University [2]. The team consists of numerous undergraduate students as well as researchers. Previous team members of *Nao Devils Dortmund* have already been part of the teams *Microsoft Hellhounds* [3] (and therefore part of the *German Team* [4]), *DoH! Bots* [5] and senior team members have been part of *BreDoBrothers*.

The team was actively participating in the RoboCup events during the last range. Major successes were the 3rd places in RoboCup 2009, GermanOpen 2009/2012 and the 2nd place in RoboCup 2011. Moreover, we were able to win the outdoor competition of the RoboCup 2016. The Team also participated in most of the *technical challenges* in these years while reaching the 3rd in RoboCup 2009 and RoboCup 2013. Additionally the team ranked second and third in the Drop-In challenge 2013 and 2014 respectively.

Team *Nao Devils* reached the third place in the Drop-In competition, and the second place in the Technical challenges in RoboCup 2015. The technical challenges at RoboCup 2016 was won by the team.

Besides official RoboCup competitions the *Nao Devils* regularly participate at other international events along the year such as the RoHoW robotic workshop in Hamburg and smaller events.

## 1.2 Software Overview

The software package used by team *Nao Devils* consists of a robotic framework, a simulator and different additional tools.

The framework, running on the Nao itself, is based on the German Team Framework [6]. Since 2013 our Software is based on the Framework released by the team *B-Human*<sup>1</sup>. This year, we used the most recent released version 2015. The framework comes with a variety of improvements regarding tools and simulation performance of the framework. More details can be found in B-Humans code release 2015 () It is important to note that we exchanged most of the vital modules in motion, cognition and behaviors sections. These modules are the subject of this report in chapter 2, 3 and 4 respectively. The framework communicates with *NaoQi* using the *libBHuman*, completely separating it from Aldebaran's software modules.

Since *B-Human*'s team report 2015 [7] covers the basics and usage of the simulator in great depth, a detailed description is exclude from this report.

To test developments in simulation, the software *SimRobot* was used instead of commercial alternatives, such as *Webots* from Cyberbotics<sup>2</sup>.

Being open source offers great advantage, allowing to adapt the code to own developments. In addition having the feature to directly connect to the robot and debug online is very convenient during development. *SimRobot* [8] is a kinematic robotics simulator developed in Bremen which (like Webots) utilizes the Open Dynamics Engine<sup>3</sup> (ODE) to approximate solid state physics. Using update steps of up to 1 kHz for the physics engine enabled the possibility of realistic simulated walking experiments closely matching the gait of the real robot. It also features realistic camera image generation including effects like motion blurring, rolling shutter, etc.

## 1.3 Recent Changes

The most important change for RoboCup 2016 was the move towards a more realistic environment for playing robot soccer. This includes walking on an artificial turf and more natural lighting. An outdoor competition was formed to fosters research in this area. Naturally, most changes in code were to cope these new requirements. In this team report, we introduced several sections that addresses the aforementioned problems by describing our changes in the vision (see section 2.5, and the motion (see section 3.6).

---

<sup>1</sup>[https://sibylle.informatik.uni-bremen.de/public/bhuman12\\_coderelease.tar.bz2](https://sibylle.informatik.uni-bremen.de/public/bhuman12_coderelease.tar.bz2)

<sup>2</sup><http://www.cyberbotics.com/>

<sup>3</sup><http://www.ode.org/>

## 1.4 Getting started

This section describes the steps to set up a Nao robot to be ready to use with our code to work with the Nao and manage your robots..

- Download the code release from our website<sup>4</sup>.
- Find out the last four digits of the robot's Body ID (e.g. by searching for 'bodyid' in the robot's memory via the web interface).
- Follow the instructions in 'Install/readme.txt' to set up your robots.
- Create a configuration folder for your robot with those 4 digits and 'Nao' as a prefix (e.g. 'Config/Robots/Nao3156/') and copy the contents of 'Config/Robots/Default' into it.
- Run 'installAlcommon' (found in Aldebaran archives).
- Follow the instructions in the *B-Human* team report 2015 [9] to set up the code for compilation.
- For deploying the Code to the robots, you can use our customized version of B-Humans bush which can be found in 'Src/Utils/'.

---

<sup>4</sup><https://github.com/NaoDevils/CodeRelease2016>

# Chapter 2

## Motion

The main challenge of humanoid robotics certainly are the various aspects of motion generation and biped walking. Dortmund has participated in the Humanoid Kid-Size League during Robocup 2007 as *DoH! Bots* [5] and before in RoboCup 2006 as the joint team *BreDoBrothers* together with Bremen University. Hence there has already been some experience in the research area of two-legged walking even before participating in the Nao Standard Platform League of 2008 as the rejoined *BreDoBrothers*.

The kinematic structure of the Nao has some special characteristics that make it stand out from other humanoid robot platforms. Aldebaran Robotics implemented the HipYawPitch joints using only one servo motor. This fact links both joints and thereby makes it impossible to move one of the two without moving the other. Hence the kinematic chains of both legs are coupled. In addition both joints are tilted by 45 degrees. These structural differences to the humanoid robots used in previous years in the Humanoid League result in an unusual workspace of the feet. Therefore existing movement concepts had to be adjusted or redeveloped from scratch. The leg motion is realized by an inverse kinematic calculated with the help of analytical methods for the stance leg. The swinging leg end position is then calculated with the constraint of the HipYawPitch joint needed for the support foot. This closed form solution to the inverse kinematic problem for the Nao has been developed in Dortmund and used since RoboCup 2008 when other teams as well as Aldebaran themselves still used iterative approximations.

### 2.1 Walking

In the past different walking engines have been developed following the concept of static trajectories. The parameters of these precalculated trajectories are optimized with algorithms of the research field of *Computational Intelligence*. This allows a special adaption to the used robot hardware and environmental conditions. Approaches to move two legged robots with the help of predefined foot trajectories are common in the Humanoid Kid-Size League and offer good results. Nonetheless with such algorithms directly incorporating sensor feedback is much less intuitive. Sensing and reacting to external disturbances however is essential in robot soccer. During a game these disturbances come inevitably in the form of different ground-friction areas or bulges of the carpet. Additionally contacts with other players or the ball are partly unpreventable and result in external forces acting on

the body of the robot.

To avoid regular recalibration and repeated parameter optimization the walking algorithm should also be robust against systematic deviations from its internal model. Trajectory based walking approaches often need to be tweaked to perform optimally on each real robot. But some parameters of this robot are subject to change during the lifetime of a robot or even during a game of soccer. The reasons could manifold for instance as joint decalibration, wear out of the mechanical structure or thermic drift of the servo due to heating. Recalibrating for each such occurrence costs much time at best and is simply not possible in many situations.

The robot Nao comes equipped with the wide range of sensors capable of measuring forces acting on the body, namely an accelerometer, gyroscope and force sensors in the feet. To overcome the drawback of a static trajectory playback, team Nao Devils developed a walking engine capable of generating online dynamically stable walking patterns with the use of sensor feedback.

### 2.1.1 Dortmund Walking Engine

A common way to determine and ensure the stability of the robot utilizes the zero moment point (ZMP) [10]. The ZMP is the point on the ground where the tipping moment acting on the robot, due to gravity and inertia forces, equals zero. Therefore the ZMP has to be inside the support polygon for a stable walk, since an uncompensated tipping moment results in instability and fall. This requirement can be addressed in two ways.

On the one hand, it is possible to measure an approximated ZMP with the acceleration sensors of the Nao by using equations 2.1 and 2.2 [11]. Then the position of the approximated ZMP on the floor is  $(p_x, p_y)$ . Note that this ZMP can be outside the support polygon and therefore follows the concept of the fictitious ZMP.

$$p_x = x - \frac{z_h}{g} \ddot{x} \quad (2.1)$$

$$p_y = y - \frac{z_h}{g} \ddot{y} \quad (2.2)$$

On the other hand it is clear that the ZMP has to stay inside the support polygon and it is also predictable where the robot will set its feet. Thus it is possible to define the trajectory of the ZMP in the near future. The necessity of this will be discussed later. A known approach to make use of it is to build a controller which transforms this reference ZMP to a trajectory of the center of mass of the robot [12]. Figure ?? shows the pipeline to perform the transformation. The input of the pipeline is the desired translational and rotational speed of the robot which might change over time. This speed vector is the desired speed of the robot, which does not translate to its CoM speed directly for obvious stability reasons, but merely to its desired average. The first station in the pipeline is the Pattern Generator which transforms the speed into desired foot positions  $\mathbf{P}_{global}$  on the floor in a global coordinate system used by the walking engine only. Initially this coordinate system is the robot coordinate system projected on the floor and reset by the Pattern Generator each time the robot starts walking. The resulting reference ZMP trajectory  $p^{ref}$  calculated by “ZMP Generation” (see section 2.1.3 for details) is also defined in this global coordinate system.

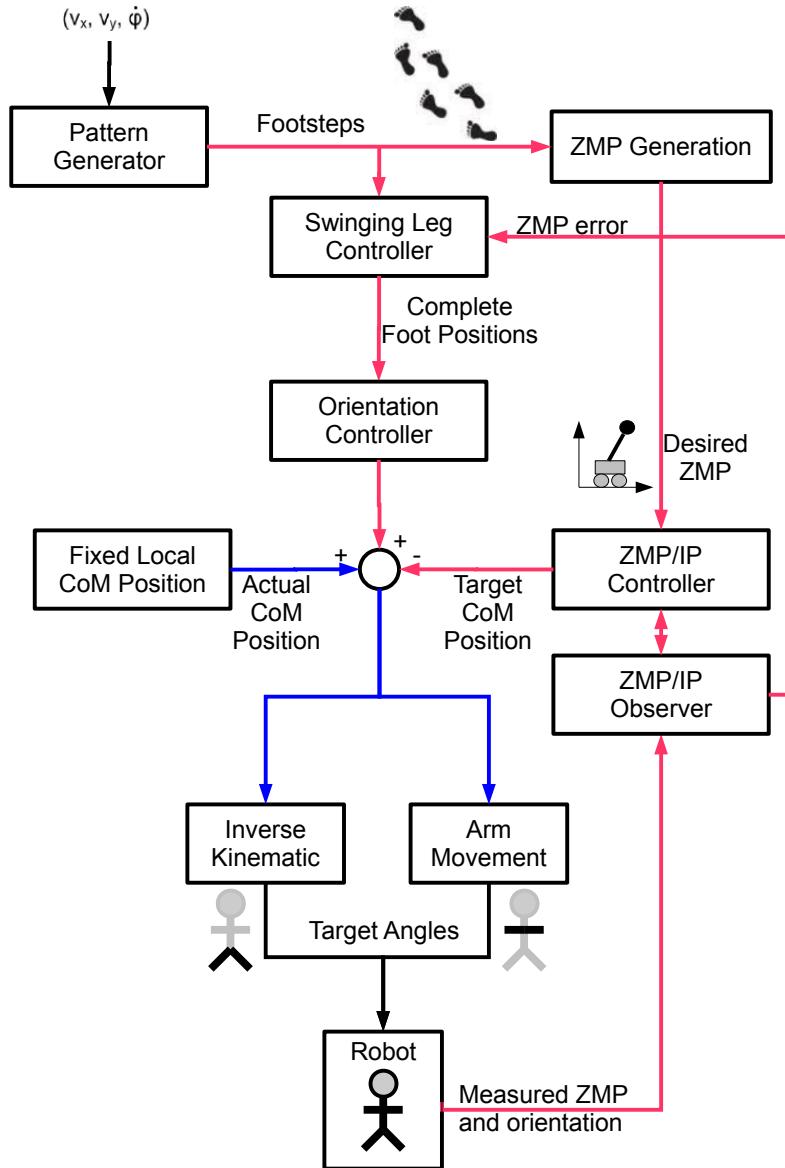


Figure 2.1: Control structure visualization of the walking pattern generation process. Data expressed in the robot coordinate system are represented by a blue line and data expressed in the global coordinate system is represented by a red line.

The core of the system is the ZMP/IP-Controller, which transforms the reference ZMP to a corresponding CoM trajectory ( $\mathbf{R}_{ref}$ ) in the global coordinate system as mentioned above. The robot's CoM relative to its coordinate frame ( $\mathbf{R}_{local}$ ) is given by the framework based on measured angles during the initial phase of the walk. After this phase  $\mathbf{R}_{local}$  is no longer updated since this would be another control loop which would cause oscillations [13]. Equation 2.3 provides the foot positions in a robot centered coordinate frame.

$$\mathbf{P}_{robot}(t) = \mathbf{P}_{global}(t) - \mathbf{R}_{ref}(t) + \mathbf{R}_{local}(t) \quad (2.3)$$

Those can subsequently be transformed into leg joint angles using inverse kinematics. Finally the leg angles are complemented with arm angles which are calculated using the  $x$  coordinates of the feet.

