Team Research Report 2016

Nicolas Riebesel, Arne Hasselbring, Lasse Peters, and Finn Poppinga

Hamburg University of Technology

1 Introduction

1.1 The Team

The SPL[1] team HULKs formed in April 2013 and thus is one of the newer teams in the SPL. The team is organized as the association RobotING@TUHH e.V. and currently consists of about 25 members, most of which are graduate students. We have participated in the RoboCup 2014, 2015 and 2016, the RoboCup German Open 2014 and 2015 and the RoboCup European Open 2016.

Prior to this year's competitions we have always been eliminated in the preliminary games. In contrast, this year we reached the quarter finals at RoboCup European Open and the second intermediate round at RoboCup World Championship. This shows that we made great progress in our playing ability and adapted to the rule changes regarding the black and white ball [10]. Additionally, we reached the second place at the Technical Challenges due to our superior performance in the No Wi-Fi Challenge [9, 11].

1.2 Structure of this Document

The success at RoboCup 2016 has been made possible due to many hours of research and work with the robots. This document is written to reflect most of the development made for RoboCup and fulfill the requirements for prequalification for next year's competition. In addition, since we did not release our code yet, there is few publicly available documentation about our software. Therefore we intend to describe a few concepts and parts of our codebase that have been developed earlier than 2016 and have not been covered yet by one of our publications [7, 8].

Another purpose of this document is that it might serve as team-internal reference manual. This is important because although our code is reasonably commented, it is helpful to have a single source of documentation that can be handed to new team members.

The remainder of this paper is structured as follows: Chapter 2 introduces the basic foundations on which our software is built. In the next chapters we present the methods we developed for solving the tasks in robot soccer. Namely chapter 3 describes the state estimation and decision process of the robot, chapter 4 is about sensor processing and motion generation and chapter 5 gives an explanation of our computer vision algorithms.

[1] Standard Platform League

2 Framework

This chapter provides an overview of the parts of our software that are not directly related to robot soccer specific tasks.

2.1 General Structure

History In the early days of our organization, the team has been split into three development-related divisions; namely *brain*, *vision* and *motion*. These divisions corresponded to separated code projects that were connected by a library called *tuhhSDK*. The *tuhhSDK* was loaded into the naoqi process and connected to the cycle slots of naoqi's DCM[2]. It then loaded the brain, vision and motion modules as shared libraries into the same process. Each of the three modules had a cycle method that would be executed in multiples of the DCM cycle time (10ms). Furthermore it provided messaging between the modules via event queues, e.g. brain could send a command to motion to walk to a given pose on the field or vision could send a result that it has detected the ball.

Interprocess A major change in our software architecture has been completed in late 2015: Previously all our code ran as several shared libraries loaded into naoqi. Throughout 2015 we worked on detaching our high-level code from naoqi so that only a small library called *libtuhhALModule* would remain inside the naoqi process. This library exports the sensor readings and some methods from the DCM API via a shared memory block. Its counterpart is the process *tuhhObserve* which now runs the brain, vision and motion modules. By now they are not even compiled as shared libraries anymore but are directly compiled into one executable.

2.2 Build Process

We use cmake as makefile generator. From the user's perspective it is encapsulated by a few scripts. It is possible to build our code from docker[3] which greatly simplifies the system setup process and enables cross platform usage.

Since the cross compiler provided by Soft Bank Robotics does not have full C++11 support, we have a custom toolchain for compiling *tuhhObserve*. This toolchain also contains newer versions of some libraries and a clang/llvm compiler. For *libtuhhALModule*, though, we still use the Soft Bank Robotics toolchain.

Another recent development is that we removed qibuild from our build process since it always added an extra step to the system setup process and added a lot of unnecessary complexity to the build system.

[2] Device Communication Manager
[3] https://www.docker.com

2.3 Hardware Access

Following the changes named in chapter 2.1, we introduced a hardware abstraction layer that encapsulates the access to sensors, actuators, camera images and audio data. This is realized in form of a purely virtual class called `NaoInterface`. There are four implementations of this interface:

ObserveInterface connects to a shared memory provided by *libtuhhALModule* for implementing the sensor and actuator related methods. It uses Video4Linux to obtain camera images and control the camera settings. Audio samples are retrieved and played back via PortAudio. This is the interface that is used when compiling for a NAO.

ReplayInterface provides sensor data and images that are loaded from a log file. This is mostly used for testing and evaluation of our computer vision algorithms. The actuator manipulation methods are no-ops.

