Team Research Report

Nao-Team HTWK

January 2018

Contents

1 Team members and affiliations

Rico Tilgner, M. Sc., HTWK Leipzig
Thomas Reinhardt, M. Sc., HTWK Leipzig
Tobias Kalbitz, M. Sc., HTWK Leipzig
Stefan Seering, B. Sc., HTWK Leipzig
Michael Wünsch, M. Sc., HTWK Leipzig
Jörg Schließer, B. Sc., HTWK Leipzig
Florian Mewes, M. Sc., HTWK Leipzig
Sascha Haßler, Undergrad, HTWK Leipzig
Anne Wissing, B. Sc., HTWK Leipzig
Tobias Jagla, M. Sc., HTWK Leipzig
Kai Dawidowski, Undergrad, HTWK Leipzig
Marcel Göbe, B. Sc., HTWK Leipzig
Philipp Freick, M. Sc., HTWK Leipzig
Stephan Bischoff, B. Sc., HTWK Leipzig
Tom Burke, M. Sc., HTWK Leipzig
Samuel Eckermann, M. Sc., HTWK Leipzig
Daniel Weiß, Undergrad, HTWK Leipzig

2 Notable work and fields of interest

2.1 Team Strategy

Our full 2016 team strategy is available as Open Source at
https://github.com/NaoHTWK/HTWKStrategy.
It contains the strategy used during the 2016 competition as well as a debugging tool that allows for easy evaluation.

The advanced development within the basic soccer regions (for example ball skills and coordination) allows to make a new step in the SPL to create a human-like soccer gameplay. The use of a static teamstrategy gives a constant gameplay, which is suitable for testing and improving basic soccer skills. But to hold a competitive ability and to reach a manlike soccer gameplay, a new approach is needed. Each player must be able to make a decision for a optimal position to occupy depending on the current game situation. Additionally a player has to react and interact with all other players on the field. To realize such a behavior only a dynamic teamstrategy can be a solution. The graduation work in [3] (language: German) shows the development of the above described dynamic teamstrategy. The problem of dynamics (to find an optimal position) is considered as an "optimization problem" and is solved by an optimizer and an evaluation function. The main objective of the teamstrategy is the development of a defensive and offensive position finding behavior.