### 2.1.2 The ZMP/IP-Controller

The main problem in the process described in the previous section is computing the movement of the robot's body to achieve a given ZMP trajectory. To calculate this, a simplified model of the robot's dynamics is used, representing the body by its center of mass only. The ZMP/IP-Controller uses the state vector  $\mathbf{x}=(x, \dot{x}, p)$  to represent the robot where  $x$  is the position of the CoM,  $\dot{x}$  the speed of the CoM and  $p$  the resulting ZMP [11].

The system's continuous time dynamics can be represented by

$$\frac{d}{dt} \begin{bmatrix} x \\ \dot{x} \\ p \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{g}{z_h} & 0 & -\frac{g}{z_h} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ p \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} v \quad (2.4)$$

where  $v = \dot{p}$  is the system input to change the ZMP  $p$  according to the planned target ZMP trajectory  $p^{ref}$ . Discretizing equation 2.4 yields the system equation

$$\mathbf{x}(k+1) = \mathbf{A}_0 \mathbf{x}(k) + \mathbf{b}v(k) \quad (2.5)$$

where  $\mathbf{x}(k)$  denotes the discrete state vector at time  $k\Delta t$ ,  $v(k)$  denotes the controller for the system and

$$\mathbf{A}_0 = \begin{bmatrix} 1 & \Delta t & 0 \\ \frac{g}{z_h}\Delta t & 1 & -\frac{g}{z_h}\Delta t \\ 0 & 0 & 1 \end{bmatrix} \quad (2.6)$$

describes the system's behavior. Details about the controller design can be found in [14].

One important fact about the controller is, that it needs a preview of the reference ZMP. As can be seen in figure 2.2, it is not sufficient to start shifting the CoM simultaneously with the ZMP. Instead the CoM has to start moving before the ZMP does. Therefore a preview of  $p^{ref}$  is needed to be able to calculate a CoM movement leading to a stable posture.

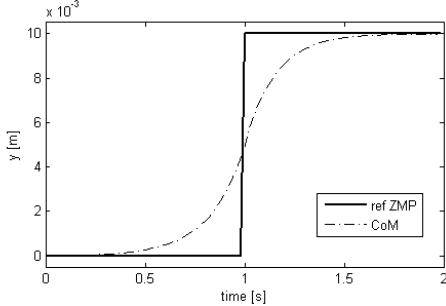


Figure 2.2: CoM motion required to achieve a given ZMP trajectory.

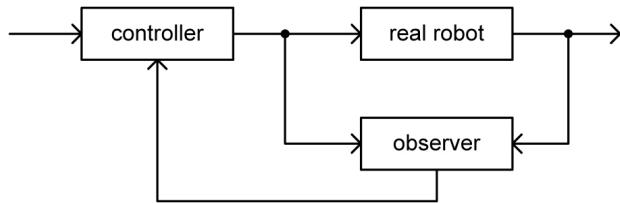


Figure 2.3: Control system with sensor feedback using an observer.

### Sensor Feedback Sources

A humanoid robot like the Nao has multiple sensors to measure the dynamical and kinematical state. Using only one of those, e.g. the ZMP measured by the foot pressure sensors or the acceleration, has some disadvantages, like noisy data or measurement delays. A common way to cope with these problems is to implement a sensor fusion using a kalman filter. Looking at the control structure of the ZMP/IP-Controller reveals that parts of the state vector can be measured, namely the position of the center of mass  $x^{sensor}(k)$  and the actual ZMP  $p^{sensor}(k)$ . The first can be calculated using the center of mass positions  $c_i$  of each link expressed in the coordinate frame of link  $i$ :

$$x^{sensor}(k) = T_s^w(k) \cdot \left( \frac{1}{\sum_l m_l} \cdot \sum_i T_{O_i}^s(k) \cdot c_i \cdot m_i \right) \quad (2.7)$$

where

- $s$  is the coordinate system of the support foot.
- $w$  is the world coordinate system.
- $O_i$  is the coordinate system of link  $i$ .
- $m_i$  is the mass of link  $i$ .
- $T_i^j(k)$  is the homogeneous coordinate system transformation matrix from coordinate system  $i$  to  $j$ .

The later is the weighted average of the data measured by each pressure sensor in the feet. In case of the Nao robot the CoP for the left  $p_{left}^{sensor}(k)$  and right foot  $p_{right}^{sensor}(k)$  are given by the API and must be combined:

$$p^{sensor}(k) = \frac{f_l(k)}{F} \cdot T_{O_l}^w(k) \cdot p_{left}^{sensor}(k) + \frac{f_r(k)}{F} \cdot T_{O_r}^w(k) \cdot p_{right}^{sensor}(k) \quad (2.8)$$

where

- $O_l, O_r$  is the coordinate system of left and right foot respectively.
- $f_l(k), f_r(k)$  is the force exerted on the left and right foot respectively.
- $F = f_l(k) + f_r(k)$  is the overall force exerted on the robot.

This kind of ZMP measurement has the same limitation as the definition of the ZMP, it is bounded to the supporting area. As a result, the ZMP will be measured at the edge of the support area when the robot tilts. The measurement of the center of mass position is limited in a similar way. The result of flexibilities and tolerances in the joints can be measured, but not the result of the tilt of the robot. This is not necessarily a disadvantage, since it limits the reaction of the robot to disturbances to securely executable movements, especially if the robots falls down. The limitation of the CoM measurement also corresponds to the ZMP measurement limits.

Regarding the coordinate system transformations, there is one noticeable point. The transformations to the world coordinate system  $T_{O_i}^w(k)$  and other transformations are separated. The reason for that is the way of calculating. The transformations between the coordinate systems of the links are calculated using forward kinematics and measured angles. However, the position and orientation of the feet in the world coordinate system cannot be measured directly. Possible sources would be the self localization or the odometry calculated by integrating the acceleration sensor, both combined with forward kinematics. But these sources are inaccurate and lead to large noise in the measured CoM and ZMP positions. Measurable errors during a walk are much lower than these measurement errors and it would be not possible to react correctly. Therefore the target foot positions and orientations given by the Pattern Generator are used for  $T_{O_i}^w(k)$ .

### Integration of the Measurements, a Sensor Fusion

In the last chapter two measurable values of the state vector  $\mathbf{x}(k)$  are presented. The measurable output of the system given by equation 2.20 can now be defined as

$$y(k) = \mathbf{c} \cdot \mathbf{x}(k) \hat{=} \begin{bmatrix} x^{sensor}(k) \\ p^{sensor}(k) \end{bmatrix} \quad (2.9)$$

with

$$\mathbf{c} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.10)$$

Since not the full state can be measured, an observer is needed. Figure 2.3 shows the overall system configuration. The observer is put in parallel to the real robot and receives the same output of the controller to estimate the behavior of the real system

and is supported by the measurements of the ZMP and the CoM. Derived from equations 2.20 and 2.9 the observer can be defined as follows:

$$\hat{\mathbf{x}}(k+1) = \mathbf{A}_0 \hat{\mathbf{x}}(k) + \mathbf{L} [\mathbf{y}(k) - \mathbf{c}\hat{\mathbf{x}}(k)] + \mathbf{b}\mathbf{u}(k). \quad (2.11)$$

It can be shown, that this system is observable.<sup>1</sup>

The controller in this equation is defined as:

$$u(k) = -G_I \sum_{i=0}^k \left[ \mathbf{C}\mathbf{x}(i) - \bar{p}_i^{ref} \right] - \mathbf{G}_x \mathbf{x}(k) - \sum_{j=0}^N G_d(j) \bar{p}_{k+j}^{ref}. \quad (2.12)$$

To calculate the gains in equation 2.21 and 2.12 a performance index must be defined:

$$J = \sum_{j=0}^{\infty} \left\{ Q_e \left[ p(j) - \bar{p}_j^{ref} \right]^2 + \Delta \mathbf{x}^T(j) Q_x \Delta \mathbf{x}(j) + R \Delta u^2(j) \right\}. \quad (2.13)$$

The idea is to minimize the difference between the reference and the actual ZMP, to ensure a smooth trajectory and to lower the reactivity of the system.

The gains can then be calculated as follows:

$$\mathbf{B} = \begin{bmatrix} \mathbf{C}\mathbf{b} \\ \mathbf{b} \end{bmatrix}, \mathbf{I} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \mathbf{F} = \begin{bmatrix} \mathbf{C}\mathbf{A}_0 \\ \mathbf{A}_0 \end{bmatrix}, \mathbf{Q} = \begin{bmatrix} Q_e & 0 \\ 0 & \mathbf{C}^T Q_x \mathbf{C} \end{bmatrix}, \mathbf{A} = [\mathbf{I}, \mathbf{F}] \quad (2.14)$$

$$G_I = [R + \mathbf{B}^T \mathbf{P} \mathbf{B}]^{-1} \mathbf{B}^T \mathbf{P} \mathbf{I}, \quad \mathbf{G}_x = [R + \mathbf{B}^T \mathbf{P} \mathbf{B}]^{-1} \mathbf{B}^T \mathbf{P} \mathbf{F} \quad (2.15)$$

$$G_d(j) = -[R + \mathbf{B}^T \mathbf{P} \mathbf{B}]^{-1} \mathbf{B}^T [\mathbf{A}_c^T]^j \mathbf{P} \mathbf{I}, j = 0, 1, \dots, N. \quad (2.16)$$

The closed-loop matrix  $\mathbf{A}_c$  is defined as

$$\mathbf{A}_c = \mathbf{A} - \mathbf{B} [R + \mathbf{B}^T \mathbf{P} \mathbf{B}]^{-1} \mathbf{B}^T \mathbf{P} \mathbf{A}. \quad (2.17)$$

To calculate  $\mathbf{P}$  the discrete-time matrix Riccati equation must be solved:

$$\mathbf{P} = \mathbf{A}^T \mathbf{P} \mathbf{A} - \mathbf{A}^T \mathbf{P} \mathbf{B} [R + \mathbf{B}^T \mathbf{P} \mathbf{B}]^{-1} \mathbf{B}^T \mathbf{P} \mathbf{A} + \mathbf{Q}. \quad (2.18)$$

Due to fact that  $\mathbf{P}$  is constant this can be done offline using Matlab or Octave.

The gain  $\mathbf{L}$  utilized for the sensor feedback can be calculated in the similar way just by executing the LQR method. This can also easily be done utilizing Matlab or Octave.

The effect of the sensor feedback is depicted in figure ???. As can be seen, if the CoM deviates from the optimal trajectory (either by measuring the CoM error directly or as a result of a measured ZMP disturbance) the controller accelerates the CoM towards the

---

<sup>1</sup>A matlab script calculating the gains can be found in the Coderelease: /Utils/Matlab/writeParamNG.m.

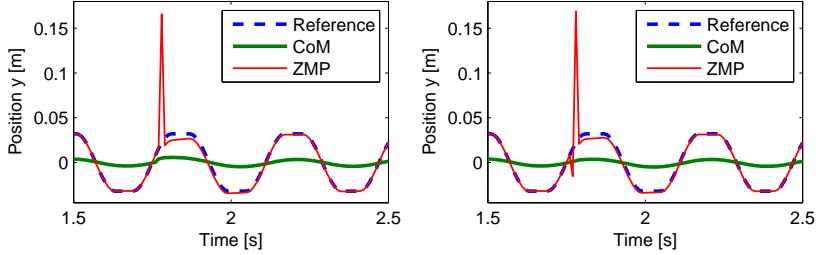


Figure 2.4: Reaction of the controller to different disturbances. On the left an error in the CoM position was measured, on the right a ZMP error.

optimal trajectory. After the acceleration the CoM is decelerated until it follows the optimal trajectory again.

A more detailed presentation of our past approaches and algorithms is presented in [14, 15]. The current approach including a detailed description of the whole algorithm can be found in [13].

### Experimental Sensor Feedback Methods

Besides the sensor feedback realized by the ZMP/IP-Controller there are two more methods implemented. While the ZMP/IP-Controller is responsible for controlling the translation of the center of mass, the orientation is expected to be 0. This assumption is not correct, and should be handled in another way. Therefore a module called GyroTiltController is implemented which realizes a PID controller using the output of the gyroscope directly to control the body roll and tilt. The gains of the controller are chosen such that it has a damping effect on an oscillating body.