QtWebsocketInterface is used to connect with another device that generates sensor data for testing without a NAO at hand. The primary use-case is to connect a smartphone that sends its accelerometer and gyroscope data to test our orientation filter (see chapter 4.3) directly in our codebase.

WebotsInterface implements a controller for the robot simulator Webots.

2.4 Debugging and Configuration

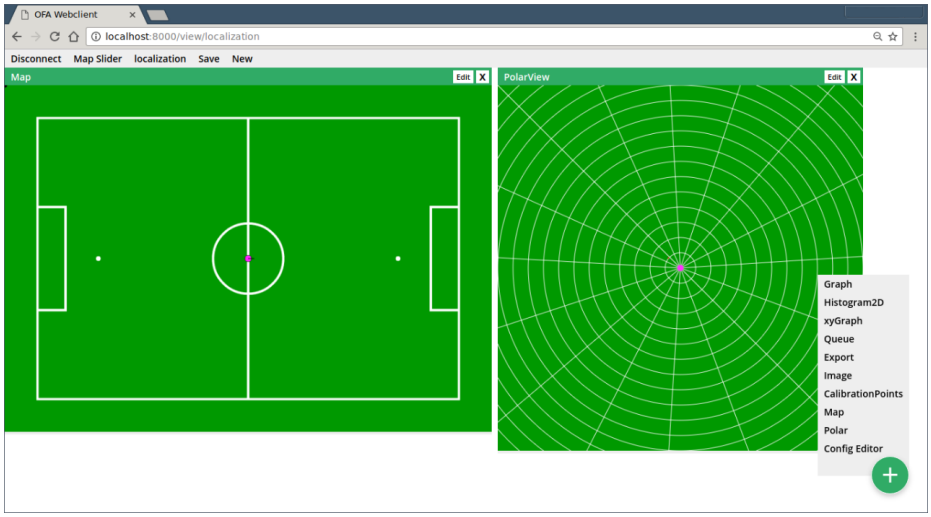

Fig. 1. This is an example view of our debugging application running in Chrome.

Our framework features a variety of debugging and configuration features. It is possible to export variables (including images) with an associated name so

that they can be sent to a PC or written to a log file. On the PC side there is a Node.js application that connects to the NAO via a TCP socket and provides a web interface. You can see the application in action in figure 1. Data can be displayed in several types of views:

Graph views are plots of a numeric value over time. Multiple variables can be plotted color-coded in the same panel. The value range can be determined automatically and the number of samples can be adjusted.
Histogram2D is a heatmap of a two-dimensional histogram.
xyGraph displays a vector of two numbers as a point in a 2D coordinate system. As the vectors are buffered, their trajectory can be observed, too.
Queue can show textual information such as log prints.
Export creates CSV files with sequences of numbers that can be used for later analysis.
Image views display JPEG compressed images that are sent by the NAO.
CalibrationPoints is an extension of an image view in which it is possible to click points that can be sent to the robot. This is intended for camera calibration purposes.
Map is a view to display the pose of the robot on the field as well as particles of the particle filter including relative observations to the particles. Additionally the robot's target and the ball can be shown.
Polar views are used to display robot relative positions such as projected ball, line or goal post observations.

A specific setup of views can be stored and loaded as file to allow for fast access to often used view combinations.

Besides getting information from the NAO for debugging and evaluation purposes there are also several parameters that influence our software. We have a configuration system that loads parameters from JSON files when given the name of a class. It will look in multiple directories that are e.g. specific to a NAO head, NAO body or a location such as RoboCup 2016 SPL_A. Values that are given in more specific files will override values from generic files. It is also possible to change parameters at runtime via the aforementioned web based debug tool. Furthermore it is possible to map single parameters to sliders of a midi-controller. Receiving new values can cause a callback to be run so that precalculations based on these parameters can be redone.

2.5 Module Architecture

The largest part of the algorithms for robot control is organized as independent units called modules. The connection of these modules can be modeled as bipartite data flow graph made of modules and the data types they exchange. Data types are stored in a database for each module manager[4]. The relation of modules and data types can be either that a module produces a data type or that it depends

[4] Brain, vision and motion implement the module manager interface.

on it. This is realized by two template classes `Dependency` and `Production` of which a module has member variables for the data types it wants to access.

Each module must have a `cycle` method that executes the code that has to be run in every frame. When a whole new cycle starts all data types in the database are reset to a defined state. The order of the cycles is determined at runtime by topological sorting to guarantee that the dependencies of all modules have been produced before their execution.