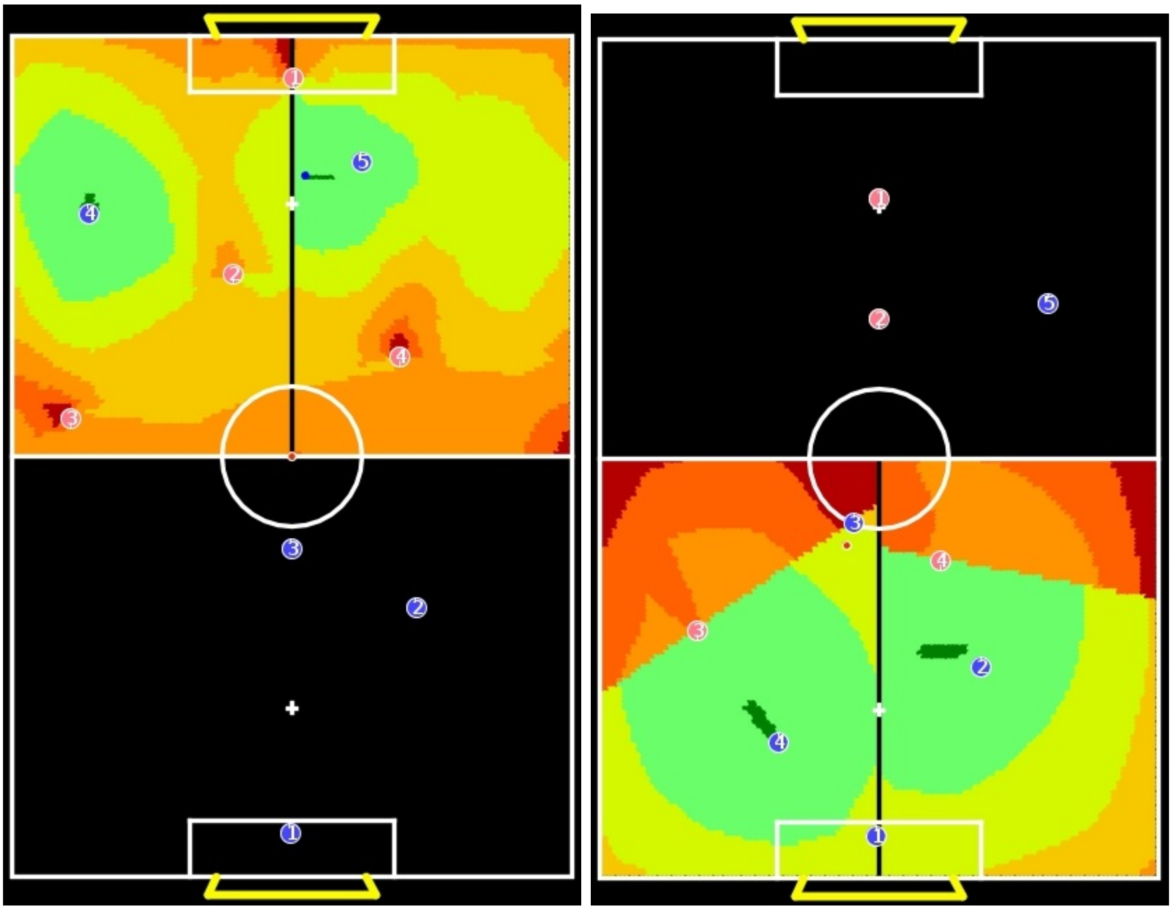

Figure 1: Offensive and Defensive Player Positioning

2.1.1 2017 Dribbling Modifications

During Robocup 2017 in Nagoya we realized, that our walking engine could not handle shooting motions on the new carpet well. This gave us a huge disadvantage because our team strategy was specialized on passing and shooting. To stay competitive, we decided at the end of the second round robin to develop a new team strategy, which is specialized on a dribbling only behaviour. This new dribbling team strategy uses one robot which shadows the attacking player and two seperate defined areas to defend.