### Results and future work

On a real Nao the most important effect of the sensor feedback is damping since the Nao tends to an oscillation during a walk. To show this positive effect, an experiment was conduct where a beam of 5 mm width and height was fixed onto the ground [13]. The robot walked with the right foot over the beam such that it induces an oscillation. Without sensor feedback, the robot fell down in 10 of 10 trials. With sensor feedback in 0 of 10 cases.

The performance of the system was also observable on the RoboCup 2012. However, after many competitions, exhibitions and experiments some Naos of our team reveal highly worn-out gears. It can be observed that in this case the gains for the CoM error feedback must be selected very conservative. Otherwise this feedback can amplify the existing oscillation resulting in different gains for different robots.

Our future work is the integration of reactive stepping. This means that the foot positions are immediately modified in case of a disturbance. The theoretical part of this work can be found in [16].

Reactive stepping is usually for balancing large disturbances. Large disturbances lead to a tipping robot and CoM errors due to this cannot be measured with the FSR. We

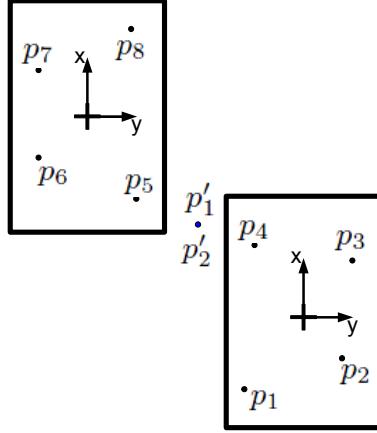


Figure 2.5: The control polygon consists of 4 points per foot with constant coordinates within the respective coordinate frame. In this example a positive x speed is assumed for a better visualization.

therefore utilize the CoM error for the step modification and the ZMP measured by the FSR for damping the oscillation.

Since the system was not advanced enough for application on a RoboCup the corresponding code parts are inactive. Together with the described issues because of worn-out robots we decided not to reactivate CoM measurements for the walk on the RoboCup 2013.

### 2.1.3 ZMP Generation

The ZMP Generation calculates a reference ZMP using the given foot steps. Within the support polygon the position can be freely chosen since every position results in a stable walk. On the x axis the ZMP proceeds with the desired speed. This results in a movement of the center of mass with constant velocity. On the y axis the ZMP is a Bézier curve with a control polygon of 4 points and dimension 1. The coordinate of each point is constant within the coordinate system of the respective foot. Figure 2.5 gives an example. In the right single support phase the control polygon is  $P_r = \{p_1, p_2, p_3, p_4\}$ . The same applies to the other single support phase. In the double support phase the control polygon consists of the points  $p_4, p'_1, p'_2, p_5$ , where  $p'_1$  and  $p'_2$  are the same point in the middle of  $p_4$  and  $p_5$ . This leads to a smooth transition between the single and double support phases. Figure 2.6 shows the resulting reference ZMP along the y axis.

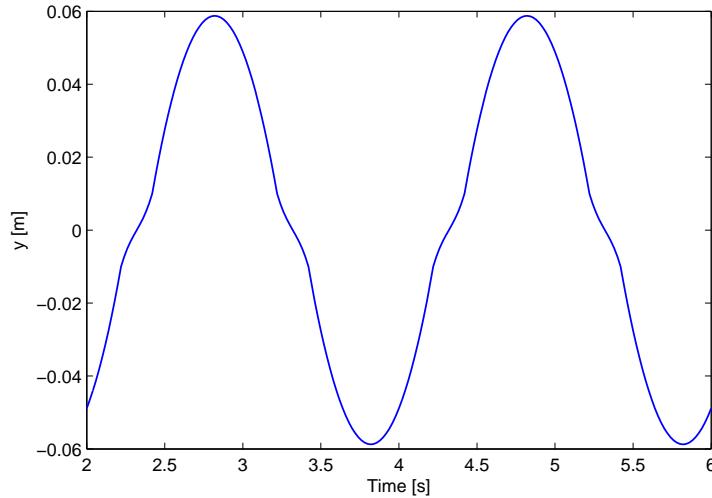


Figure 2.6: Resulting reference ZMP along the y axis.

#### 2.1.4 Swinging Leg Controller

The PatternGenerator sets the foot positions on the floor. They could be imagined like footsteps in the snow. Therefore the footpositions of the swinging leg during a single support phase are missing. They are added by the Swinging Leg Controller which calculates a trajectory from the last point of contact to the next utilizing a B-spline. The control polygon consists of 9 points with 3 dimensions each. The x and y coordinates are set along the line segment between the start and end point. The z coordinate over time can be seen in figure 2.7a. To reduce the influence of the leg inertia the feet are lifted and lowered at the same speed. Figure 2.7b shows the z coordinate over x. The foot reaches its end position along the x axis some time before the single support phase ends. This reduces the error if the foot hits the ground too early.

#### 2.1.5 Reactive Stepping

Even if the stabilizing effect can be shown, it is obvious that not every disturbance can be balanced this way. From observing human beings it can be followed that large disturbances can only be balanced by modifying the desired foot placement. This is also true for walking robots, but the derivation is different. In figure 2.8 at 1.7s a disturbance of the CoM trajectory can be seen. The preview controller balances this by accelerating the CoM to the original trajectory since it is optimal for the given ZMP reference. This acceleration causes a deviation of the actual ZMP trajectory that can be seen in figure 2.8. We can conclude that the reaction to the disturbance can cause the robot to fall. So it is reasonable to say that not a disturbance causes a fall of the robot but an inappropriate reaction to it. An inappropriate reaction is for example to try to regain the originally planned walk in case of disturbances. Human beings modify the foot placement instead. For our ZMP based walk this means that the reference trajectory must be modified

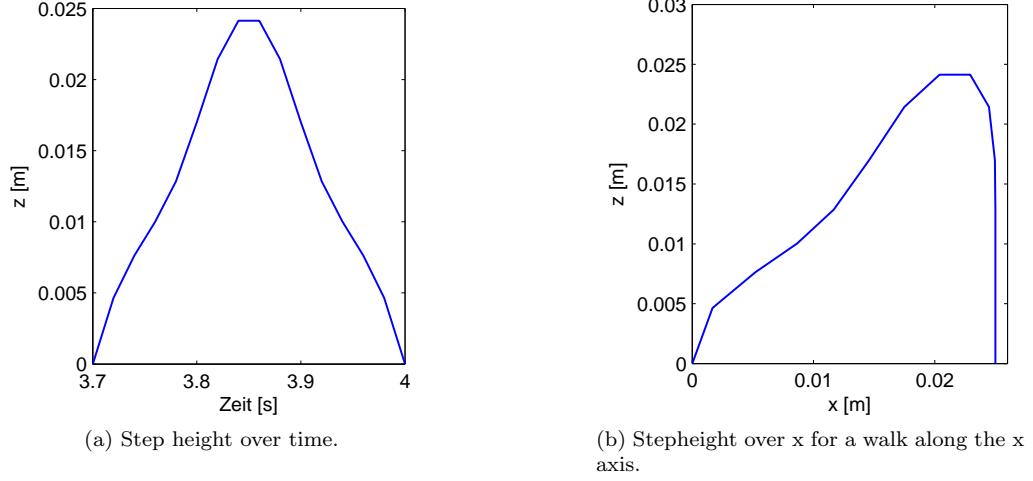


Figure 2.7: Movement of the swinging leg in the global coordinate system.

accordingly. In [16] we present a method to calculate the modification of the ZMP by calculation a simple matrix multiplication:

$$S(t, \mathbf{e}) = \mathbf{G}_{e,t} \mathbf{e}, \quad t \in \{1, \dots, N\}.$$

Input of function  $S$  is the time  $T$  where the step is modified and the measured error  $\mathbf{e}$  given by the ZMP/IP observer. Matrix  $\mathbf{G}_{e,t}$  is constant and can be calculated in advance. The walk with the same disturbances but balanced with step modification can be seen in figure 2.8 on the right. The derivation and further details can be found in [16].

Besides the question about the mathematical realization of the step modification, the implementation of the robot must also consider various topics to realize modification that lead to a stabilization. A tilting robot must be raised after a lunge which requires

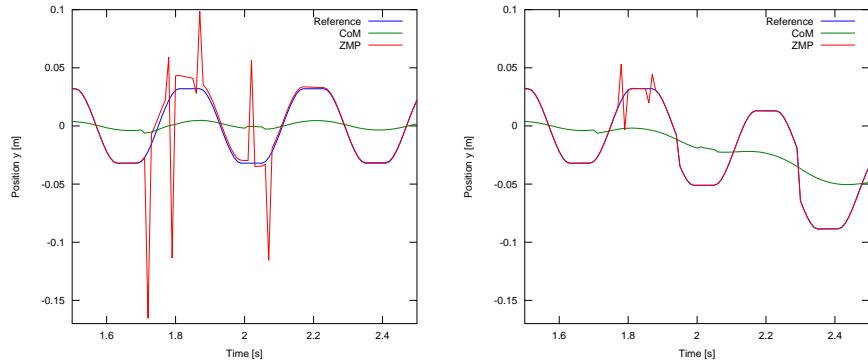


Figure 2.8: Balancing without lunges (on the left) and with lunges (on the right).

high torques. We therefore lower intentionally the CoM linear to the measured body tilt. Additionally the swinging foot is rotated around the stand leg to avoid undesired collisions with the ground.

While it can be proven in various experiments that this modification can stabilize the robot while it would fall down without, we currently investigate the advantage while walking with high speeds. However, the modification is clearly needed to be able to walk on problematic floors, like artificial grass etc.

## 2.2 Path Planning

The Dortmund WalkingEngine accepts as input the desired speed which can be directly set by the behavior. However, usually it is important to avoid obstacles like other robots or goal posts to reach the desired target which makes it difficult to set the speed based on a state machine. Path planing aims at creating in path beginning at the own position to the desired target to simplify this task. Nevertheless the path must be converted to the desired speed as this is the only input accepted by the Dortmund WalkingEngine. This is a critical and often underestimated issue as the Dortmund WalkingEngine is fully omni-directional and walking along a path with a desired orientation has more than one solution. The conversion from the desired path to the desired speed therefore includes also decision that influence the overall time till the target is reached. As a result, we developed a module that focus on this conversion with a high priority on rapid development and efficient debugging.

robot3.PathToSpeed														
ID	Value	Entry Min	Entry Max	Exit Min	Exit Max	Factor X	ID X	Value	Factor Y	ID Y	Value	Factor R	ID R	Value
next_WP_Angle	0.827432	0	100	-20	0	invMax_Foward	300	0	next_WP_Angle	0.8274;-1	Max_R	45.8366		
Next_WP_Dist	402.329	300	10000	-100	500	0	Max_Foward	300	0	Next_WP_Angle	0.8274;-1	Max_R	45.8366	
Next_WP_Angle	0.827432	-100	-50	-50	30	0	Max_Foward	300	0	Next_WP_Angle	0.8274;-1	Max_R	45.8366	
Next_WP_Dist	402.329	2000	10000	-300	200	1	Max_Foward	300	0	Max_Left	140	0	Max_R	45.8366
Next_WP_Dist	402.329	600	2000	-300	500	0.8	Max_Foward	300	0	Next_WP_Dist_Y	5.81	0.4	Next_WP_Angle	0.827432
Next_WP_Dist_Y	5.81	0	10	-10	200	150	One	1	1.5	Next_WP_Dist_Y	5.81	1.5	Next_WP_Angle	0.827432
Next_WP_Angle	0.827432	0	8	-20	20	150	One	1	1.5	Next_WP_Dist_Y	5.81	1.5	Next_WP_Angle	0.827432
Angle_at_Next_W	0.533875	0	8	-20	20	150	One	1	1.5	Next_WP_Dist_Y	5.81	1.5	Next_WP_Angle	0.827432
						1.4	Next_WP_Dist_X	402.287	2.6	Next_WP_Dist_Y	5.81	3	Angle_at_Next_W	0.533875

Figure 2.9: Window with rules during normal behaviour.

The conversion is based on rules as depicted in Fig. 2.9. A rule has the following properties:

- A number of entry conditions that are all must be true to enter into this rule,
- a number of exit conditions that define a hysteresis before the rule can be leaved,
- for all dimensions (x, y and rotation) a constant factor with a multiplier that defines the speed that is executed as long as this rule is active.

All conditions are define by a minimum value and a maximum value that are compared to a variable. If all variables are within the desired range, the rule is selected. To exit a rule, all variables must be outside the range that is define by the exit minimum and maximum value. Many variables are available:

- Euclidean distance to next way point,

- angle to next way point,
- maximum speed in x and y direction and maximum rotational speed,
- angle to destination,
- distance to destination,
- angle at next way point (relative to current angle),
- distances to next way point on single axes (x and y),
- distance to next obstacle,
- angle to ball,
- distance to ball,
- and the constant 1.