In general, the semantics of the module architecture are similar to the one presented in [12], but the implementation is completely macro-free and instead based on templates. This has the advantage that no design specific language is introduced and every person capabale of reading C++ can easily read the code at the cost of a bit more verbose declarations of dependencies and productions.

3 Brain

3.1 Behavior

Our behavior is currently based on the state machine framework `boost::msm`. We implement a static strategy where each robot is assigned a fixed role and position. There are state machines for the basic game controller states (initial, ready, penalized etc.) and dedicated ones for each role in the playing state. A construct called `SMSwitcher` switches between those state machines depending on the game state since compilation times for `boost::msm` become very high when creating too large state machines.

Furthermore, the head is controlled by separate state machines that can be set by the body state machines. There are two head state machines currently in use: one that keeps the head idle and another one that tracks the ball.

The behavior is currently facing a complete redesign because `boost::msm` is unpractical for large hierarchical strategies. The current idea for the rewrite is to use a purely functional structure where a root function must return a command for the robot that should be executed. Hierarchy is implemented by calling other functions whose action can be either returned or combined with an action for another body part. We currently plan to focus mostly on developing a new behavior since it is the next step for more competitive soccer playing.

3.2 Robot Localization

In an initial attempt in 2013 to target the problem of robot localization a multimodal Kalman filter was implemented to determine the robot's location on the field. Position hypotheses were generated from field line perceptions, which then where fed into the filter to track and eventually merge them. Due to the fact that this method depended on sufficient line features to generate a hypothesis from a single picture it only worked well on the small field we tested it on at the time. Despite these efforts to establish a pose estimation, we still suffered from chronic mislocalization in the following years and thus started a new approach in the season of 2016.

The current robot localization is implemented by a *Brain* module called `PositionKnowledge`. It produces `RobotPosition` data based on field line data, odometry data and goal post inputs. To estimate the robots position based on past robot state, a particle filter with multiple stages is used.

A particle filter is a stochastic state estimation method based on random sampling of the state space. In the case of the robot position estimation problem, the state space consists of three variables: the robot's x and y coordinate and its rotation about the z-axis, respectively. A randomly generated sample of the state space is called *particle*. By comparing each particle with the measurements acquired during the game, a weight is calculated per particle. Each step, some particles are replaced with newly generated random samples. The convergence from the particle filter's estimate to the actual robot pose results from the fact, that the probability of a particle to be deleted after one time step is inversely proportional to its weight. New particles are more likely to be generated near existing particles with higher weight.

To improve the quality of the position estimation, some special enhancements have been made to the general algorithm of particle filtering. Each particle belongs to a particle cluster, which tracks the origin of the particle (e.g. it has been generated as part of a post-penalty pose). This allows for a position estimation based on local weight maxima in most cases, without the need for a dedicated and time consuming clustering algorithm as e.g. *k-Means*.

The particle filter's cycle consists of the following stages:

State Update Particles are generated, depending on the current `GameState`, e.g. if the robot was manually placed in the *SET* game state, particles are generated based on the manual placement positions as defined in the rules [10].

Odometry All particles are moved according to the robot's displacement as measured by the `SensorFusion` and the `WalkingEngine`.

Position Estimation The position estimate is calculated using the mean of the particle cluster with the highest total weight.

Particle Weighting Each particle is weighted based on the measured field line data and goal posts position.

Resampling The sum of the particle weights is normalized. A fraction of the particles is replaced by new particles, generated from sensor data (*sensor resetting*).

3.3 Whistle Detection

A reliable detection of the whistle yields an important time advantage after each kickoff. The whistle detection used in 2016 is signal power based. A mono-mix of all microphones of the NAO is analyzed in three steps. First the signal is sliced into buffers of equal length $n_{\text{buffer}} = 4096$ samples at a sampling rate of $f_s = 44\,100\,\text{Hz}$. Each buffer signal is filtered with a variable position and bandwidth bandpass filter to separate the audio signal into a whistle band and a stop band. In both bands, the power is determined using autocorrelation. If the quotient of whistle band power and stop band power is greater than a threshold, the whistle is marked as detected.

3.4 No Wi-Fi Challenge

One of 2016's technical challenges was the *No Wi-Fi Challenge*. In order to get points in this challenge, two robots needed to demonstrate the ability to communicate over varying distances, using no Wi-Fi or ethernet technology.

The teams showed different approaches to this challenge that can be classified into audio-based methods and infra-red based methods. All teams, except the Austrian Kangaroos, used audio-based means of communications i. e., playing a sound at the transmitting robot and recording it using the robot's microphones at the receiving robot.