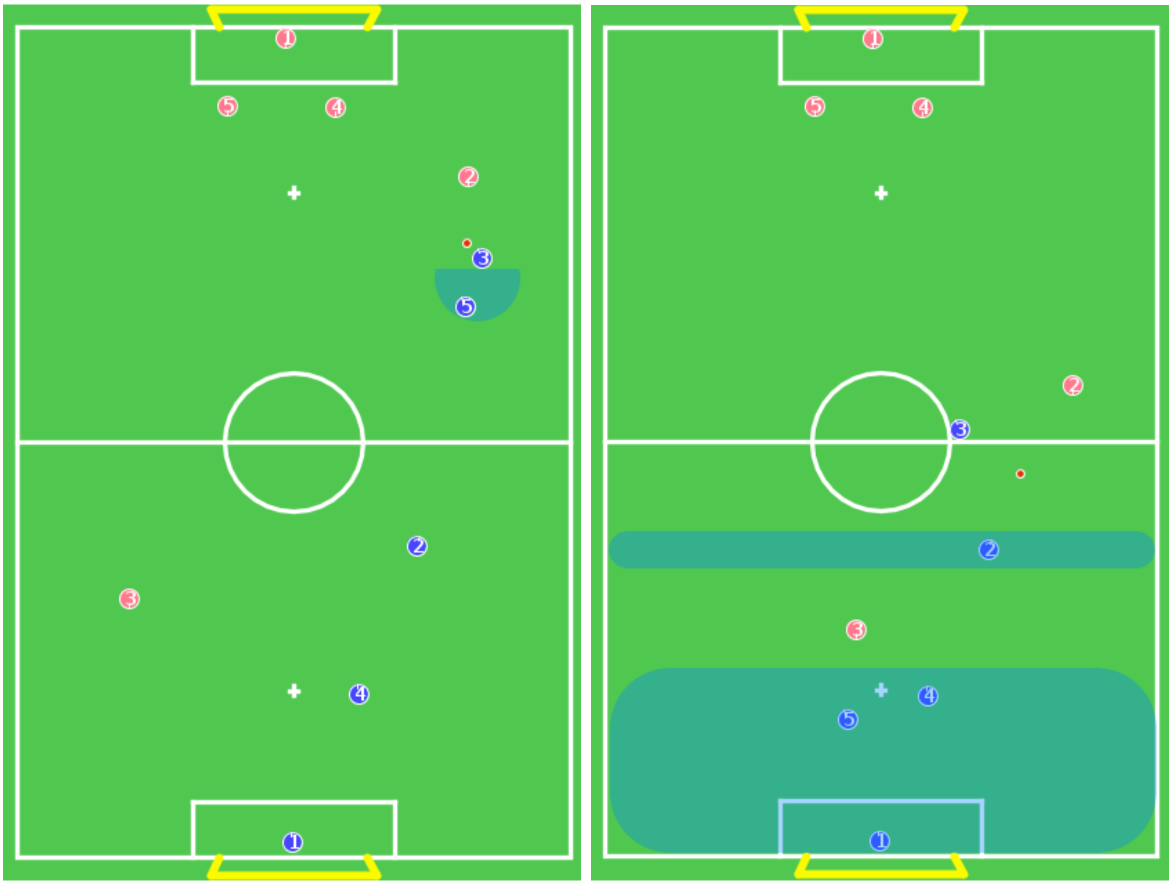

Figure 2: Range of shadow system, defined defender areas

The shadow system consists of one player, who is dribbling the ball (striker), and an additional player, who stays within a one meter range around the striker (shadow). In case, that the striker loses the ball - because of falling or another reason, the shadow will always be near to increase the chance of getting the ball back. The two defined defender areas are used to hinder long shot ranges and to spread team players over the whole team fieldside. To have as much players as possible to defend, the shadow is able to switch to the defending behavior. As used in the older team strategies, all players can switch between behaviors depending on the position of all team mates.

2.2 Vision

The identification of the field, field features and objects on it is an essential part of playing soccer. The biggest problem for most color-table based methods are the inability to cope with changing light conditions and the need to generate the color-table, which can be very time consuming. Changing lighting conditions (e.g. between daylight and artificial light as seen with the 2016 outdoor challenge) make it impossible to classify objects solely based on their color. Also the introduction of a black and white ball for the 2016 season prevents purely color-table based methods. Therefore, a real-time capable object detection with no need for calibration would be advantageous. By applying the knowledge of the objects' shapes we developed several specialized object detection algorithms that can handle changing light conditions and colors robustly without the need for prior calibration.

2.2.1 HTWKVision Library

Our full 2017 HTWKVision library is available as Open Source at
https://github.com/NaoHTWK/HTWKVision.

It contains many performance improvements and new or improved functionality compared to the 2015 version.
Please be aware that it does not contain a pretrained net for the ball detection but all tools necessary to train new nets based on labelled data. Teams interested in using our labelled images or nets can contact us at naohtwk@gmail.com.

2.2.1.1 Key Features

- simple to integrate
- no external dependencies
- fast 2x30fps on Nao
- no calibration needed
- good detection rates and accuracies
- simple demo program included

2.2.1.2 Field Color Detection We reworked our field color detector completely in 2015. It is now based on machine learning algorithms and is able to extract the correct field color completely automatically in a wide range of different and even adversarial lighting conditions. The algorithm uses a dynamic YCbCr-cube size estimation for green classification. Offline training using CMA-ES optimization has been performed using labeled color settings from 300 different images from SPL events between 2010 and 2017. The evaluation of the algorithm was performed on a set of 200 additional labelled images.

