technische universität
dortmund

Nao Devils

Team Report 2019

Ingmar Schwarz, Oliver Urbann, Aaron Larisch, Dominik Brämer

Robotics Research Institute
Section Information Technology

Contents

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.

Stable motions are of crucial importance in the context of biped robots. Thus, the following chapter of the document will describe the motion control process emphasizing the *Dortmund Walking Engine* which has been the first closed loop walking engine applied to the Nao in RoboCup 2008. The version of RoboCup 2019 was able to reach walking speeds of up to $35cm/s$ on artificial turf, while ensuring a stable walk. Compared to 2018, we revised the sensor control mechanisms of our walking engine, see section 2.1.5. Additionaly, we added our own key frame engine, see section 2.3.

Together with team HULKs we participated in the Mixed-Team-Competition as the joint team "Devil SMASH", winning the third place. The behaviorial approach is described in chapter 6.

This code release includes all software used at RoboCup 2019, except our behavior.

We want to thank the team B-Human for their great work developing their framework, which provides the base for our own developed modules. The specific changes are listed in section 1.2.

1.1 Team Description

The *Nao Devils* 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

Figure 1.1: The *Nao Devils* team members at RoboCup 2019. From left to right: Max Brämer, Sebastian Hoose, Torben Seeland, Maximilian Otten, Janine Frickenschmidt, Arne Moos, Aaron Larisch and Ingmar Schwarz.

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 years. Major recent successes were the 1st place in the Outdoor Competition at RoboCup 2016, the 1st place at the technical Challenges at RoboCup 2016, the 3rd place at GermanOpen 2017 as well as the second place at the German Open 2018. At RoboCup 2017 we finished third place and reached the quarter final at last year's Champions Cup as well as the 2nd place in the mixed team competition. At RoboCup 2019, we finished in 4th place and reached the second place in the "Directional Whistle" challenge, see chapter 6. Throughout the year we take part in several local workshops and events such as our own workshop, RoDeO in Dortmund, Germany, or the RoHOW workshop in Hamburg, Germany.

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 code release 2015 of team B-Human[1]. Compared to the 2015 code release of team *B-Human* we changed or adapted the following parts (minor changes, changes to behavior or config files are omitted):

[1] https://github.com/bhuman/BHumanCodeRelease/releases/tag/coderelease2015

- Changed the V5 camera driver to better fit our needs[2].
- Changed the cognition process, the update of the camera images and the simulator to get have both the upper and the lower image in one cognition frame, reducing the cognition frame rate to 30 fps.
- Updated the CABSL base class to the version released in 2016 by *B-Human*.
- Updated *B-Human's* KickEngine to their 2016 version and changed the sensor control part as well as the framework integration to fit our needs.
- Replaced the other motion modules with our own.
- Replaced all vision and modeling modules with our own.
- Replaced most of the team communication with our own.
- Updated the infrastructure to our needs, including the deploy tool "bush", which we replaced with "dorsh".
- Implemented our own automatic camera calibration procedure (see Section 3.5).

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

To test developments in simulation, the software *SimRobot* was used instead of commercial alternatives, such as *Webots* from Cyberbotics[3]. 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 [6] is a kinematic robotics simulator developed in Bremen which (like Webots) utilizes the Open Dynamics Engine[4] (ODE) to approximate solid state physics.

[2]The changed driver can be downloaded at https://github.com/NaoDevils/NaoKernel
[3]`http://www.cyberbotics.com/`
[4]`http://www.ode.org/`

1.3 Getting started

This section describes the steps to set up a Nao robot to be ready to play with our code. It assumes you have already followed the instructions of the "README.md" file in our code release.

Our procedure consists of three calibration steps, after the code was compiled and deployed to the robot:

- Calibrate the camera matrix, see section3.5.
- Check the camera image settings. Since we use auto calibration we only adjust the settings for very bright or very dark environments. In both cases we adjust our auto calibration parameters[5].
- Calibrate the walking engine. Usually we only calibrate our sensor controls via the "csConverter2019.cfg" file and adjust center of mass offsets in the robots walking parameter file "walkingParamsFLIPM.cfg".

All other important setup options can be set with our version of*B-Human's* deploy tool *dorsh*.

[5]For V5 adjustments, see the documentation in our camera driver at https://github.com/NaoDevils/NaoKernel for more details.

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) [7]. 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 [8]. 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.

$$p_x = x - \frac{z_h}{g}\ddot{x} \tag{2.1}$$

$$p_y = y - \frac{z_h}{g}\ddot{y} \tag{2.2}$$

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 [9]. Figure 2.1 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.