As the environment of a *RoboCup* event was assumed to be very noisy in the audio band accessible using the NAO hardware [2], a very robust modulation method had to be used. The transmission over a 6 m aerial channel ruled out modulation methods commonly used by dial-up modems. Therefore a transmission method based on the work of Lee et al. [5] was evaluated and implemented in preparation of *RoboCup* 2016 [6].

The method proposed by Lee is based on two chirp signals an up-chirp x_u and a down-chirp x_d, respectively (see (1)).

$$x\left(t\right) = \cos\left(2\pi f_1 t + \mu t^2\right) \tag{1}$$

with

$$\mu = 2\pi \frac{(f_2 - f_1)}{N}$$

The key benefit of these signals is the fact that they are nearly orthogonal with respect to the standard correlation function, as can be seen in figure 2. This property enables one to encode a message using the chirp-signals as symbols. To ensure a low byte error rate, the communication is split into packets of 16 bit length. Each packet is announced with a dedicated preamble signal, which is an up-chirp with five times the symbol duration (see figure 3). This method is called CBOK[5].

[5] Chirp Binary Orthogonal Keying

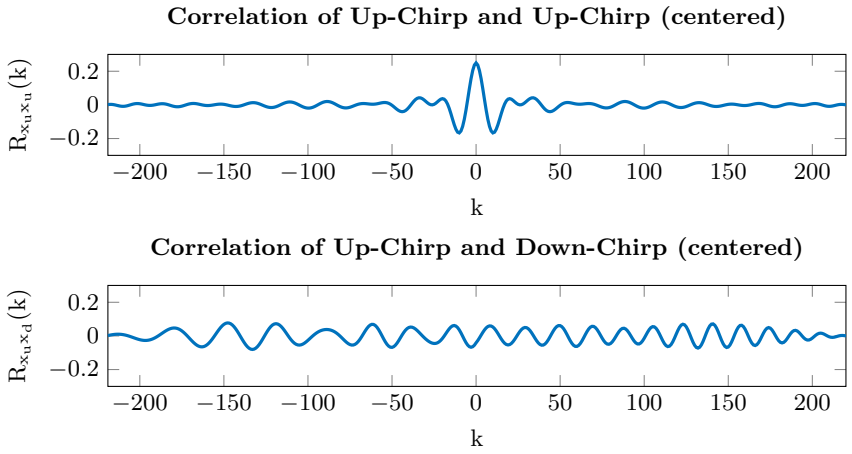

Fig. 2. Auto- and cross-correlation of the different chirp signals..

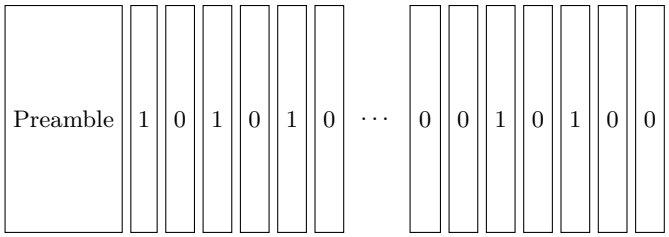

Fig. 3. Layout of the packet in time domain. Each chirp is followed by a guard interval. The packet is announced by a preamble.

The use of CBOK allowed for relatively high data transmission rates of up to $12\,\mathrm{Byte\,s^{-1}}$ in the competition and $30\,\mathrm{Byte\,s^{-1}}$ in a sound-proof environment [6]. As a result, the *HULKs* won the No Wi-Fi Challenge by being able to transmit the most data in 15 seconds.

4 Motion

4.1 Walking

Our walking engine was initially implemented in 2014 as part of the preparation for the German Open and is based upon [4].

The model used to predict the CoM[6] trajectory consists of two submodels, one for each x- and y-direction.

y-direction: A LIPM[7] is used to model the coronal CoM trajectory and thus sets the timing for the support foot changes. Since the LIPM is well documented [3] it won't be further described at this point.

x-direction: A spline approximated LIPM is used to model the sagittal CoM trajectory. The main idea of this approach is to fit a spline to meet the boundary conditions of one step phase and thus mimic the hyperbolic behavior of the LIPM. Due to the fact that the spline is also updated with the measured state, it can then be used to track the ZMP[8], simply by using it in the differential equation for the ZMP. This way the ZMP is tracked and – if necessarry – the gait is adjuted to keep the ZMP within the convex hull of the support polygon.