Some of the values are only useful for definition of the resulting speed. E.g. the maximum speed can be multiplied with an arbitrary value to select a constant speed.

All rules are prioritized. If a rule with higher priority (higher position in the hierarchy as depicted in Fig. 2.9) and the entry conditions are met, the exit conditions of the previously selected rule with a lower priority are ignored and the higher rule is selected immediately.

For fast debugging we developed the GUI as shown in Fig. 2.9. The colors clarify which rules are active and why (green: entry conditions met, red: exit conditions not met). Also the current values are printed. Additionally the rules can be changed online and reloaded during execution in simulation as well as on a physical robot.

## 2.3 Kicking

Kicking motions have different objectives. First the ball must be kicked as precise as possible with an adjustable power. On the other hand, the kicking motion should be as short as possible to avoid unnecessary delays during gameplay. As a consequence we have different kicking motions in use. One can be used for precise and adjustable kicks while the other is executed within a normal walk cycle.

### 2.3.1 Precise Kick Motion

In every SPL soccer match, whether it is for a pass or a shot on the goal, a precise kick that can be adjusted at the last possible moment, to cover up an imprecise ball model or a moving ball in an one on one situation, has become a necessity. Since covering all possible kick angles and strengths with Special Actions (see section 2.4) is simply not possible, a dynamic kick was needed.

Our current implementation uses the inverse kinematic from our walking engine to move the kicking foot towards the ball. With a fast kick execution time, the stability of the robot can hardly be ensured with any kind of reactive controller. Therefore we concentrated on a implementation that works without a controller and still is able to

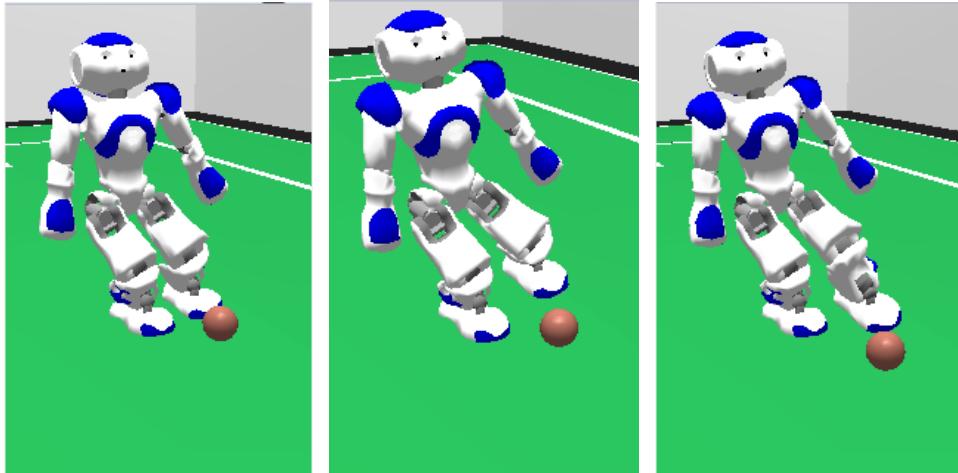
execute a stable kick with all desired angles and strengths. The kick motion needs only one input, the kick target: a field position relative to the robot.

The main focus in the development for our kick was its stability while still being able to kick hard. Therefore the kick process is divided into several phases (see figure 2.10):

- lean - shift the robots weight to the standing foot
- prepare - move the foot to a point near the ball
- execute - a fast straight motion towards the ball
- prepare back - back to the position after lean
- lean back - back to the original robots position (standing)

The separation into these phases allow a parameterization of the kick which can be adjusted and optimized easily. After the leaning process, our kick takes the relative ball coordinates from our ball model at the beginning of the prepare phase, moves the foot to a point in front of the ball so that the execute phase can kick the ball in one straight move of the foot in the desired direction. For this, the kicking foot origin is placed in a fixed distance to the ball on the the circle that this specific distance creates. Due to the different trajectories resulting from a sidekick compared to a frontkick, seperate leaning angles for those motions are used and, since the foot of the Nao robot is not completely smooth or round, the kicking angle is not entirely free. Currently, the kicking angle is limited from 0 to 25 degrees for a frontkick and from 65 to 90 degrees for a sidekick, which is quite enough for almost any real game situation.

After the kicking foot has been brought into position, the kicking motion itself is done before the robot begins a reverse leaning process to again ensure its stability.



(a) Robot after the lean phase. (b) Robot after the prepare phase. (c) Robot after the execute phase.

Figure 2.10: The Robot in the three main phases of the kick, triggered from SimRobot.

### 2.3.2 Instant Kick

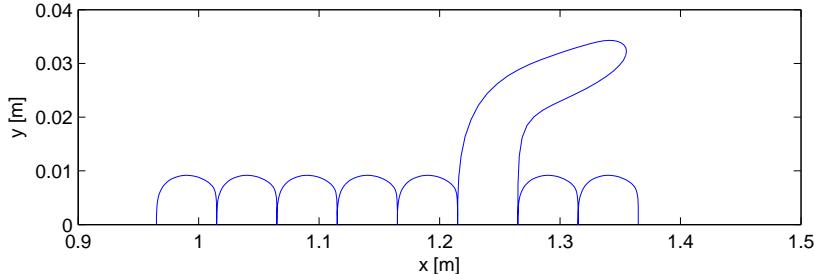


Figure 2.11: Visualization of the foot trajectory kicking a ball.

While all executed kicks should be as precise as possible, it is sometimes important to execute the kick as fast as possible. An illustrative situation is a near opponent challenging for the ball. Kicking precisely is no more necessary if the opponent reaches and kicks the ball before the own player has executed the kick. For this reason another kicking motion exists, the "Instant Kick". This is not a motion separated from the walking engine like the Precise Kick or the older Special Action Kicks (see sec 2.4). It is executed during the phase where the Swing Leg Controller (see sec. 2.1.4) moves the swing leg from one position on the floor to the next. Instead of using a movement depicted in fig. 2.7 a curve is calculated that lifts off the foot, executes the kick and sets the foot on the target position on the floor where it would have set if no kick is executed, see fig. 2.11. The walk is therefore untouched and continues without a difference to a walk without kicking, besides the resulting instabilities. Since also all sensor feedback algorithms are still activated during kicking the motion is stabilized by the well proven walking engine.

To enable kicks straight forward and additionally kicks to the "inner"<sup>2</sup> side two separated set of parameters are utilized, one for kicking straight forward and one for kicking with the maximum angle. The actually used set of parameters is a linear interpolation of both depending on the desired angle.

A kick straight forward or with a low angle is calculated using the actual ball position. Faulty measurements of the ball position can lead to large leg movements causing a fall down especially when executing a kick to a side. If an angle larger than half of the maximum is desired, the ball is therefore assumed to be at a predefined position.

The kick is divided into three phases. During the first, the foot is moved to a start position, e.g. 8 cm in front of the ball. In the second phase, the foot is moved behind the ball and in the last phase, the foot is placed on the position for the next step. This three phases therefore replaces the normal movement of the foot during a single support phase. The distance in front of the ball, behind the ball and the duration of the phases can be configured as well as the fixed ball position for kicks to a side.

### 2.3.3 Preview Kick

In previous years we applied a kick that is executed within the swing phase during the walking motion without changes. It is a fast and robust kick but due to the requirement

---

<sup>2</sup>When kicking with left foot this is a kick to the left and vice versa.

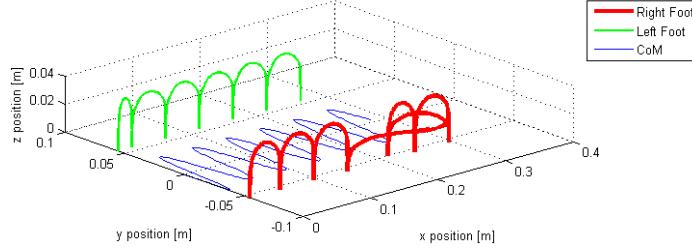


Figure 2.12: Feet positions in world coordinate system of a kick with the right foot during a walk with  $\dot{x} = 5\text{cm/sec}$ . The ball lays at  $y = -5.5\text{cm}$  and is kicked to the left side. The kick is executed within the walk without the need to stand before or after the kick.

of the walking engine to provide a preview of the ZMP, the position of the feet cannot be altered immediately and the kick must start and end at the preplanned foot positions, see Fig. ???. Thus the foot must be moved to the ball and back to the desired position, resulting in a large distance for a short time that reduces the precision of the executed foot motion.

We therefore enhanced the structure of the kick that it is planned approximately half a second before it is executed. Thus we are able to reduce the length by placing the foot on the position of the ball. This way the kick is executed with only approximately half of the distance and a significantly improved precision.

## 2.4 Special Actions

All movements except walking and kicking are executed by playback of predefined motions called *Special Actions*. These movements consist of certain robot postures called key frames and transition times between these. Using these transition times the movement between the key frames is executed as a synchronous point-to-point movement in the joint space. Such movements can be designed easily by concatenating recorded key frames.

## 2.5 Handling the Outdoor Competition

The focus of this section is the presentation of several details relevant for various aspects of a physical robot, e.g. gravity compensation of undesired elasticities etc. for walking on flat carpets and artificial lawn. This section quotes the paper "Playing Soccer Outdoors with Humanoid Robots" which was published at IEE-Humanoids 2016 [?].

### 2.5.1 Controller

Based on the 3D linear inverted pendulum model (3D-LIPM) [?] where the robot is modeled as a single center of mass (CoM), we utilize a preview controller as proposed by Kajita et al. [11] in a modified version.

The 3D-LIPM relates the movement of a single mass along the x-axis  $c_x$  to a Zero Moment Point (ZMP)  $p$ , where no torques around the lateral and transversal axis is

exerted on the robot:

$$p = c_x - \frac{c_z}{g} \ddot{c}_x. \quad (2.19)$$

For a linear system the height over ground  $c_z$  is constant. The gravity is denoted as  $g$ . Given this relation, a discrete-time system can be formulated describing the physical behavior:

$$\mathbf{x}_{k+1} = \mathbf{A}_0 \mathbf{x}_k + \mathbf{b} u_k, \quad (2.20)$$

where  $\mathbf{x}_k$  is the current state of the system, consisting of the CoM position, CoM speed and the ZMP.  $\mathbf{A}_0$  denotes the matrix for predicting the state in the next time frame and  $\mathbf{b}$  a vector incorporating the preview controller output  $u_k$  into the system. Details about the controller definition can be found in [?].

### 2.5.2 Gravity and Inertia Compensation

Our sensor feedback methods are able to react to disturbances caused by issues of the physical robot. However, some can be foreseen and should therefore be handled before they occur.

In [?], we show that the robot has unintended flexibilities and elasticities in the gears, links etc. Hence, we proposed [?] the Flexible Linear Inverted Pendulum Model that extends the known 3D-LIPM about a spring-damper-mass system. In the following paragraphs, we describe the gravity and inertia compensation in more detail:

#### General Considerations

As explained in 2.5.1, we utilized the traditional 3D-LIPM on RoboCup 2016 and extended the algorithm by various methods to cope with gravity and inertia issues related to the elasticities.

There are parameters of our walking engine that can be adjusted to lower the impact of flexibilities: The step duration has an effect on the swinging amplitude of the body in y direction (and therefore the lateral acceleration in each step). Thus, it should be chosen to reduce that oscillation by preferring higher step frequencies. Moreover, a larger step height can have a positive impact as the touch-down of the foot on the ground can induce a force/torque supporting the lateral oscillation.

The lateral oscillation is desired for a walk in contrast to an oscillation into the x direction with a growing amplitude. The sensor feedback methods presented in Sec. 2.5.3 are intended to reduce these undesired oscillations. Additionally, the configurable hardness of the joints has a huge impact as lower hardness dampens these oscillations. Furthermore, for the swinging foot we lift up the tiptoe that depends on the current height over ground to avoid undesired collisions with the ground.

#### Offsets

Besides these considerations, the algorithm contains possibilities for manual enhancements. For both x and y axis translation and rotational offsets are applicable:

- Translational offset for x and y axis.

- Fixed, i.e. constant over time.
- Dynamic, i.e. depending on current speed.
- Rotational offsets.
  - Rotation of body around y axis.
    - \* Fixed, i.e. constant over time.
    - \* Dynamic, i.e. depending on current speed.
  - Angle offset added to desired joint angles.
    - \* Fixed, i.e. always applied.
    - \* Dynamic, i.e. applied only if corresponding foot is currently the supporting foot.