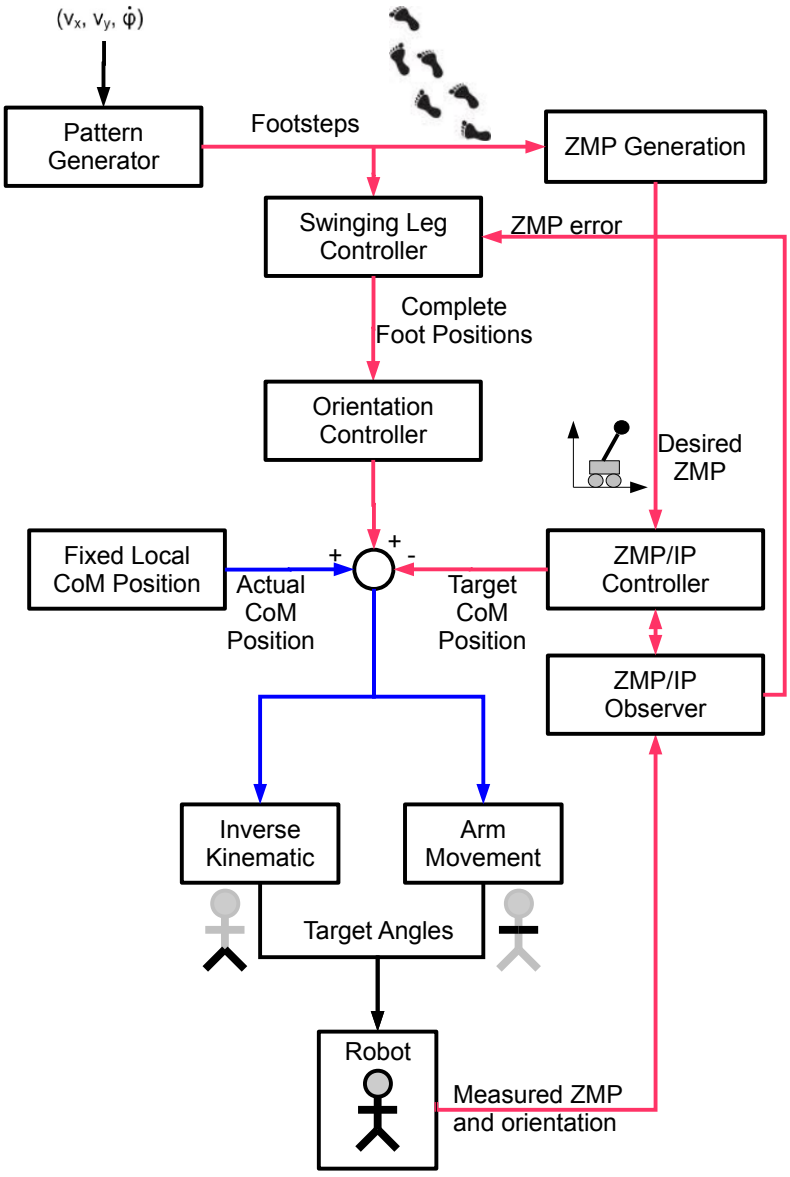


Figure 2.1: Control structure visualization of the walking pattern generation process. Blue lines refer to robot coordinate system and red lines refer to the global coordinate system.

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 [10]. Equation 2.3 provides the foot positions in a robot centered coordinate frame.

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

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. For the RoboCup 2017, we used the FLIP model which we recently introduced[11]. This entire section is based on the findings and description of the paper. We derived a discrete state-space representation that can be applied by a preview controller as proposed by Kajita et al.[12].[1] The well-known LIPM proposed by Kajita et al.[12] is a simplification of the robots dynamics to a single center of mass and is usually applied to derive the Zero Moment Point (ZMP) from its dynamics. The ZMP p of this system can be calculated by:

$$p = c_1 + \ddot{c_1} \cdot \frac{z_h}{g} \tag{2.4}$$

where g is the gravity and c_1 the position of the CoM. The linearity is a consequence of the constant height of the CoM and important to design closed-form algorithms for motion generation. We extended the system by a spring and a damper. To retain linearity we added a second cart with an additional small mass as depicted in Fig. 2.2. Here, two carts are connected via a damper and a spring (with constants b and k respectively). The left cart (cart 1) has no mass but is connected to CoM 1 with constant height zh by a pole of variable length. Similarly to the variable length of the pole, the length of the cart is also variable. It is adapted such that the spring and the damper is always connected to the cart at the position $c1$ of CoM 1. The right cart (cart 2) has the additional CoM 2 and is the only cart that is accelerated by u. The ZMP of cart 1 is located at p and CoM 1 and 2 at c_1 and c_2 respectively. FLIPM represents all flexible parts that are accelerated by the motors, e.g. the gears. A spring and a damper connects it to the cart with the large CoM that is only accelerated by the force exerted by the spring and the damper. To control the system only the second cart is actively accelerated. More deriviations, and mathematical details of the work can be found in[11].

To derive a controller that is able to generate the walking motion, we chose to define the walk by a reference trajectory of the ZMP as proposed by Kajita et al.[12]. We

[1] A script containing all derived equations and the preview controller can be downloaded here: https://github.com/OliverUrbann/FLIPM/

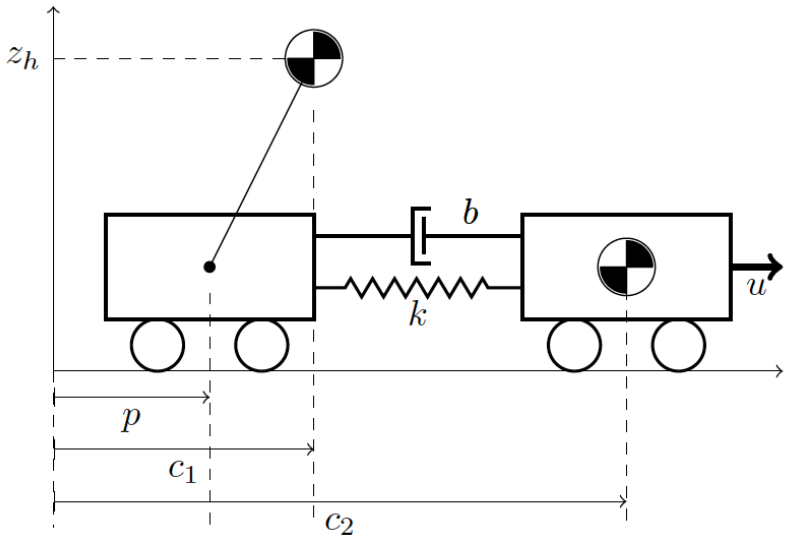


Figure 2.2: Flexible Linear Inverted Pendulum Model (FLIPM).