Before RoboCup 2016 Tests in previous competitions had shown that the described model was quite stable for the y-direction but led to soar up effects in x-direction. Especially the v3.3 NAOs suffered from an unstable gait due to backlash in the joints.

While this soar up would not cause the ZMP to leave the support polygon it still caused a collision of the foot with the ground due to the forward tilt of the robot. This would then immediatley cause an unstable gait that the walking engine was not capable of recovering from.

In a first symptomatic fix for this problem a dynamic step height was introduced, to adjust the the z-trajectory depending on the tilt of the robot. In addition to that, we virtually decreased the expansion of the support polygon by dynamically reducing the ZMP limit, as instability is detected based upon measurements of the angular velocity.

As a result of these changes we were able to deploy a stable gait, which – as tested at RoboCup 2016 – led to a significant reduction of fallen robots in comparison to previous events. Despite these improvements the walking speed was still insufficient to compete with the superior teams and – as tested during the outdoor challenge – was not capable of ensuring stability when walking on artificial turf as required for the SPL season 2017.

Current fields of research In order to improve the gait's stability and speed we started developing a new walking engine as we returned from Leipzig. In this walking engine, we stick to the LIPM for the coronal balance shift, as it proved to be a robust model within the last seasons. Yet we focus on testing different approaches to ensure sagittal stablilty, as well as a high walking speed. As part of these research efforts we tested different control strategies for the ankle joints, which yet seem to be a promising measure to support the walking

[6] Center of Mass
[7] Linear Inverted Pendulum Model
[8] Zero Moment Point

engine. Especially when walking on soft surfaces – as it is the case on artificial turf – the ankle controller enables the walking engine to force the CoM onto the planned trajectory.

One major problem of designing a gait is to effectively shift the load between the support legs and thus enable the robot to lift the swinging leg. Due to the fact that the robot has to force the ZMP instantly to the opposite foot some additional measures have to be taken to compensate for the backlash in the joints. Up until now we therefore apply a so called hip correction angle onto the hip roll. As an alternative strategy we recently tested adding a sinusoid swinging motion onto the z-trajectory of the support leg in order to more dynamically shift the load at support change time. While this turned out to work well and resulted in a smooth and natural gait, it led to a significantly higher engergy consumption and thus caused problems with overheated joints after a short period of time.

Apart from that we did some research on dynamically modifying the step length as a control strategy. For this purpose we developed an energy based model, from which we can calculate the step length to constrain the total amount of energy to meet a certain level and thus react to external energy contributions (e.g. collisions).

While up to now we only simulated this model, it is still to be implemented and tested on the NAO, to verify its function in physical reality.

4.2 Fall Manager

The fall manager is a safety feature that was introduced as part of the open challenge at RoboCup 2014. The purpose of this module is to catch the robot from falling when an unstable situation occurs. In order to do so, the fall manager tracks the IMU measurements as well as the angular velocity. If the angular velocity reaches a certain threshold and the robot is leaning in the same direction simultaneously, a catch motion is triggered.

As a basic catch motion we initially triggered a knee down motion in order to lower the CoM. This way the robot had to interrupt the walking engine but could recover from the disruption way faster than if it acutally fell down. While this prevented most robots from falling, unsuccessful triggering would lead to more severe hardware damage, caused by a higher probability of the robot to fall on its head, which often caused damaged CPU fans.

In order to prevent this issue, we introduced a seperate motion to be triggered by forward falling. This motion uses the arms to support the torso and thus enlarges the convex hull of the support polygon. Also from this pose the robot can recover faster than from a total fall down. While this method significantly decreased the number of broken fans it led to more severe damage of the arms in some cases. Future research will thus focus on developing a capture step strategy, which would allow the robot to recover even faster or – in best case – does not even noticeably interrupt the walking pattern.

4.3 Sensor Fusion

In late 2015 a new `SensorFusion` module was introduced. This should fuse gyroscope data and accelerometer data together, to form a better estimation of the current worldrotation of the robot. Our filter is heavily based on [1].

The `SensorFusion` module is implemented with a first order complementary filter. The internal representation uses quaternions and Euler angles at the moment.

While implementing the `SensorFusion` we hit a problem with the gimbal lock. So we are currently working on changing to work with axis angles instead of euler angles. In the future we want to calibrate the sensors to aid the odometry with the estimation of the translation of the robot.

5 Vision

Our vision is organized as a processing pipeline in which later modules can access data about the image that have been gathered by the previous modules.