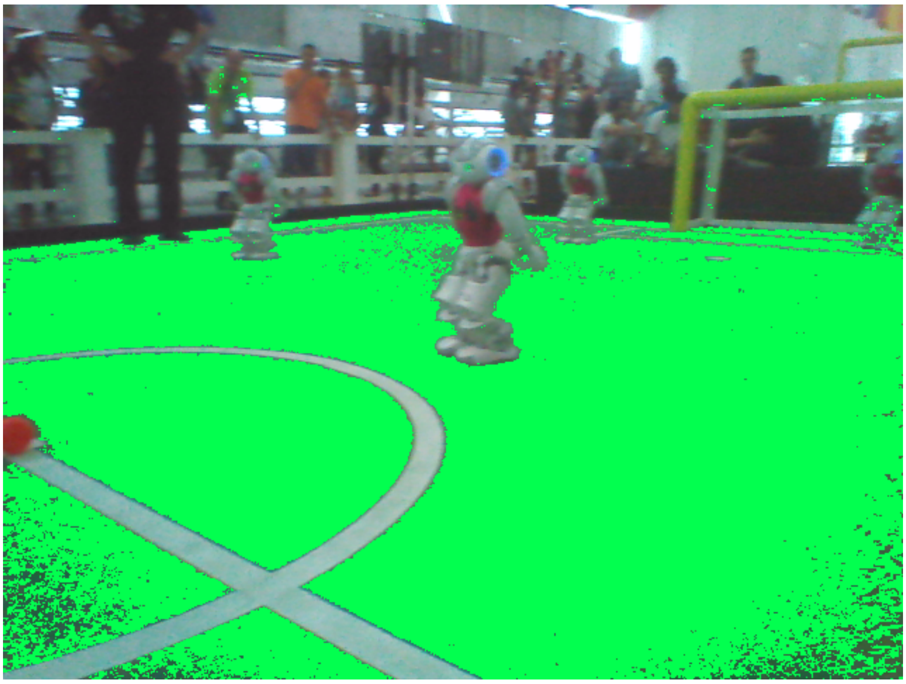

Figure 3: Automatically detected field color

2.2.1.3 Scanline Classification For several subsequent detection steps a scanline based image classification algorithm is used to detect white, green and unknown segments. Both vertical and horizontal scanlines are used to decompose the camera image.

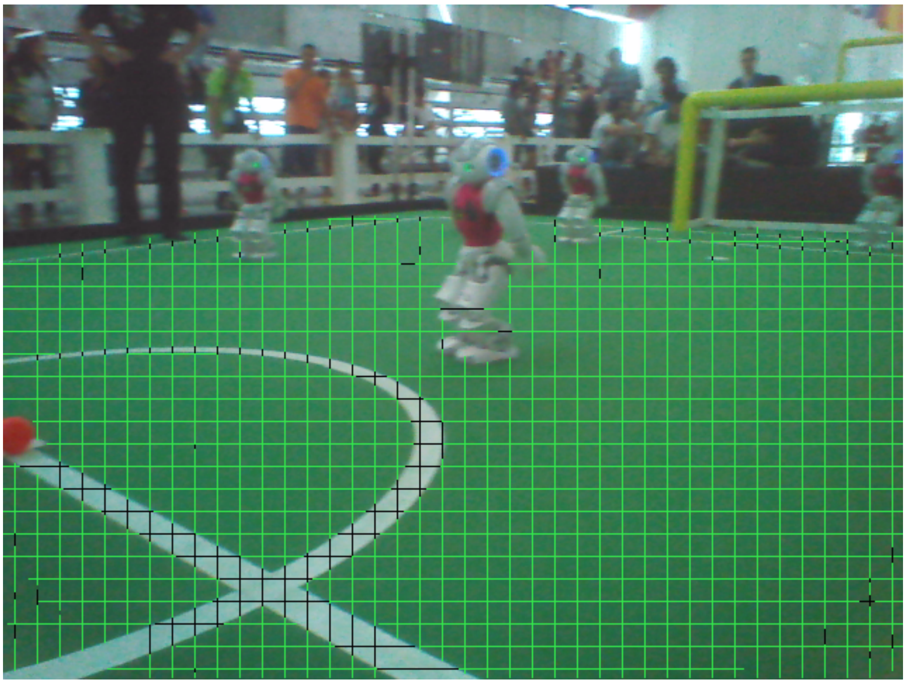

Figure 4: Scanline Classification

2.2.1.4 Field Border Detection The field border detection algorithm estimates the position of the upper field border by analyzing vertical scanlines and searching linear relation of their green to unknown class transitions. A model of two straight lines are matched using a variant of ransac algorithm.