therefore define the output of the system as:

$$p_k = c \cdot x_k \tag{2.5}$$

$$c = (1, 0, -\frac{z_h}{g}, 0, 0, 0) \tag{2.6}$$

As described in [2] it is not sufficient to de

ne the desired ZMP of the current time frame. A preview of approximately 1s is required. The controller u_k is given by:

$$u_k = -G_I \sum_{i=0}^{k} [C_{x_i} - p_i^{ref}] - G_x x_k - \sum_{j=1}^{N} G_{d,j} p_{k+j}^{ref}, \tag{2.7}$$

where the gains G_I, G_x and $G_{d,j}$ that minimize a given cost function are derived by applying the procedure proposed in [8]. It includes the solution of a discrete matrix Riccati equation that can be done online before the walk is generated online on the robot. On platforms like the NAO, the input of the joints are the desired joint angles. They can be calculated by applying an inverse kinematic on the desired foot positions f_R given in a local coordinate system of the robot. They are given by the following equation:

$$f_R = c_b - c_2 + f_W, \tag{2.8}$$

where f_W are the desired foot positions in world coordinate system and c_b the actual CoM position of the robot in the local coordinate system. We determine f_W together with the reference ZMP and calculate c_b by measuring the current joint angles and apply

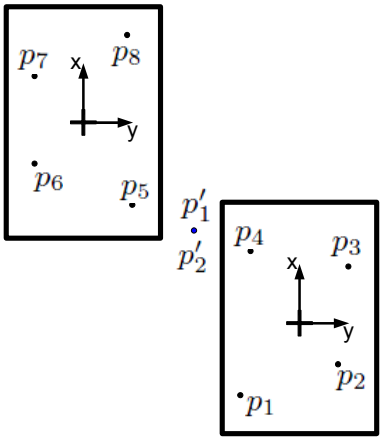


Figure 2.3: The control polygon consists of 4 points per foot with constants coordinates within the respective coordinate frame. In this example a positive x speed is assumed for a better visualization.

a forward kinematic to combine c_b from all CoMs of the links. As can be seen in Eq. 5, it is not possible to apply the controller output directly to the robot, which is the third derivative of c_2. For further details please see [8]. Assuming the physical parameters (k, b, m_1 and m_2) are selected correctly, the CoM of the physical robot follows the large mass of the model.

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 2.3 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 2.4 shows the resulting reference ZMP along the y axis.

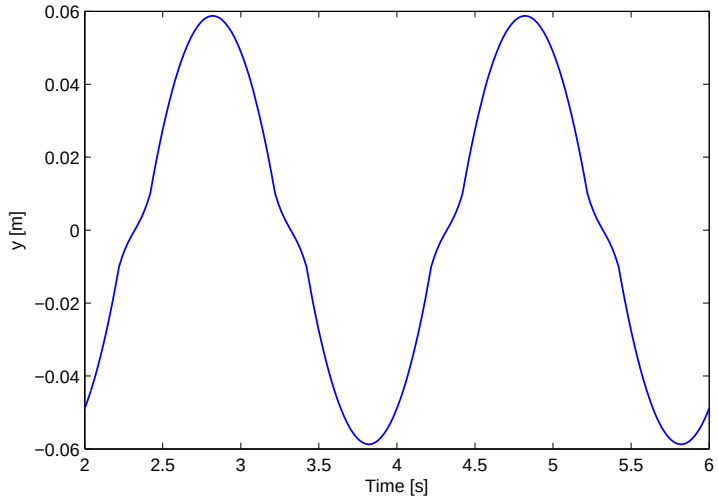


Figure 2.4: Resulting reference ZMP along the y axis.

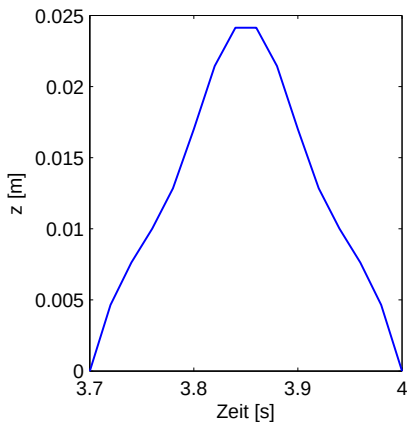
2.1.4 Swinging Leg Controller

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 coordinates are increased and decreased respectively by $\frac{1}{6}$ of the maximal step height. The z coordinate over time can be seen in figure 2.5a. To reduce the influence of the leg inertia the feet are lifted and lowered at the same speed. Figure 2.5b 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.