Given the foot position in robot coordinate system, a translational offset is added to this position. Thus, it can be avoided to fall more frequently to the back or front.

A rotational offset of the body is applied to ensure a minimum torque on the gears to reduce changing torques that can cause oscillation. The intention of the rotational offset at the joints is gravity compensation in combination with the undesired elasticities.

### 2.5.3 Sensor Feedback

The motivation to modify the walk by utilizing sensor feedback is evident as not every event can be foreseen. In our approach we combine three different methods, each specialized for a specific domain, that all together utilize every available sensor sources. While we also proposed a method for stabilizing strong pushes [?] the main intention here is to balance smaller disturbances like oscillations, slightly uneven grounds and issues of the physical robot.

#### Observer

An observer can be utilized if the state  $\mathbf{x}_k$  cannot be completely measured. In our case we can measure (denoted by  $\mathbf{y}_y$ ) the position of the CoM by using a forward kinematic and the ZMP utilizing the Foot Resistive Sensors (FSR). The observer equation is comparable to Eq. 2.20:

$$\hat{\mathbf{x}}(k+1) = \mathbf{A}_0 \hat{\mathbf{x}}_k + \mathbf{L} [\mathbf{y}_y - \mathbf{C}_m \hat{\mathbf{x}}_k] + \mathbf{b} \hat{u}_k. \quad (2.21)$$

The hat denotes that the corresponding value is now based on the estimation of the observer. Matrix  $\mathbf{C}_m$  selects the measurable part of the state:

$$\mathbf{C}_m = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.22)$$

Details of the method can be found in [?].

## Orientation Controller

While the observer is based on the linear CoM movement within a plane, a physical robot will tilt and thus leaving this restriction. As rotations are not covered by the above described system, a further control loop is implemented based on the gyroscope measurement  $\dot{\varphi}_k$ . First, the measured value is filtered (using temporary values  $\dot{\varphi}'_k$  and  $\dot{\varphi}''_k$ ):

$$\dot{\varphi}'_k = (1 - \alpha_o) \dot{\varphi}'_{k-1} + \alpha_o \dot{\varphi}_k, \quad (2.23)$$

$$\dot{\varphi}''_k = (1 - \beta_o) \dot{\varphi}''_k + \beta_o (\dot{\varphi}_k - \dot{\varphi}'_k), \quad (2.24)$$

with a filter gain  $\alpha_o$  and  $\beta_o$ . Next, the desired orientation  $\varphi_k^d$  of the body is calculated by:

$$\varphi_k^d = k_d \dot{\varphi}''_k. \quad (2.25)$$

Thus, it could be seen as a PD controller based on the filtered orientation with gains  $k_p = 0$  and an arbitrary  $k_d$ . We apply it for the x axis only.

## Linear Acceleration

For the observer and orientation controller the acceleration sensor is not utilized. We therefore introduce an additional sensor controller to have all available sensor sources incorporated.

As explained in Sec. 2.5.2, we add a fixed offset to the desired CoM trajectory determined by the controller to optimize the body x position relative to the feet. This controller modifies this value dynamically based on the measured acceleration:

$$x_k^{f,d} = x_k^f - \alpha_a |a_t| \underbrace{\text{sgn}(\bar{a}) \min(|\bar{a}|, 0.03)}_{\in [-0.03, \dots, 0.03]}, \quad (2.26)$$

where  $\alpha_a$  is an arbitrarily chosen gain,  $x_k^{f,d}$  the offset added to the feet positions (in robot coordinate system),  $\text{sgn}()$  the sign function,  $\min()$  the minimum of the given arguments,  $a_t$  the measured acceleration at time  $t$  and  $\bar{a}$  the mean value of the last five measured accelerations. Basically, Eq. 2.26 subtracts from the fixed offset the median that is limited to  $[-0.03, \dots, 0.03]$  and multiplied with a gain.

### 2.5.4 Acceleration

The controller (Sec. 2.5.1) minimizes but not limits the difference between the desired and actual ZMP. Thus, for high accelerations the actual ZMP may leave the support polygon. We therefore limit the allowed acceleration to a maximum speed change  $a_{max}$  within a defined time  $T$ : given a speed  $s_t$  at time  $t$ , the speed at  $s_{t+T}$  must fulfill  $s_{t+T} - \min(s_t, \dots, s_{t+T-1}) < a_{max}$ . This also means that deceleration is unlimited.

# Chapter 3

## Cognition

From the variety of Nao's sensors only microphones, the camera and sonar sensors can potentially be used to gain knowledge about the robot's surroundings. The microphones haven't been used so far and are not expected to give any advantage. The sonar sensors provide distance information of the free space in front of the robot and can be used for obstacle detection. However, the usage of these sensors depends on maintaining a strictly vertical torso or at least tracking its tilt precisely since otherwise the ground might give false positives due to the wide conic spreading of the sound waves. Additionally, for certain fast walk types the swinging arms were observed to generate such false positives, too, and the sonar sensor hardware in general is currently unreliable and often does not recover from failure once the robot has fallen down. Thus for exteroceptive perception the robot greatly relies on its camera mounted in the head.

The following chapter describes the information flow in the cognition process starting from image processing and its sub-tasks in section 3.1. Special focus is given to new developments since 2011, meaning an image processing that does not rely on color tables. Also, the localization module based on a multiple-hypotheses unscented Kalman filter (see section 3.4).

### 3.1 Image Processing

In the past years the Nao Devils Team was using a color table based image processor which was based on the Microsoft Hellhounds' development of 2007 [17]. This image processing however, took at least several hours to calibrate and was very susceptible to lighting condition changes and color changes, for example when changing between two different fields. To address these problems, the currently used image processing was written with the idea to minimize this configuration process. Furthermore, due to the increased field size, the used resolution for each camera was increased from 320x240 for both cameras to 1280x960 for the upper and 640x480 for the lower camera.

The image processing in use still uses the color information from the color coded field, but does not need a specific calibration to get a good detection rate. To achieve this, the field color (green) is newly calculated every frame using a weighted color histogram based on pixel samples on the image. The key idea is to use as much a-priori information as possible to remove the need of color tables, reduce the scanning effort and minimize the

needed subsequent calculations. Since the limited cpu power does not allow for a scan of two whole images, we process only a small fraction of all pixels, scanning the images along fixed vertical and horizontal scan lines.

These scan lines search for changes within the y-channel to detect the white lines (if surrounded by the detected field color), and also detect possible ball locations based on the color channel changes. To reduce computation time, the detection of goals takes place only in the upper part of the images on the horizontal scan lines, also using the information about the field border detection.

All relevant objects and features are extracted with high accuracy and detection rates. Since the Nao SPL is the first RoboCup league (neglecting the simulation leagues and the Small Size League with global vision) that plays on a symmetric field, detecting features on the field itself becomes more important. The detection of those is described in more detail in the following section.

The current implementation allows us to play in natural lighting conditions and as shown in the open challenge for 2014, using auto camera settings, our robots continue to play and score even with huge lighting condition changes in games.

Since the image processor in use was implemented to adapt to different resolutions, the changes noted in the beginning did only result in a slightly slower runtime for both images and we were still able to process both images at 30 fps each.

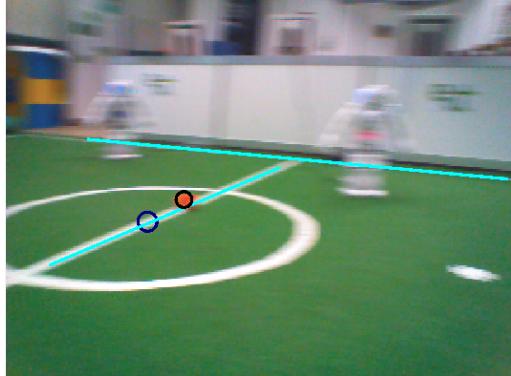
## 3.2 Line and goal detection

The line detection is using a similar approach to the last years. Based on line points detected on the scan lines, these points are connected to chains to determine possible lines and center circles. No gradient calculations are used since these calculations are susceptible to noise (especially on far away lines that have a width of 1 or 2 pixels) and also runtime was saved. Furthermore, especially using auto camera settings, the image is blurred in game on rapid head or robot movements (see figure 3.1), thus making gradient calculations even less accurate. The connected points are verified as lines with extra verification scans. Every detected line is then enhanced by scanning along the lines direction at its ends to maximize the lines length.

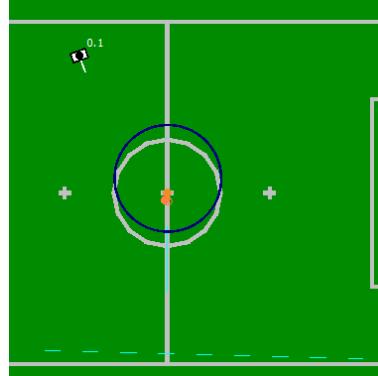
Since the field size was doubled in 2014, the used scan lines proved to be too few to detect the ball reliably in a distance of more than 5 meters. To address this problem, several additional scan lines that scan specifically for the ball were added if the ball was not be found using the standard image processing. For the position of these scan lines, the ball model, including its prediction is taken into account. If the ball is still not found, since the ball is only expected to be lost at a certain distance, only the part of the upper image near the horizon then is scanned with randomly placed scan lines. This significantly increased the ball detection rate and helped to detect the ball in a distance of up to 9 meters.

## 3.3 White goal detection

In comparison to 2014, the obvious challenge was to detect white goals. To this end, the current concept was enhanced to not only use yellow features but edges in general as the input for goal perception. These edges are then connected to goal post side lines and



(a) Camera Image.



(b) Field view.

Figure 3.1: Detection results on a blurred image.



(a) White goal detection at the  
German Open 2015. (b) Goal post base point adjusted  
with line detection.

Figure 3.2: White goal detection in different conditions.

scans are performed to verify width, height and unity of color on the goal post as in the previous years. As an additional measure to identify the position of the goal post on the field, detected field lines are used since there might not be a visible edge between field line and the bottom of the goal post.

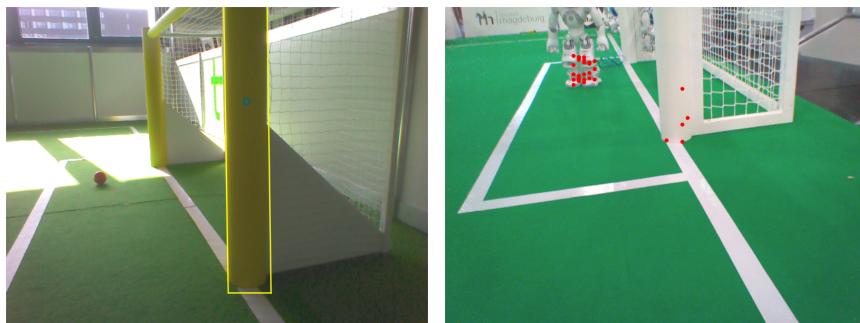
The white goals presented a problem for our robot detection since depending on the structure of the goal it contains a lot of white resulting in false positive percepts in the goal area. This problem was solved by adding a simple 1D-Feature scan on the legs of the possible robot. On a real robot a lot more of these features are present making it easy to distinguish between a real robot and a false positive percept.

Figure 3.2 shows some examples of the white goal detection and an example of found 1D-Features on a robot. The current implementation allows to detect all goal posts independent on their color, as long as the edges are found.



(a) Correctly detected goal posts with side indication.  
 (b) White goal detection error due to similar background.

Figure 3.3: White goal detection and robot detection in different conditions - no calibration was done throughout presented images.



(a) Detection of a yellow goal post.  
 (b) 1D-Features on robot and goal post.

Figure 3.4: White goal detection and robot detection results in different conditions - no calibration was done throughout presented images.

### 3.4 Localization

One main focus of research is on Bayesian filters, where several enhancements for real time vision-based Monte Carlo localization systems [18] have been presented, and the approach based on the detection of field features without using artificial landmarks has won the “almostSLAM” Technical Challenge at RoboCup 2005 [6]. The methods used up until RoboCup 2009 have all been based on those particle filters developed between 2005 and 2007. For RoboCup 2010 however a different approach has been developed and used in the competitions of 2010, and also in 2011 without any modifications [19].

Inspired by [20] the basic idea was to combine the smoothness and performance of Kalman filtering with a multi-hypothesis system. The latter is necessary to allow recovery from huge errors due to extended periods of integrated odometry errors without correcting through observations, or rapid unexpected position changes due to contact with other robots or “teleportation” by human intervention. All of those issues occur in robot soccer games and are amplified by the huge odometry errors inherent in fast biped walking.