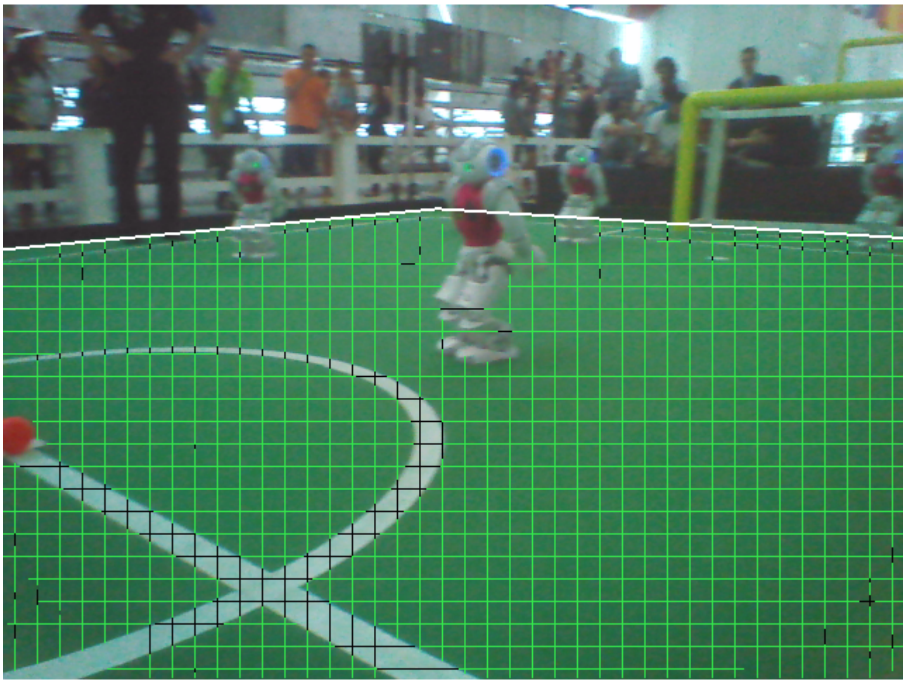

Figure 5: Field Border Detection (white line)

2.2.1.5 Line Detection The line detection algorithm uses the horizontal and vertical scanline classification results to group white region together under some constraints.

Figure 6: Line Detection

2.2.1.6 Ellipse Fitting of Center Circle We are using an ellipse fitting method to precisly detect the inner and outer edge of the center circle.

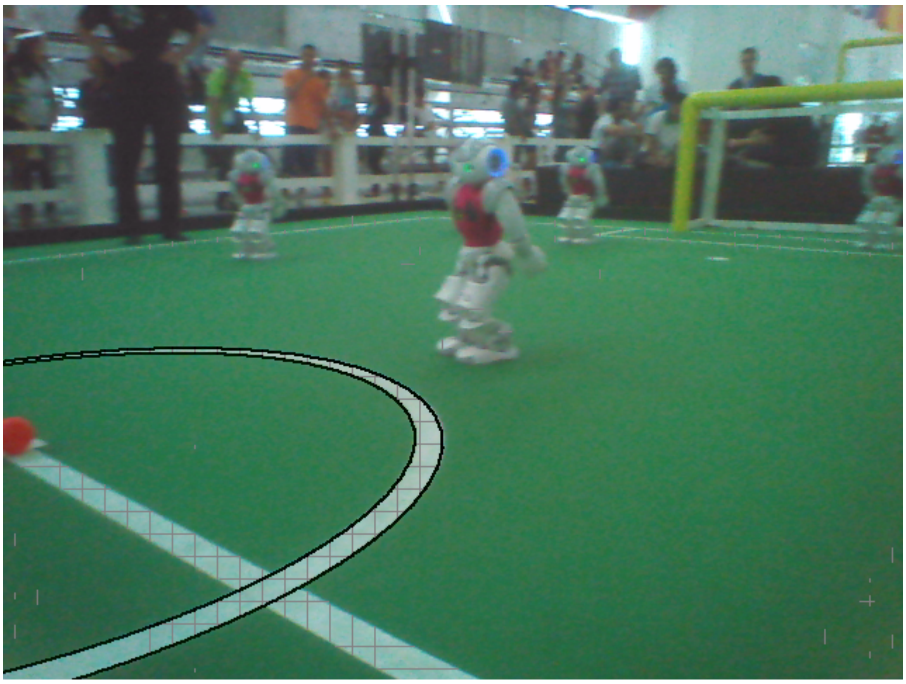

Figure 7: Center Circle Detection

2.2.1.7 Goal Detection The change in rules to use white goals instead of yellow goals necessitated a complete rethink of our goal detector.
The currently used goal detector first generates goal post candidates by analyzing vertical gradients just above and below the field border. We then extract 14 geometrical features from each of these candidates and evaluate them using a neural network. The features are independent of color and brightness which enables a calibration free detection in varying lighting conditions.