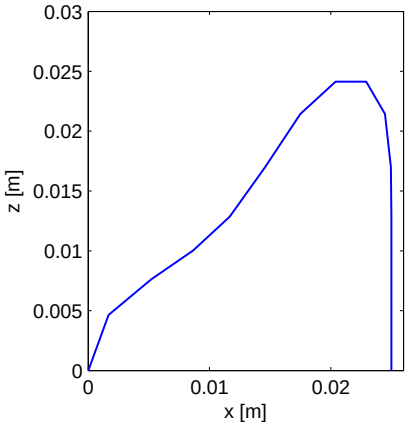
Sensorfeedback

In addition to our previous methods of stabilization[13], we use several PID controllers which we moved into one module, the CSConverter2019, this year. For these controllers we use the gyro and angle sensors provided by the NaoQi framework as well as our own IMU model, based on the code from team Berlin United. Several error correction methods are applied there.

- Center of mass shift on body angle error as well as on arm movement to avoid enemy contact,
- a PID controller for the hip and ankle joints based on filtered gyro and body angle error and
- leg rotation of the swing leg to keep the foot parallel to the ground on body angle error.



(a) Step height over time.



(b) Stepheight over x for a walk along the x axis.

Figure 2.5: Movement of the swinging leg in the global coordinate system.

This led to an increased stability while fighting for the ball, during side steps and when walking with high speeds of ¿30 cm/s.

2.1.5 Preview Reset

The Dortmund WalkingEngine uses a preview controller[11] for generating a stable center of mass trajectory. This has the disadvantage that we can not change steps that are currently used for this preview resulting in a delayed motion reaction on the field. In 2018 we removed this restriction by resetting our ZMP generation after each step. This leads to small disturbances in the resulting controller output but can be migitated by an interpolation between the old and new (reset) ZMP controller output. At the German Open 2018 we successfully tested this in a real soccer match and used this in every game since RoboCup 2018.

Results and future work

We successfully removed the preview phase for our walking engine, improving the reactivity of our robots greatly.

For the next year we hope to rewrite our several sensor feedback mechanisms to unify them and stabilize our walking engine further.

2.2 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. This section describes our other method of kicking which is integrated into our walking engine. With this method

we can can design any step pattern using start and stop positions, a 3D step trajectory as well as the step duration for each individual step.

2.2.1 Kicking while walking

Since 2016 we are using specific, predefined steps in our walking as fast and small kicks in one-on-one situations. For this year we improved these kicks by giving each step a custom swing foot trajectory. This enabled us to create stronger kicks without leaving our walking engine.

In addition we have implemented prepended steps for kicks to smoothe our transition between the kick decision of our behavior and our kick execution. Different kicks right now would need a different approach point in our behavior. If the ball would move in the last second we would not hit the ball at all. This can be adressed by prepending steps to our kicks dynamically following the latest ball percepts (see Figure 2.6). To get a constant speed throughout these steps, we need to take the coordinate system changes from a possible rotation into account and solve this problem iteratively until the required steps are within our speed limits.

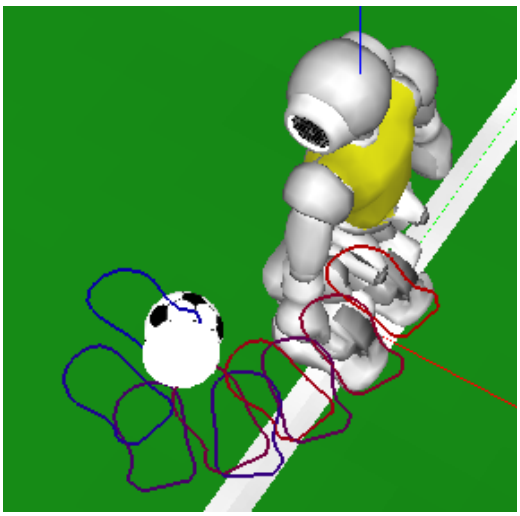


Figure 2.6: Prepended steps calculated for a rotated kick to the goal with the right foot achieving a constant velocity in all dimensions. Usually this many steps would not be needed.

Once the ball is close enough to the robot, we use the latest, fixed, set of steps to avoid flickering kick decisions and ensure a more stable kick process. This has sped up our kick behavior greatly.

For 2019 we added several new kicking motions in order to reduce the time to set up a kick.

2.3 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.

This year we introduced a new module named *Key Frame Engine* which adds additional configuration and stabilization possibilities to the fixed motions. For every key frame we can set a target upper body angle that can be used for a balancer which tries to compensate for the execution error or stop the motion if the error is too great. This is especially helpful for stand up motions. In addition we can choose to wait until the body motion has stopped, which we check using the gyroscope. This has been used to prevent interpolation into key frames or other motion types from an unstable robot. Lastly, we can use a upper body tilt correction based upon the hip yaw error, similar to the concepts used in the B-Human and UNSW frameworks.

The Key Frame Engine replaces the old module completely.

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 3.2. Special focus is given to new developments since 2016, meaning the use of a neural network to detect the ball. Additionally we describe our new automatic calibration of our cameras position (see section 3.5).