5.1 Field Color Detection

The field color detection uses preconfigured upper bounds for the Y component and the sum of the squares of the Cb and twice the Cr channel of a color. A control algorithm adapts these thresholds so that they are a fixed amount above the mean of the previous field color. The field color controller allows a reliable detection of the field border in varying lighting conditions.

5.2 Image Segmentation

The image is segmented only along vertical scanlines. An edge detection algorithm determines when a region starts or ends. Subsequently, representative colors of the regions are determined and possibly classified as field regions depending on their color.

5.3 Field Border Detection

The field border detection uses the upper points of the first regions on each vertical scanline that are classified as field colored. Through these points a line is fitted with the RANSAC method. This chooses the line that is supported by most points. If enough points are left, i.e. not belonging to the first line, a second line is calculated with RANSAC. It is only added to the field border if it is approximately orthogonal to the first one.

The module also creates a second version of the image regions that excludes all regions that are above the field border or are field colored since those are the relevant regions for most other object detection algorithms.

5.4 Ball Detection

We developed two ball detections for this year where the first one was used at the RoboCup European Open and was deprecated by the second version that has been used since RoboCup World Championship.

The first approach was an extension of the ball detection that was written for the Realistic Ball Challenge at RoboCup 2015. It uses a greedy clustering approach for unclassified regions. The bounding boxes of these clusters form ball candidates which are then further classified using Haar-like features.

Our current ball detection is based on finding ball seed points that are sufficiently dark regions in between two light regions including some size relation checks. Ball candidates are then generated by tracing some radial scanlines until enough field color pixels are found or the scanline becomes too long. The size of this region is compared to the size that a ball would have at that position in the image and is possibly discarded. On each of the candidates a classifier is executed which uses an FFT of some sampled points in the region. It is planned to replace this classifier with another model in the future.

5.5 Line Detection

The line detection takes image regions that start with a rising edge and end with a falling edge and where the gradients of both edges point towards each other. For each of those regions its middle point is added to a set of line points.

After this, up to five RANSAC lines are fitted through the points, where each line is checked for holes and possibly split into multiple parts where some parts can be discarded.

References

1. Baluta, S.: A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications. (2009), http://www.starlino.com/imu_guide.html
2. Bergmann, F.: Acoustic Communication Between Two Robots Based on the NAO Robot System (2015)
3. Kajita, S., Kanehiro, F., Kaneko, K., Yokoi, K., Hirukawa, H.: The 3D Linear Inverted Pendulum Mode: A simple modeling for a biped walking pattern generation. In: Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on. vol. 1, pp. 239–246. IEEE (2001)
4. Kaufmann, S.: Implementierung und Analyse verschiedener Regelstrategien für dynamische Laufbewegungen humanoider Roboter auf Basis des NAO-Robotiksystems (2011)
5. Lee, H., Kim, T.H., Choi, J.W., Choi, S.: Chirp signal-based aerial acoustic communication for smart devices. In: 2015 IEEE Conference on Computer Communications (INFOCOM). pp. 2407–2415. IEEE (2015)
6. Poppinga, F.: Implementation and Evaluation of Audio Based Methods for Robust Inter-Robot Communication (2016)
7. Poppinga, F., Bergmann, F., Kaufmann, S.: HULKs - Team Research Report 2014 (2015)

8. Poppinga, F., Göttsch, P., Hasselbring, A., Linzer, F., Loth, P., Schattschneider, T., Schröder, E., Tretau, O.: Qualification Document for RoboCup 2016 (2015)
9. RoboCup Technical Committee: RoboCup Standard Platform League (NAO) results (2016), http://www.tzi.de/spl/bin/view/Website/Results2016
10. RoboCup Technical Committee: RoboCup Standard Platform League (NAO) rule book (2016), http://www.tzi.de/spl/pub/Website/Downloads/Rules2016.pdf
11. RoboCup Technical Committee: RoboCup Standard Platform League (NAO) technical challenges (2016), http://www.tzi.de/spl/pub/Website/Downloads/Challenges2016.pdf
12. Thomas Röfer and Tim Laue and Judith Müller and Michel Bartsch and Malte Jonas Batram and Arne Böckmann and Martin Böschen and Martin Kroker and Florian Maaß and Thomas Münder and Marcel Steinbeck and Andreas Stolpmann and Simon Taddiken and Alexis Tsogias and Felix Wenk: B-Human Team Report and Code Release 2013 (2013), http://www.b-human.de/downloads/publications/2013/CodeRelease2013.pdf

All links were last followed on Thursday 1st December, 2016.