Figure 8: Goal Detection

2.2.1.8 Near Obstacle Detection This new module detects robots that are very close (less than 2m), even when we can only see their white feet in the lower camera image. It generates a relatively simple model for obstacle detection using pixel groups with high variation and differences w.r.t. the field color.

2.2.2 Black and White Ball Detection

As a development in difficulty the SPL changed the orange streethockey ball that has been used since 2010 to a ball with a black and white classic soccer ball pattern.
Our 2016 ball detection algorithm uses 2 phases: hypotheses generation using an integral image and hypotheses classification using a deep convolutional neural network. This achieves a very high hitrate and precision while being runtime efficient enough to run in real-time on the Nao.

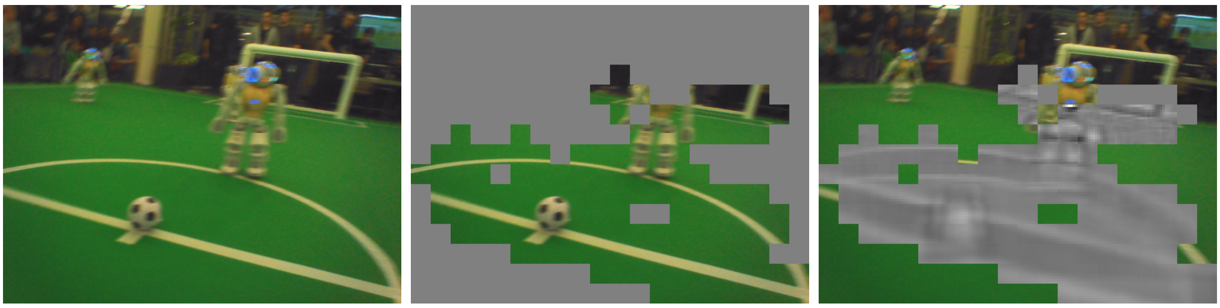

Figure 9: Hypotheses Generation, original image (left), low contrast / above field border regions (center, grey), CB-channel difference within high contrast regions (right, grey)

2.2.2.1 Hypotheses Generation The hypotheses generation excludes regions which are above the field limitations or with low contrast characteristics. For each remaining block in the input image it then calculates the difference between the inner and outer regions of the object (using CB-Channel) by means of the estimated ball size. As a characteristic of the CB-Channel, black and white colored pixels have a higher values than green colored pixels. That means objects with the size of the ball have

a higher result value in this calculation.
Local maxima from this calculation are going to be used as ball hypotheses.