3.1 Camera Settings

In order to capture sharp images with even illumination, we adjusted the Nao V6 camera parameters to fulfill the requirements of our image processing. Playing with enabled auto exposure is mandatory for us, if we want to play under changing lighting conditions like outdoors or near windows. Because the Nao's default camera parameters are not optimized for this purpose, we basically changed three settings:

- **Auto expsoure window:** Because the robot might look partly into the outside or a very bright area with its upper camera, especially when looking straight ahead, we decided to exclude the upper third of the upper image from the exposure window. Otherwise, the brightness of the field might decrease too much. The camera offers settings that allow to adjust the image coordinates of the window that is used for the internal brightness averaging algorithm used for the auto exposure.
- **Maximum exposure:** The robot's cameras are optimized for stationary capturing that prefer to increase the exposure time before raising the gain if the lighting

conditions become darker. That is counterproductive for us, because the cameras are moving a lot during the walk and the head motions, which increases the motion blur. However, the camera has a feature called "night mode" that can be used to limit the maximum exposure time to e.g. 10ms. That slightly increases the noise, but reduces the motion blur, which is more important for us.

- **Target image luminance:** In order to tune the overall image brightness needed for our image processing, we also change the target image luminance thresholds that are used by the auto exposure. The camera offers settings for a stable and unstable image luminance region. If the measured image luminance exceeds the limits of the unstable range, the auto exposure starts adjusting the exposure and gain until the image luminance reaches a value inside the stable range again. That allows to adjust the image brightness precisely without changing the parameters too frequent.

3.2 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 [14]. 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. Since 2012 we play our games with our camera set on auto exposure and auto white calibration. We however cahnged the camera driver to use all the auto calibration features of the camera, which is available online[1] and is based on B-Human's camera driver[2] from 2015.

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 y-channel changes as well as possible obstacles.

All relevant objects and features, except the goal posts to save runtime, 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 current implementation allows us to play in natural lighting conditions and as shown in the open challenge for 2014 and in the outdoor competition in 2016. Using auto camera settings, our robots continue to play and score even with huge lighting condition changes in games.

[1] https://github.com/NaoDevils/NaoKernel
[2] https://github.com/bhuman/BKernel

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.

3.3 Ball Detection

Our ball detection is based on the idea of a R-CNN pipeline. It starts with a heuristic approach to extract region proposals which is comparable to our complete ball detection process before 2017 [15]. By using scanlines we first identify regions that are most likely not considered background (i.e. carpet). As we use scanlines only on the field and stop at the field border we ignore all balls or similar objects outside of the field. We then check if the found object is at least partially round and whether it is within expected bounds. This results in up to approximately 50 region proposals per image. These are resized to 16x16 grayscale images. On the region proposals we apply a small CNN for classification. The structure is:

1. Convolution 3x3, Pad "same", 4 Filter, Activation ReLU
2. Max-Pooling 2x2
3. Convolution 3x3, Pad "same", Activation ReLU, 8 Filter
4. Max-Pooling 2x2
5. Convolution 3x3, Pad "same", Activation ReLU, 16 Filter
6. Max-Pooling 2x2
7. Dropout 0.4
8. Convolution 2x2, Pad "valid", Activation Softmax, 2 Filter

We use Python and Keras to implement the model, training and validation. We use a custom loss function for training as false positives are way more confusing our game play than false negatives:

$$-y_b \cdot log\left(\frac{\hat{y}_b}{\hat{y}_b + \hat{y}_n}\right) \cdot w_b - y_n \cdot log\left(\frac{\hat{y}_n}{\hat{y}_b + \hat{y}_n}\right) \cdot w_n,$$

where y_b and $\hat{y}_b$ are the ground truth and predicted values for the ball class respectively and y_n, $\hat{y}_n$ for the non-ball class and w_b and w_n the weights for the ball and non-ball class respectively. We set $w_n = 0.999$ and $w_b = 0.001$.

To speedup the execution on the robot, the model is quantized after training to 8 bit integer with an additional error of $< 1\%$. For the execution we compile the model to ANSI C code with no dependencies to a library which has two advantages. First, it simplifies the deployment significantly as no additional library or software is required on the robot. We just copy the code file into our project. Second, we can apply various code generation schemes to speed up the execution, e.g. highly optimized SSE proto-snippets and loop unrolling. Further details can be found in [16].

3.4 Robot Detection

We created an infrastructure to train and run CNNs based on the YOLO[17] architecture on the Nao robot. At RoboCup 2019 we updated our networks to run on both the upper and the lower image. While both networks look for robots, the lower image network also puts out possible ball candidates which are then validated by the above classification network. The upper image network runs at 5.1ms on the Nao V6 with an input of 160x120 RGB image and the following structure:

1. Seperable Depthwise Convolution 3x3, Pad "same", Depth Multiplier 4, 3 Filter, Stride 2x2, Batch Normalization, Activation Leaky ReLU
2. Convolution 1x1, Pad "same", Activation Leaky ReLU, 16 Filter
3. Seperable Depthwise Convolution 3x3, Pad "same", Depth Multiplier 4, 16 Filter, Stride 2x2, Batch Normalization, Activation Leaky ReLU
4. Convolution 1x1, Pad "same", Activation Leaky ReLU, 24 Filter
5. Seperable Depthwise Convolution 3x3, Pad "same", Depth Multiplier 4, 24 Filter, Stride 2x2, Batch Normalization, Activation Leaky ReLU
6. Convolution 1x1, Pad "same", Activation Leaky ReLU, 16 Filter
7. Seperable Depthwise Convolution 3x3, Pad "same", Depth Multiplier 4, 16 Filter, Batch Normalization, Activation Leaky ReLU
8. Convolution 1x1, Pad "same", Activation Leaky ReLU, 32 Filter
9. Seperable Depthwise Convolution 3x3, Pad "same", Depth Multiplier 4, 32 Filter, Stride 2x2, Batch Normalization, Activation Leaky ReLU
10. Convolution 1x1, Pad "same", Activation Leaky ReLU, 40 Filter
11. Fully Connected 8x10x6

As the output suggests, only one box per grid cell is created for a single class, which reduces the network complexity greatly.