A classic Kalman filter applied to localization in a SPL scenario can only be used for position tracking, and only as long as the estimate does not deviate too much from the true positions. Most perceptions on a SPL field are ambiguous, so the sensor update will only be done with the most likely data association. At the same time, the perception of field features like field line crossings is often uncertain so that L- and T-crossings can not be distinguished. In such cases wrong associations tend to drive the estimation further away from the true position. Recovery from such situations is only possible when assigning huge weights (e.g. small observation covariances) to unique perceptions like observing both goal posts in the same frame which then decreases the robustness against false perceptions originating from the audience around the field.

This problem is addressed in [20] with a sum-of-Gaussians Kalman filter, where each Gaussian is split into several new ones representing the results of different association choices. Applying all possible data associations to every hypothesis generates exponential growth which needs cutting back shortly after by applying pruning heuristics and fusing similar Gaussians to prevent an explosion of computational complexity. Thus lots of processing time is wasted on creating and destroying new hypothesis which are either unlikely or very similar to the ones that already exist.

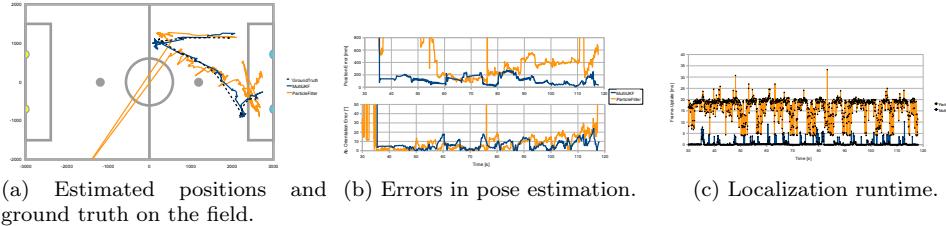


Figure 3.5: Localization of Multiple-Hypotheses UKF compared to previous particle filter solution (which was used in RoboCup 2009). Both are running in parallel on the Nao using the same perception as input. Ground truth is provided by a camera mounted above the field.

A different approach is implemented in the module *MultiUKFSelfLocator*. Only few new hypotheses are generated periodically at positions with high probability based only on

recent sensor information. Those hypotheses are only updated using data that lies inside a certain expectation threshold. Non-linearity in the sensor model is addressed using the unscented transformation technique. Several other approximations and simplifications result in a localization method that is an order of magnitude faster than the previously used particle filter while providing superior localization quality and increased robustness to false positive perceptions (see figure 3.5). A more detailed presentation of the approach and its stochastic soundness is given in [19].

### 3.5 Distributed World Modeling

Current work includes robust cooperative world modeling and localization using concepts based on multi robot SLAM [21], as well as using probabilistic physics simulation and estimation to improve the robot's state estimation and odometry information [22]. Keeping track of the robot's dynamic environment opens the possibility for tactical behavior decisions beyond simple reactive behaviors which are currently in use.

The algorithm described in [21] estimates the robot's location and the surrounding dynamic objects simultaneously. In this joint modeling of the robot's state a particle filter estimates the robot's pose. Clusters of particles are combined into super-particles which map the dynamic environment using a number of Kalman filters. This represents an approximation of FastSLAM and both decreases the integration of odometry error compared to robot-centric local modeling (see figure 3.6) and allows resolving multi-modal localization belief states using shared information. The approach even preserves its SLAM functionality and is able to maintain a robot's localization based on mapped dynamic obstacles only. A detailed presentation of the approach is presented in [21].

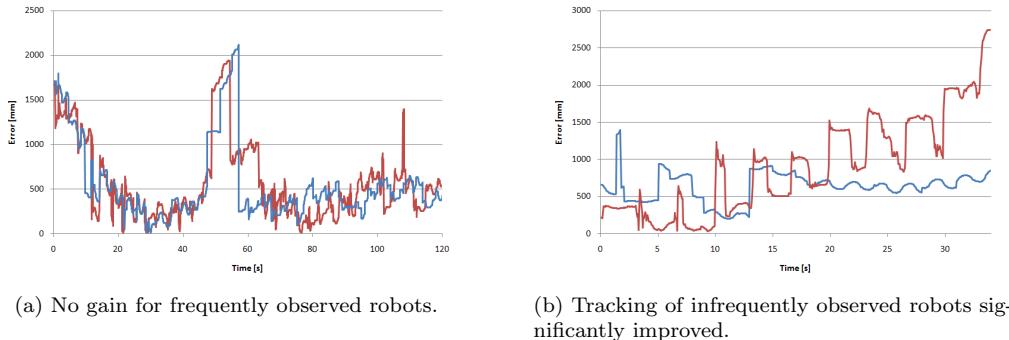


Figure 3.6: Advantage by unified (blue) compared to separate robot-centric modeling (red).

By specifically addressing the heterogeneity of the perceived information and the need to synchronize the estimation between the team of robots the task's complexity can be reduced to be in the range of applicability on limited embedded platforms. This module's average runtime on the Nao in the configuration used on the RoboCup 2010 would have been slightly above 20 ms which would not have allowed to keep a frame rate of 15 Hz or even 30 Hz. Especially the switch to a motion frame rate of 100 Hz made its application

impossible. Additionally the system is based on the outdated particle filter localization of previous years.

For 2011 parts of this distributed world modeling approach have been integrated with the new localization system described in section 3.4. Since the UKF localization concept does not allow the transformation of a combined estimator according to the Rao-Blackwell theorem, the SLAM aspect can not be transferred in this case directly. In the 2011 code used in the competitions, the SLAM aspects were not fully implemented. The ball and robot modeling is still done in a cooperative way using the approximations described in [21], but there has been no feedback into the localization and only a single maximum-likelihood hypothesis is maintained at each time.

In the code release accompanying this report there is the possibility to choose between local models, which are the result of the local aggregation of percepts (percept-buffering, but without periodically flushing the results into the global model, and the global model itself, which is a fusion of all distributed information. Due to the lack of feedback into a robot's own localization there are situations where the local model is still to be preferred. Precisely approaching close balls for example requires accurate robot relative information instead of the precise global position of the ball on the field. In the global model, the ball is more precise in global coordinates, but moderate errors in a robot's localization would have a major impact on the relative positioning, as long as the robot's pose is not corrected by the distributed information, too.

## 3.6 Handling the Outdoor Competition

This section quotes the paper "Playing Soccer Outdoors with Humanoid Robots" which was published at IEE-Humanoids 2016 [?].

### 3.6.1 Handling Varying Lighting Conditions

The outdoor competition at RoboCup 2016 presented different and new challenges for the competing teams (see Figure 3.7). Since the fields were inside a glass hall they had to cope with different scenarios within one game. These scenarios include

- looking into sunlight,
- shadows from the ceiling steel beams, spectators and robots or
- fast changes in illumination due to head movement or clouds.

These issues are handled by looking for local edges and gradients instead of relying on absolute predefined values while scanning the image. Furthermore, every object detection algorithm (line, robot and ball) has been altered to rely less on field color detection and more on shape verification.

Since the presentation of our vision system at RoboCup 2015, a more efficient implementation reduced the runtime without the ball detection from 18 ms to less than 10 ms. This enabled us to use a more sophisticated ball detection as described below. Since the goal detection was less reliable than the line detection and did not improve the localization task, it was removed to allow even more runtime for the ball detection.



Figure 3.7: Picture of the venue at RoboCup Leipzig 2016.

### 3.6.2 Black-White Ball Detection

In the previous years our vision system did use less than 0.5 ms of runtime on the NAO robot for ball detection since the orange color was unique on the soccer field and very few image regions had to be scanned. Since the colors on the current ball are present in many parts of the image, possible ball spots generated from our vision system are filtered before a detailed verification with shape verification is run like in the previous years. This is done by a four-direction scan originating from every ball spot that removes objects of wrong size and impossible shape for a ball.

The main challenge introduced by the new ball however lies in its partial similarity with the other objects on the field such as robot feet and goal post. Without depth information, to distinguish between those objects and to still detect a ball lying in front of them, an additional feature-check was added to the ball detection. With this feature-check we are searching for the dark areas on the ball and verify them by size and distribution. The feature size is verified by using the detected ball size as a comparison while the distribution is verified through number and minimum distance between the features. Since all of this can still apply to parts of the robot, especially when dealing with blurry images, a color distribution check was introduced. This simple check discards the possible ball if one part of the scanned region has too much white or black in it. To keep the system calibration free and cope with balls in very bright or shadowy areas, these colors are determined online while verifying the shape of the ball.

Another problem in the detection of a mainly white ball is the possible overlap in the image with other white objects on the field such as lines or robots. To cope with this difficulty we memorize the overlapping areas when using star-like shape verification scan lines from the center of the ball. Size and position of these areas determine if the scanned

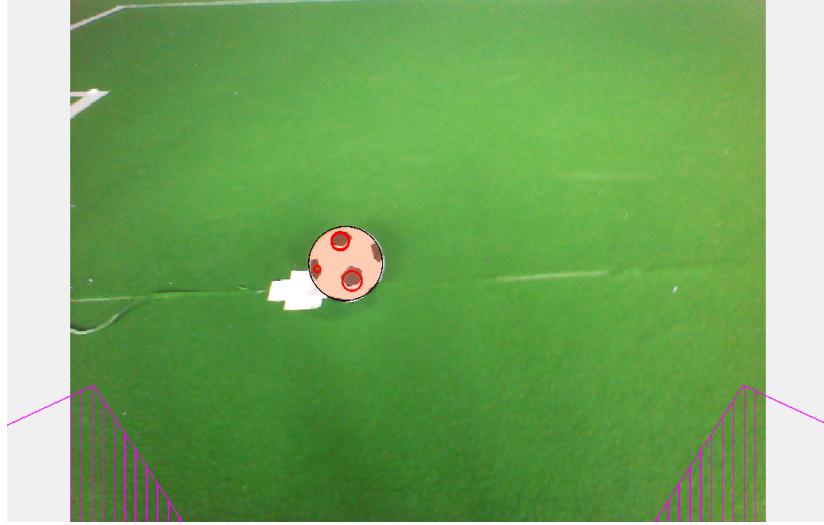


Figure 3.8: Detected features on a ball overlapping with the penalty cross.

object lies on a line, in front of another robot or can not be a ball. The feature scan is performed only on a ball with an overlap either with another object (see Fig. 3.8) or the image border to further reduce the runtime.

### 3.6.3 Using Modified Drivers

This years challenge was to optimize the image recognition to the changing conditions of light. Therefore it is necessary to choose the camera parameters wisely (exposure time and gain). Due to the fact that on the one hand the NAO's camera autoexposure (Aptina MT9M114) does not operate sufficiently and on the other hand the autoexposure is optimized on static images, improvement is required. To save computing power, the camera should be able to choose the parameters by itself. Because the camera's driver does not support this, we took a look into the camera's manual. This showed us, for example, that there is a possibility to weight certain parts of the image differently. The original manual does not give further information to set the weights, therefore we successfully adapted the information from a similar camera (ON Semiconductor ASX340AT).

This new gained knowledge makes it possible to set the following parameters:

1. Minimum and maximum gain
2. Weight of every 5x5 area
3. Target luminance

The optimization lead to the fact that we can set a minimal gain to get minimal exposure times and resulting in less blurred images. Through the weight it is possible for the autoexposure to ignore unimportant areas. Those are for example the lower part of the image (often this is the NAO's body) and the upper part which often entails parts outside of the field. Furthermore the target luminance allows it to adjust the total overall



Figure 3.9: Detected ball against the light.

brightness to our image processing algorithms. In conclusion this new gained awareness allows us to find new indoor and outdoor settings with which we can play.

The following Figures 3.10 and 3.11 compare the difference between the standard settings and a minimum gain of 128 which was set by the driver.

The driver, together with the Nao Kernel, is publicly available at <https://github.com/NaoDevils/NaoKernel>



Figure 3.10: Detected features on a ball overlapping with the penalty cross.



Figure 3.11: Detected ball against the light.

# Chapter 4

## Behavior

*Nao Devils* as well as previous Dortmund teams implemented behavior mostly by utilizing state machines such as XABSL (*Extensible Agent Behavior Specification Language*). XABSL was developed in its original form in 2004 using XML syntax [23] in Darmstadt and Berlin and adapted in 2005 to its current C-like syntax and a new ruby-based compiler by the *Microsoft Hellhounds*. Since 2013, we use CABSL which was developed by team B-Human, and implements XABSL as C++-Macros allowing for easy access to all data structures (i.e. representations) required for decision making.