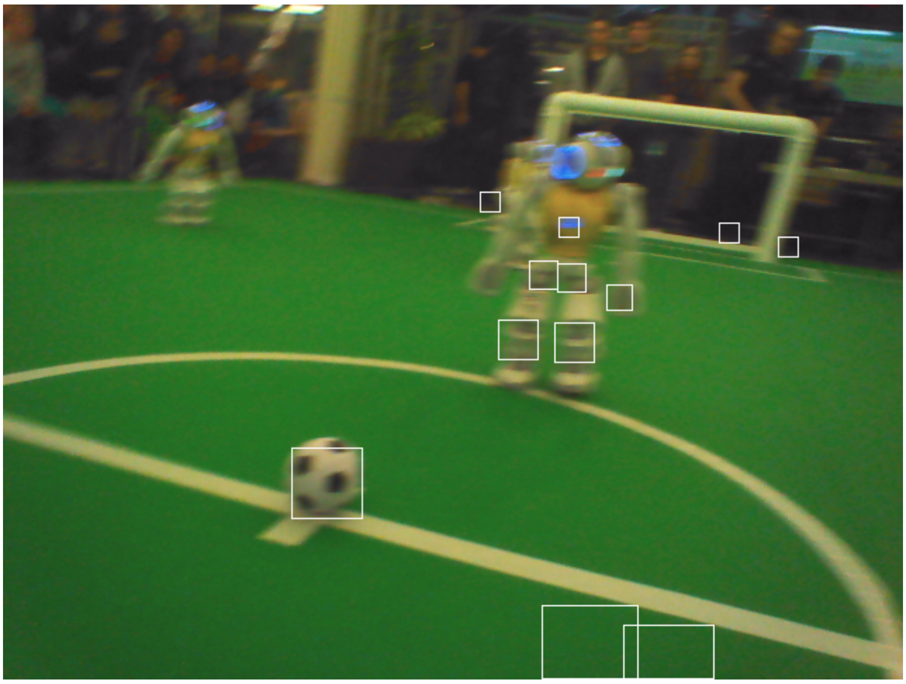

Figure 10: Ball Hypotheses

2.2.2.2 Hypotheses Classification The second part of the ball detection is the classification of the generated hypotheses with a deep convolutional neural network (CNN). The classification isn't as 2016 separated into two stages to save computing time. We switched to Caffe as deep-learning framework and optmized it for the Nao. We shared our changes to Caffe in a public repository (https://github.com/tkalbitz/caffe)

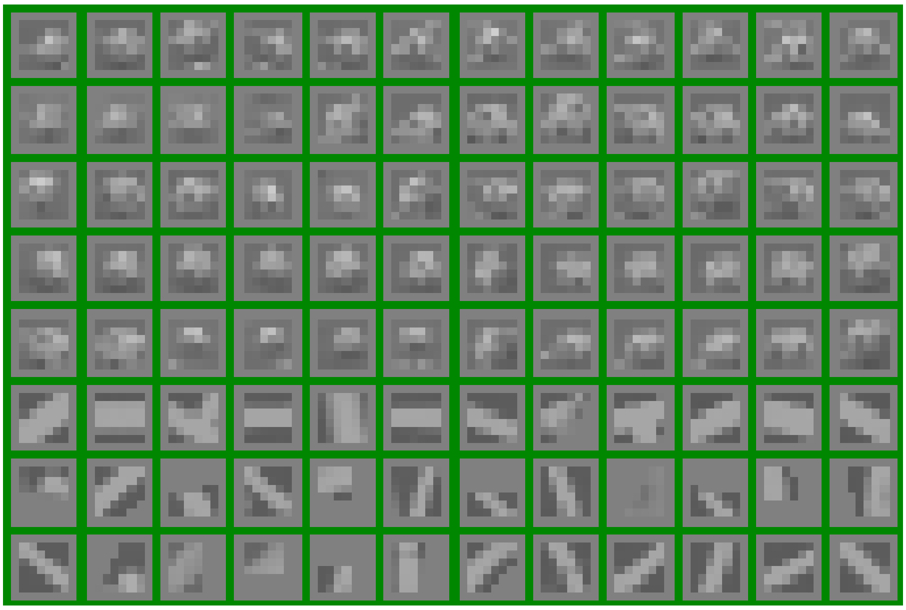

Figure 11: Hypotheses

The C20NN is a eight layer CNN, which uses 20x20px patches. It only classifies the hypothesis with the best score given by the first stage.

The statistics of our testset are 90.3% recall at 99.3% precision. The testset represents 30,000 ball true annotated and 60,000 false annotated images (resolution: 640x480). These images were recorded during test matches and the robocup in China and Leipzig and represent a realistic statistical distribution of possible game situations.

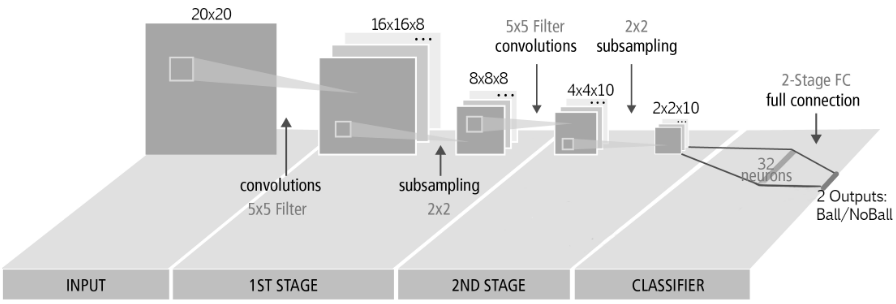

Figure 12: Classification CNN

2.2.2.3 Robot Detection As a consequence of using the Black and White Ball detection from 2016, we had to disable the robot detection, because of the performance restrictions given by the NAO hardware. Using this problem as motivation - a robot new detection was developed in [4]. To stay within the performance restrictions - even with the usage of a robot detection - the detection had an additional requirement to share as much steps as possible with the ball detection. As a result the detection uses the same separation of detection steps: Generate Object Hypotheses, Classify these Hypotheses and Create a visual memory.