Our lower image network has a similar runtime with a 80x60 RGB image input, but needs to detect two objects - robot and ball - and is structured differently:

1. Convolution 3x3, Pad "same", 16 Filter, Batch Normalization, Activation Leaky ReLU
2. Max-Pooling 2x2
3. Convolution 3x3, Pad "same", 24 Filter, Batch Normalization, Activation Leaky ReLU
4. Max-Pooling 2x2
5. Convolution 3x3, Pad "same", 16 Filter, Batch Normalization, Activation Leaky ReLU

6. Max-Pooling 2x2
7. Convolution 3x3, Pad "same", 32 Filter, Batch Normalization, Activation Leaky ReLU
8. Convolution 3x3, Pad "same", 40 Filter, Batch Normalization, Activation Leaky ReLU
9. Fully Connected 7x10x7

With these networks we can detect close robots reliably, even if they are lying as well as robots that are up to 5 meters away. The lower network boosted our ball detection as well in tricky circumstances, for example if the ball is not surrounded with green when fighting for the ball.

For next year we want to improve these networks by adding more objects. Currently we have good results with an image segmentation network detecting robots, balls and goals.

3.5 Automatic Camera Calibration

In order to determine the exact position of objects on the field, the robot must be able to project image coordinates from its cameras into the appropriate field coordinates using the position and orientation of the camera also termed CameraMatrix. For a precise transformation the exact position of the cameras must be known. This position is calculated using an inverse kinematic of the current joint angle sensor values and the robot dimensions. In theory this would be enough, but practically the joints are slightly different between each robot which results in small offsets in the body posture and also the cameras are not fixed tightly in the Nao's head. These small differences can result in big errors especially when calculating the position of far objects in the camera image. As these parameters also vary because of the joints wearing out, the camera's glue is getting warm and after repairs, they must be adjusted from time to time.

We calibrate the x and y (in the coordinate system of the NaoQi Framework) rotation offsets of both cameras and the body. Since this can be quite time consuming to do manually for each robot and it is not very intuitive, we implemented an automatic approach using the field line detection.

Our process starts by placing the robot on a fixed position on the field (e.g. in the middle of the center circle looking to the side line in our case) and we then compare the detected horizontal side line and vertical middle line to the expected positions using the CameraMatrix. We calculate the total error by measuring the distance and angle difference between the expected and detected field lines. To minimize small fluctuations in the line detection, we aggregate the lines of multiple frames and only use their median. Our optimization algorithm iterates over all possible rotation offsets for the camera calibration between $-12\,\text{deg}$ and $+12\,\text{deg}$ with decreasing step sizes and tries to find the parameters with minimal total error.

The robot moves its head to four predefined positions and records the found lines on the field. First we optimize the body rotation using the upper camera looking straight towards the T-cross. We set the upper camera calibration to zero and start the optimization algorithm for the body rotation to minimize the error. After that, we have to

determine which part of the previously calculated body rotation correction belongs to the upper camera. Therefore, we move the head 45 deg to the left and 45 deg to the right and try to minimize the error using the same algorithm again, but modifying the values for the upper camera calibration this time and keeping the sum of body and upper camera calibration constant. This ensures that the minimal error from the center position of the head is still the same. Consequently we move the head upwards, so that the robot is able to see the T-cross with its lower camera. Using this position we also minimize the total error by optimizing the calibration values for the lower camera correction. As a result both cameras and the body are fully calibrated and a reliable CameraMatrix should be provided now. If at any time the calculated calibration parameters still result in too high errors, the calibration process starts from the beginning. An incorrect detection of the field lines is the most common reason for that.

To make the calibration process even more convenient, we implemented a simple calibration behavior, which can be controlled via the chest button and automates the whole calibration process. That allows us to calibrate one robot after another without need to connect a notebook every time. The generated camera calibrations can be downloaded from the robots using the deploy tool BUSH afterwards.

3.6 Modeling

Currently our modeling consist of several modules which use our perception to create a robust filtered world map. In this section we give a short description of the models that we improved during the last year.

Obstacle Model

Our obstacle model, termed *RobotMap*, is using the percepts from our visions system to create a robust model of all robots on the field. In Addition to the percepts, we add obstacles detected by the foot bumpers in front of the robots as well as our team mate's positions from our team communication. Compared to last year we are working on using this information for our localization.

Ball Model

We simultaniously have three different ball models at any time. Each ball model is a multi hypothesis kalman filter to ensure a stable model even with false positive ball percepts. The first one is the local ball model, which is only updated through local percepts. The second model is a remote ball model, which is only udpated with remote percepts sent by team mates. And finally our third model is the team ball model which uses the best of those two models depending on the quality and currency of the percepts.