To this end, behavior is specified by option graphs. Beginning from the root option, subsequent options are activated similar to a decision tree until reaching a leaf, i.e. an option representing a basic skill like “walk” or “execute\_special\_action” which are parameterized by the calling option. Each option contains a state machine to compute the activation decision based on a number of values stored in the representations (see section 3.1).

While it is possible to design complex behavior using CABSL, several tasks may prove difficult or impossible to specify using CABSL alone, e.g. robust and efficient path planning including obstacle avoidance and also any strategic team behavior.

There were no significant change in the behaviour except bugfixing during the last years. Consequently, a restructuring and further feature improvement of the code will be done during the course of the year. This affects mainly the head motions.

The following subchapters reflect our approach to team behavior 4.1, and path planning.

### 4.1 Teamplay

#### Team Strategy

For 2013, there have been significant rule changes in the SPL: The size of the field has been doubled and the number of robots increased to five per team. Hence, major parts of the existing behavior code had to be adapted, especially dynamic positioning. This year’s behavior is similar to the one used in 2013 and defines the roles of the field players by their location and takes the current positions of each team robot into account.

The assignment of those positions of the field players is divided into two parts:

- The robot that has the easiest (i.e. preferably no obstacles and good orientation for a kick) and shortest way to the ball attempts to go there. Once it has reached the ball, it may opt either for shooting to the opponent's goal or for passing the ball to a team mate. If its way is blocked by an opposing robot and no team mate is in position to receive a pass, the robot plays the ball around the opponent.
- The other field players either support the robot which is in charge of the ball, or defend the own goal. For that purpose, positions are calculated dynamically depending on the game situation, including own and opposing player models as well as the ball model. Each player is assigned to one of those locations. The decision which robot is going to a particular spot is taken based on a shortest path heuristic. This strategy, as seen in 4.1 also minimizes the collisions between team mates.

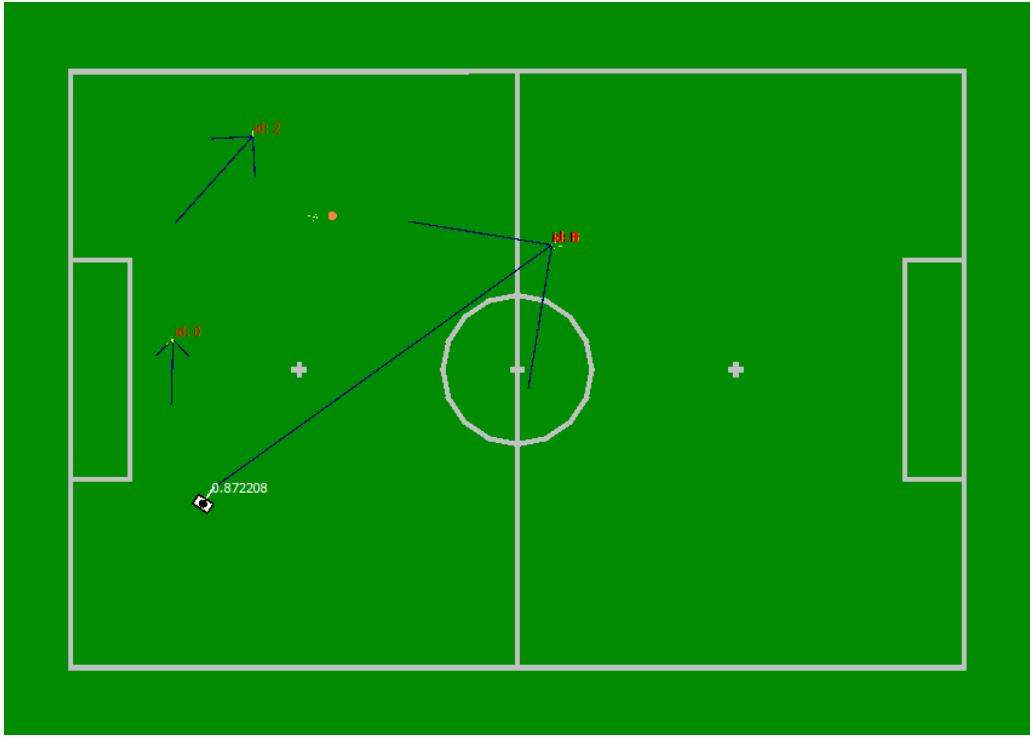


Figure 4.1: Example for the assignment of players to positions.

The task of assigning positions to players underlines the importance of having sophisticated distance metrics. To this end, we abstract from the Euclidian, and use a heuristic which includes penalties for obstacles, and adverse rotations (e.g. the robot is rotated opposite to a target position).

Apart from that, our behavior features:

- Five distinct roles: The *striker* always attempts to reach the ball. The *defender* is always acting in the own half. The *backup-striker* supports the striker in its vicinity. The *supporter* moves to distinct positions in the offense, or defense, depending on

the ball position. As those positions are located away from the striker, he might be able to pass the ball to the supporter. The *goalkeeper* is the only role that is fixed during the complete game.

- Dynamic, but constrained positioning for all roles. For example, the defender is not allowed to pass the center line.
- Configurable strategies allowing user to switch between more defensive and offensive play. The configuration affects specific positions of the field players that are not in charge of the ball.
- Kicking motions can be performed during the walk without stopping.
- A kick selection that decides in which direction the ball is shot. In case of obstacles in front of the robot, the ball is moved edgewise around the obstacle. If the robot is free, it might pass the ball to a team member or shoot to the goal.
- A goalkeeper which is capable to counter the ball by jumping or making a quicker block motion in case that the ball would pass near the keeper.
- Adaptation to the current number of team players during the game. If robots are taken out, roles in the following order are removed: supporter, striker-backup, defender.
- Kickoff positioning independent from the robot number (except goalkeeper).

One drawback of the proposed behavior is its reliance on the wireless network. This affects especially the distribution of percepts that are needed for the world model. There were major problems with the wireless (delays, framedrops) during the RoboCup competition in the past years. For 2014, a fallback strategy was introduced that does not rely on wireless communication. Instead, it uses fixed roles that prioritize the main role objective (e.g. defender), although every role would go to the ball if it is near enough. To avoid clumping of our robots in the latter situation, every robot uses the visual robot detection to check if another one of our robots is near the ball. This strategy was also used during the DropIn games, since the data of the team mates was in many cases considered unreliable.

### **HeadControl**

In 2013, we decided to redesign the head control along with the behavior. In previous years, each separate behavior had customized head motions implemented in XABSL. We now introduced high level behavior commands that trigger head motions. For that purpose, a new module called *Head Control Engine* was added.

The engine provides five different head control modes:

- **ball:** A mode that fixates the ball. If this is physically impossible, a simple sweep along the walking path is executed. If the ball gets lost, a search for the ball is started.
- **localize:** This mode tries to improve the localization validity by sweeping the field, and looking at points of interest for localization. Such locations are goal posts, the ball, and line crossings.

- **opponents**: This mode is intended to be used by a stationary observer, sweeping the field for obstacles and verifying their positions.
- **soccer**: This is a high-level mode that is now being used by our team for the game play. It uses aforementioned functions. Additionally, while walking towards a target, the robot moves its head to keep path, walking target and obstacles on its way in sight.
- **direct**: This option allows the behavior engineer to directly set the target positions of the head joints.

## 4.2 Path Planning

The motion commands used in previous years have been based on desired speed vectors which were updated with every behavior execution. This mode of control dates back to the AIBOs which could be controlled like omnidirectional vehicles. Additionally for an AIBO it was sufficient to walk straight to the ball to grab and turn with it. For humanoid robots however the the omnidirectional characteristic of the walking generation is much less distinct, and at the same time a target position close to the ball has to be reached with a certain target orientation which increases the difficulty for trajectory control. While it is possible and also commonly done to generate omnidirectional walking patterns with the walking engine described in section 2.1, for a humanoid robot such as the Nao it is far more convenient to walk straight than it is to walk sideways. This is reflected in the possible walking speeds in each direction. Generating smooth path trajectories following the characteristics optimal for those described walking capabilities is obviously not possible in an intuitive way using a state-based behavior description language such as XABSL, or CABSL.

To overcome those limitations and ultimately to achieve more precision and speed in positioning close to the ball, a more advanced approach to path planning has been done for the *Nao Devils*' robots since RoboCup 2010. The utilized *Dortmund Walking Engine* is based on foot step planning (see section 2.1). In 2010, a basic behavior has been introduced to XABSL allowing the trajectory planning to be done by the walking engine which has much better control and feedback about the executed motion. The motion request has been adapted accordingly to accept a target position and orientation and different *go\_to* commands have been implemented to cover common motion tasks while avoiding obstacles. In 2013, the command has been improved continuously by realizing a path planning through a potential field generated from static and dynamic obstacles.

Although the approach using a potential field worked, the robots were very cautious and inefficient, especially when avoiding dynamic obstacles. For 2014, a much simpler and more efficient approach has been implemented. The path starts with just two waypoints, the robot and the target position. Until there are no obstacles on any line between two waypoints, for each obstacles on such a line a new waypoint next to the obstacle is generated. For a more stable path, obstacles that are close to each other are merged. This approach, in contrast to the path which the potential field produced, allows fast and predictive obstacle avoidance while minimizing the need to slow down when the robot is near such an obstacle.

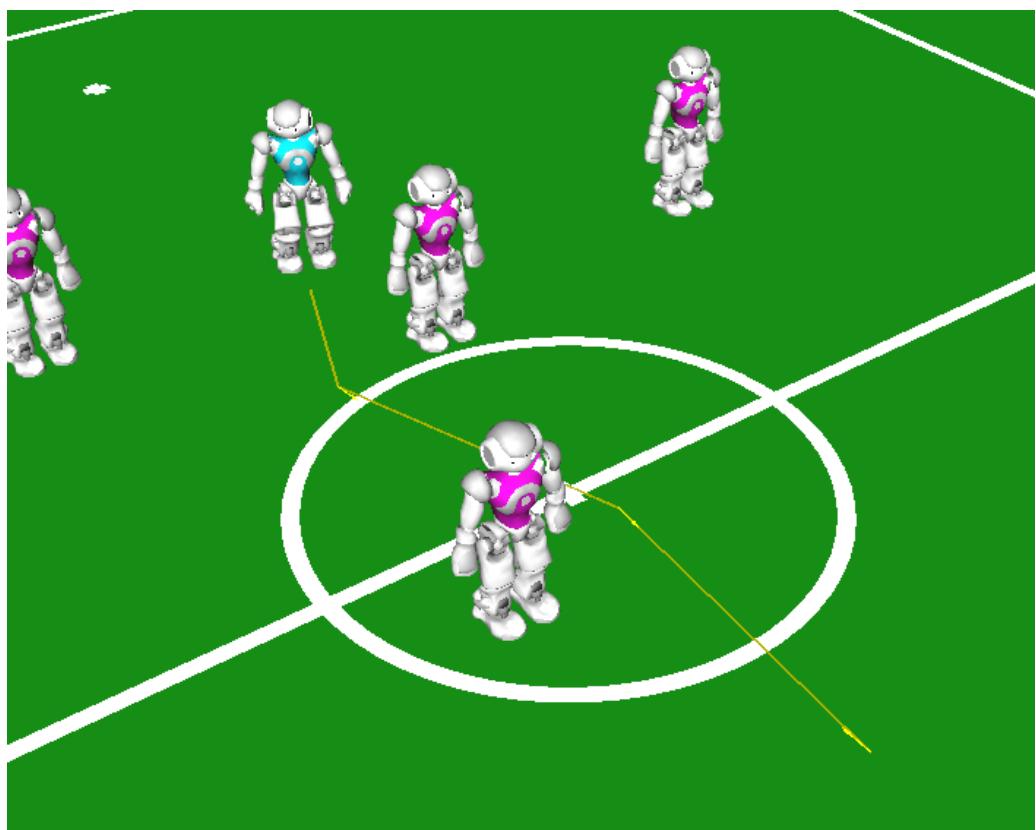


Figure 4.2: Example of a path around obstacles.

## Chapter 5

# Conclusion and Outlook

The *Nao Devils Dortmund* is a team from the TU Dortmund University with roots in several other teams which have competed in RoboCup competitions over the last years. This team report covered a short overview of the main ideas and concepts that were employed and successfully used in RoboCup 2016. This affects mainly the newly introduced outdoor competition.

A lot of effort has been put into the configuration capabilities and robustness of the system. To this end, the calibration-free image processor has been further developed and fixed.

For RoboCup 2017, beside integrating the new soccer rule changes, we plan to revise the team play and overall soccer behavior. Moreover, we plan to improve ball detection by employing a convolution neural network. We will continuously improve

Together with Berlin United, we aim to participate in the Mixed Team Competition.