Our behavior adds a third layer to these models by choosing the prefered model depending on the tactical situation. Due to the lack of feedback into a robot's own localization there are situations where the local model is still to be prefered. 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 team ball 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.

Chapter 4

Behavior

Nao Devils as well as previous Dortmund teams implemented behavior mostly by utilising state machines such as XABSL (*Extensible Agent Behavior Specification Language*). XABSL was developed in its original form in 2004 using XML syntax [18] 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 3.2).

Until 2018, our behavior used static roles with static jobs to ensure a stable assignment. This however had the disatvantage that we could not dynamically react to different scenarios that the robots encounter during a game. Hence for 2019, we changed our assignment to a dynamic one, where we first determine what is needed on the field given the amount of available players. After that we assign our players depending on their current position to avoid path crossings, prioritizing the player that kicks the ball. With this approach we can decide our general strategy on a high level before letting the robots sort out how to best implement it. Since all robots have to use the same decision we decide to use the lowest available player number as the one dictating the strategy. This way we are not dependent on every player calculating the same strategy at all times for a stable decision. The only exception for this is the ball chaser (or striker), whose assignment can change faster - if we would have to wait for the next package to confirm the decision, it could be too late already. Therefore this assignment can be overwritten locally on each robot.

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. Hence, the remainder of this chapter is structured as follows: Section 4.1 describes our new kick selection for this year and section 4.2 describes a path planning approach to augment

CABSL.

4.1 Kick Selection

Kick Selection

Since the German Open 2018, we have tried to tackle one of our weaknesses in a one-on-one situation, namely speed and stability of decisions. We added a new kick selection to our walking engine to solve this problem.

Since RoboCup 2018, our behavior does not select the best approach and kick for the ball in our current implementation, but rather the walking engine itself does this. The input for our walking engine is a direction and strength of the kick and the walking engine creates a pattern of steps leading up to the optimal kick for this input. Usually this results in from one up to ten steps which are put in the preview of our walking engine and are then executed. Through this we achieved a fast and smooth transition from approaching the ball to the actual kick execution.

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. Since 2014, a much simpler and more efficient approach has been implemented. The path starts with just two way-

points, 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.

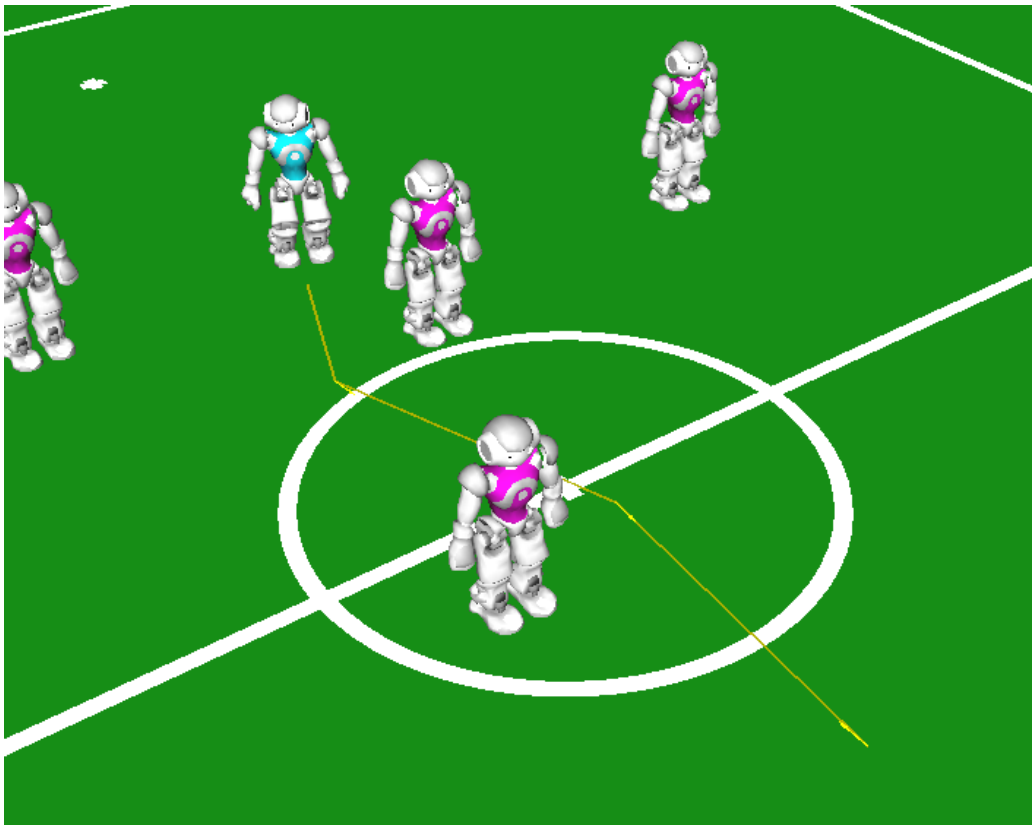


Figure 4.1: Example of a path around obstacles.

Chapter 5

Infrastructure

5.1 Time synchronization

For reliable communication between robots and consistent timestamps, it is important that the clock on all robots is set correctly. Furthermore, the algorithm used during the whistle direction challenge described earlier is highly dependent on an accurate clock.

During the last few years, we synchronized the clocks of the robots during our deploy script using ntpdate[1] to a given NTP server. However, this has the drawback that it slows down the deploy process, it is quite inaccurate and it might fail if the robot turns off during the game for some reason and its battery-backed RTC clock looses the time.

Therefore, we installed chrony[2] on the robot that runs continuously as a daemon in the background and keeps the time in sync with a given NTP server if connected to LAN. That has the advantage that chrony monitors the time difference over a longer period of time and also compensates the clock drift of the Nao. That improves the clock accuracy without a NTP server connection afterwards. Nevertheless, a complete time loss for some reason is still not handled by this approach. That is why we additionally implemented a simple NTP-like time synchronization via team communication during the game as a backup if the time offset to the other robots gets too big.

[1] `https://linux.die.net/man/8/ntpdate`
[2] `https://chrony.tuxfamily.org/`

Chapter 6

Challenges

In the following chapter, we describe the development for the mixed team competition and the directional whistle challenge we participated in at the RoboCup 2019.

6.1 Mixed Team Competition

The mixed team competition was newly introduced at RoboCup 2017 where two teams with three robots each played 6v6 games on a standard SPL field against other mixed teams. This year we started a new joint team with the team HULKs - team Devil SMASH. To ensure synchronized behavior we used the mixed team competition to update our time synchronization between our robots by creating our own version of NTP 5.1. This ensures a reliable team communication even when our default method - offline NTP sync over cable - does not work.

6.1.1 Behavior Adjustments

This year's focus with the HULKs was on fusing our role decisions and assignments into a single working behavior, despite using a different approach. We have been successful in the end but could not complete all features, such as coordinated team movements, as desired.

6.2 Directional Whistle Challenge

For this challenge we used an approach based on detection of time differences between the robots. This is similar to the approach described in [19] or the method used with GPS and GSM. Using this lateral approach we gained the highest score for far away signals but had a bad result with close whistles on the own field.

The whistle challenge taught us two things: First, we need to combine an angular and a lateral approach to receive the best result and second, that using meta information, like possible sane referee positions, will be important.

In the end, after looking at the challenges result, no team was able to clearly distinguish between a referee on the own or on another field. Nevertheless, combining several

approaches, we hope to tackle this problem for the next RoboCup.

6.3 Results

We were able to get the second place in the Direction Whistle Challenge and the third place in the Mixed-Team-Challenge. We will build upon the whistle recognition challenge to prepare for the coming years.

Chapter 7

Conclusion and Outlook

The *Nao Devils* are 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 2019.

A lot of effort has been put into fixing and simplifying our walking engine code as well as the introduction of the new KeyFrameEngine. We did a full rewrite of our behavior in the two months before the RoboCup as well, preparing us to adding new features in the coming years. Due to the new hardware in the V6, we also tried and tested a lot of neural network options for object detection and were able to successfully employ them in real games while maintaining the full 30fps.

For RoboCup 2020, beside integrating the new soccer rule changes, we plan to improve our side walking speed as this was one of our main weaknesses in a one-on-one fight for the ball. We will increase the use of neural networks for tasks like object detection and whistle recognition and hopefully start to develop new behavior features allowing us to use more advanced teamplay. Another big change for RoboCup 2020 will be the use of modern C++ features and libraries to make better use of the V6 multicore system.

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] 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

[7] Vukobratovic, M., Borovac, B.: Zero-moment point – Thirty five years of its life. International Journal of Humanoid Robotics **1** (2004) 157–173

[8] 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

[9] 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

[10] Urbann, O., Tasse, S.: Observer based biped walking control, a sensor fusion approach. Autonomous Robots (2013) 1–13

[11] Urbann, O., Schwarz, I., Hofmann, M.: Flexible linear inverted pendulum model for cost-effective biped robots. In: Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on, IEEE (2015) 128–131

[12] Kaneko, K., Kanehiro, F., Kajita, S., Hirukawa, H., Kawasaki, T., Hirata, M., Akachi, K., Isozumi, T.: Humanoid Robot HRP-2. ICRA2004 (2004) 1083–1090

[13] Hofmann, M., Moos, A., Rensen, F., Schwarz, I., Urbann, O.: Playing soccer outdoors with humanoid robots. In: Proceedings of the 11th Workshop on Humanoid Soccer Robots in conjunction with the 2016 IEEE-RAS International Conference on Humanoid Robots. (2016)

[14] Hebbel, M., Kruse, M., Nisticò, W., Wege, J.: Microsoft hellhounds 2007. In: RoboCup 2007: Robot Soccer World Cup XI Preproceedings, RoboCup Federation (2007)

[15] Schwarz, I., Hofmann, M., Urbann, O., Tasse, S.: A robust and calibration-free vision system for humanoid soccer robots. In: Proceedings RoboCup 2015 International Symposium, Hefei, China (2016)

[16] Urbann, O., Camphausen, S., Moos, A., Schwarz, I., Kerner, S., Otten, M.: A c code generator for fast inference and simple deployment of convolutional neural networks on resource constrained systems (2020)

[17] Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2016) 779–788

[18] Lï¿½tzsch, M., Bach, J., Burkhard, H.D., Jï¿½ngel, 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

[19] Bucher, R., Misra, D.: A synthesizable vhdl model of the exact solution for three-dimensional hyperbolic positioning system. Vlsi Design **15** (2002) 507–520