## Appendix A

# Walking Engine Parameters

In this section the most important parameters to control the Walking Engine are described. They can be found in the file <coderelease directory>/Config/walkingParams.cfg. Some parameters needed by the ZMP/IP-Controller are located in the file <coderelease directory>/Util/Matlab/NaoV3.m. The Matlab script writeParamsV3.m calculates the values used by the ZMP/IP-Controller using the parameters in this file and stores the results to <coderelease directory>/Config/Robots/<robot name>/ZMPIPController.dat. To execute the script open Matlab, change the current directory to <coderelease directory>/Util/Matlab and enter the following command:

```
writeParamV3(NaoV3('<robot name>'))
```

All values are expressed in SI units, unless otherwise stated.

### A.1 File walkingParams.cfg

The file walkingParams.cfg consists of the following parameters:

#### **footPitch**

To avoid hitting the ground with the forward section of the foot an offset is applied to the foot rotation around the y axis. The added value reaches its maximum at the middle of the single support phase. The maximum can be set using footPitch.

#### **xOffset**

The center of the feet can be shifted along the x axis by setting this value unequal to 0. This offset is constant for the whole walk.

#### **stepHeight**

Maximum z position of the foot during the single support phase, see section 2.1.4.

### **sensorControlRatio**

The sensorControlRatio is multiplied with the difference between the target ZMP and the measured ZMP. Legal values are [0...1], where 0 means ‘no sensor control’ and 1 ‘full sensor control’.

### **doubleSupportRatio**

Proportion of the double support phase during a step.

### **crouchingDownPhaseLength, startingPhaseLength, stoppingPhaseLength**

These values define the duration of the transitions between a walk and special actions.

### **armFactor**

The movement of an arm depends on the x coordinate of the opposite foot. The x coordinate is multiplied with armFactor and applied to the ShoulderPitch.

### **arms1**

This value is the angle of ShoulderRoll for both arms. It is constant during the walk.

### **zmpSmoothPhase**

To avoid hard sensor feedback during transitions from special actions to walk and vice versa the ZMP error is faded in and out. This parameters sets the length in frames of this phase.

### **maxSpeedXForward, maxSpeedXBack, maxSpeedYLeft, maxSpeedYRight, maxSpeedR, maxSpeedXForwardOmni**

These values define the maximum speed. All requests sent to the Walking Engine are clipped to these values.

### **stepDuration**

The PatternGenerator defines the footsteps based on two single support phases and two double support phases. The duration of all together is the stepDuration.

### **footYDistance**

Distance between the feet.

### **stopPosThresholdX, stopPosThresholdY, stopSpeedThresholdX, stopSpeedThresholdY**

Stopping a walk and executing a special action is only possible when the Walking Engine sets a flag which indicates that it does not compromise the stability. The Walking Engine uses this 4 indicators to check if the speed of the center of mass is low enough and the position within a stable range.

**zmpLeft, zmpRight**

Position of the target ZMP in the (left/right)foot coordinate system when the robot stands on one leg.

**outFilterOrder**

Before sending the angles to the robot they are low-pass filtered. This parameter defines the order of the filter.

**tiltControllerParams, rollControllerParams**

Gains for the tilt/roll PID controller params implemented in the module GyroTiltController.

**sensorDelay**

The sensor control relies on a good comparison between the measured ZMP and the target ZMP. In most cases the measurements are some frames old and must be compared with the target of the same frame.

**halSensorDelay**

Delay of the hal sensors used to measure the joint angles.

**maxFootSpeed, fallDownAngle**

The Walking Engine has an integrated check for instabilities to avoid breaking joints. Due to the sensor control the robot can react in an unexpected way resulting in very fast movements. The maximum speed of the feet can be limited by using maxFootSpeed. Additionally all movements are stopped immediately when the robot exceeds the maximum tilt or roll angle defined by fallDownAngle.

**polygonLeft, polygonRight**

As mentioned in section 2.1.3 the reference ZMP is calculated using a Bézier curve. This parameters define the y coordinates of the control points in the corresponding foot coordinate system.

**offsetLeft, offsetRight**

Due to high torques acting on the joints during the single support phase large angle errors can be observed. To compensate the effects a constant offset is added to each leg joint which can be configured using these parameters.

**walkRequestFilterLen**

To avoid abrupt speed changes a low-pass filter can be activated to filter the incoming speed requests. A value of 1 means deactivated.

### **accXAlpha**

Gain controlling the acceleration sensor feedback (see section 2.1.2).

### **maxAccX, accDelayX, maxAccY, accDelayY, maxAccR, accDelayR**

Besides the low-pass filter there is an other way to limit speed changes. If the acceleration resulting from the new speed request is higher than maxAcc only maxAcc is applied until accDelay frames are elapsed.

### **speedApplyDelay**

This is the third way to limit speed changes. If a speed change has been detected further speed changes are ignored for speedApplyDelay frames.

### **legJointHardness**

The stiffness parameter for the leg joints. The stiffness cannot be set for each leg separately.

### **heightPolygon**

This five values are multiplied with stepHeight to get the z coordinates of the control polygon for the swinging leg as explained in section 2.1.4.

### **standRollFactor**

A factor especially for standy on one leg. Instead of reaching he target center of mass position by translating the body, this factor allows to reach the target by rotating the body.

## **A.2 File NaoNG.m**

The following parameters can be found in the file NaoNG.m:

### **g**

The Gravity.

### **z\_h**

Target height of the center of mass over the ground.

### **dt**

Length of one frame.

### **R**

Controller-Parameter R [14].

**N**

Duration of the preview phase as explained in section 2.1.2.

**Qx**

Controller-Parameter Qx [14].

**Qe**

Controller-Parameter Qe [14].

**Ql**

Gain for calculating L [14].

**RO**

Gain for calculating L [14].

**path**

Path to the file ZMPIPController.dat.

# Bibliography

- [1] Czarnetzki, S., Kerner, S.: Nao Devils Dortmund Team Description for RoboCup 2009. In Baltes, J., Lagoudakis, M.G., Naruse, T., Shiry, S., eds.: RoboCup 2009: Robot Soccer World Cup XII Preproceedings, RoboCup Federation (2009)
- [2] Czarnetzki, S., Hebbel, M., Kerner, S., Laue, T., Nisticò, W., Röfer, T.: BreDoBrothers Team Description for RoboCup 2008. In Iocchi, L., Matsubara, H., Weitzenfeld, A., Zhou, C., eds.: RoboCup 2008: Robot Soccer World Cup XII Preproceedings, RoboCup Federation (2008)
- [3] Hebbel, M., Nisticò, W., Kerkhof, T., Meyer, M., Neng, C., Schallaböck, M., Wachter, M., Wege, J., Zarges, C.: Microsoft hellhounds 2006. In: RoboCup 2006: Robot Soccer World Cup X RoboCup Federation, RoboCup Federation (2006)
- [4] Röfer, T., Brunn, R., Czarnetzki, S., Dassler, M., Hebbel, M., Jüngel, M., Kerkhof, T., Nisticò, W., Oberlies, T., Rohde, C., Spranger, M., Zarges, C.: Germanteam 2005. In Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y., eds.: RoboCup 2005: Robot Soccer World Cup IX Preproceedings, RoboCup Federation (2005)
- [5] Czarnetzki, S., Hebbel, M., Nisticò, W.: DoH!Bots Team Description for RoboCup 2007. In: RoboCup 2007: Robot Soccer World Cup XI Preproceedings. Lecture Notes in Artificial Intelligence, Springer (2007)
- [6] Röfer, T., Laue, T., Weber, M., Burkhard, H.D., Jüngel, M., Göhring, D., Hoffmann, J., Altmeier, B., Krause, T., Spranger, M., Schwiegelshohn, U., Hebbel, M., Nisticò, W., Czarnetzki, S., Kerkhof, T., Meyer, M., Rohde, C., Schmitz, B., Wachter, M., Wegner, T., Zarges, C., von Stryk, O., Brunn, R., Dassler, M., Kunz, M., Oberlies, T., Risler, M.: GermanTeam RoboCup 2005. Technical report (2005) Available online: <http://www.germanteam.org/GT2005.pdf>.
- [7] Röfer, T., Laue, T., Müller, J., Fabisch, A., Feldpausch, F., Gillmann, K., Graf, C., de Haas, T.J., Härtl, A., Humann, A., Honsel, D., Kastner, P., Kastner, T., Könemann, C., Markowsky, B., Riemann, O.J.L., Wenk, F.: B-human team report and code release 2011 (2011) Only available online: [http://www.b-human.de/downloads/bhuman11\\_coderelease.pdf](http://www.b-human.de/downloads/bhuman11_coderelease.pdf).
- [8] Laue, T., Spiess, K., Röfer, T.: Simrobot - a general physical robot simulator and its application in robocup. In Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y., eds.: RoboCup 2005: Robot Soccer World Cup IX. Number 4020 in Lecture Notes in Artificial Intelligence, Springer; <http://www.springer.de/> (2006) 173–183

- [9] Röfer, T., Laue, T., Richter-Klug, J., Schünemann, M., Stiensmeier, J., Stolpmann, A., Stöwing, A., Thielke, F.: B-Human team report and code release 2015 (2015) Only available online: <http://www.b-human.de/downloads/publications/2015/CodeRelease2015.pdf>.
- [10] Vukobratovic, M., Borovac, B.: Zero-moment point – Thirty five years of its life. *International Journal of Humanoid Robotics* **1** (2004) 157–173
- [11] Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Yokoi, K., Hirukawa, H.: Biped walking pattern generator allowing auxiliary zmp control. In: IROS, IEEE (2006) 2993–2999
- [12] Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., Hirukawa, H.: Biped walking pattern generation by using preview control of zero-moment point. In: ICRA, IEEE (2003) 1620–1626
- [13] Urbann, O., Tasse, S.: Observer based biped walking control, a sensor fusion approach. *Autonomous Robots* (2013) 1–13
- [14] Czarnetzki, S., Kerner, S., Urbann, O.: Observer-based dynamic walking control for biped robots. *Robotics and Autonomous Systems* **57** (2009) 839–845
- [15] Czarnetzki, S., Kerner, S., Urbann, O.: Applying dynamic walking control for biped robots. In: RoboCup 2009: Robot Soccer World Cup XIII. Lecture Notes in Artificial Intelligence, Springer (2010) 69 – 80
- [16] Urbann, O., Hofmann, M.: Modification of foot placement for balancing using a preview controller based humanoid walking algorithm. In: Proceedings RoboCup 2014 International Symposium, Eindhoven, Netherlands (2014)
- [17] Hebbel, M., Kruse, M., Nisticò, W., Wege, J.: Microsoft hellhounds 2007. In: RoboCup 2007: Robot Soccer World Cup XI Preproceedings, RoboCup Federation (2007)
- [18] Nisticò, W., Hebbel, M.: Temporal smoothing particle filter for vision based autonomous mobile robot localization. In: Proceedings of the 5th International Conference on Informatics in Control, Automation and Robotics (ICINCO). Volume RA-1., INSTICC Press (2008) 93–100
- [19] Jochmann, G., Kerner, S., Tasse, S., Urbann, O.: Efficient multi-hypotheses unscented kalman filtering for robust localization. In Röfer, T., Mayer, N.M., Savage, J., Saranli, U., eds.: RoboCup 2011: Robot Soccer World Cup XV. Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2012) to appear
- [20] Quinlan, M.J., Middleton, R.H.: Multiple model kalman filters: A localization technique for robocup soccer. In: RoboCup 2009: Robot Soccer World Cup XIII. Lecture Notes in Artificial Intelligence, Springer (2010) 276–287
- [21] Czarnetzki, S., Rohde, C.: Handling heterogeneous information sources for multi-robot sensor fusion. In: Proceedings of the 2010 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI 2010), Salt Lake City, Utah (2010) 133 – 138

- [22] Hauschildt, D., Kerner, S., Tasse, S., Urbann, O.: Multi body kalman filtering with articulation constraints for humanoid robot pose and motion estimation. In Röfer, T., Mayer, N.M., Savage, J., Saranli, U., eds.: RoboCup 2011: Robot Soccer World Cup XV. Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2012) to appear
- [23] Lötzsch, M., Bach, J., Burkhard, H.D., Jüngel, M.: Designing agent behavior with the extensible agent behavior specification language XABSL. In Polani, D., Browning, B., Bonarini, A., eds.: RoboCup 2003: Robot Soccer World Cup VII. Volume 3020 of Lecture Notes in Artificial Intelligence., Padova, Italy, Springer (2004) 114–124