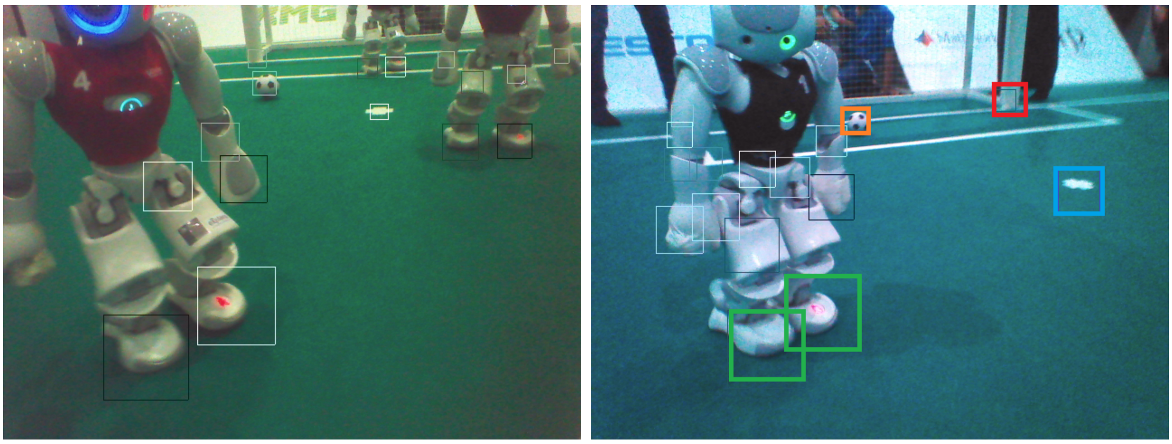

Figure 13: Object Hypotheses

The creation of hypotheses was developed to find areas, which could possibly contain the ball, a robot foot or a penalty spot (goal posts can be found, but are not further used in the detection). The resulting hypotheses are now classified by one deep convolutional neural network, which can differentiate between the three object types. This approach achieves a recall of 50% of all robots, 93% of balls and 78% of penalty spots - with an overall precision higher than 99%.

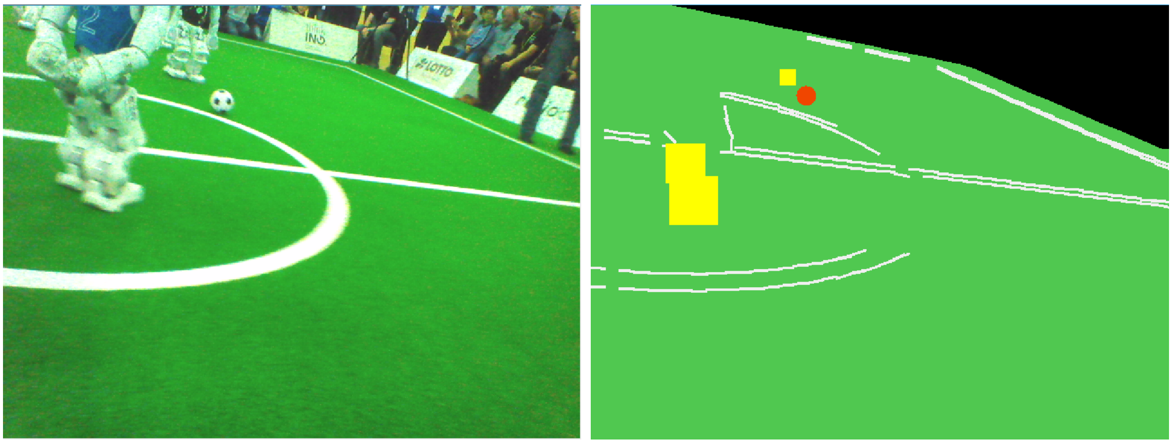

Figure 14: Original and classified image

To use the detected robots, the multi-target worldmodel - which was introduced in 2016 - was adjusted to handle all these objects the same way as before. Because of the clustering in the multi-target worldmodel, the two feets of one robot are merged into one robot object.

2.3 Relative Multi-Target World Model

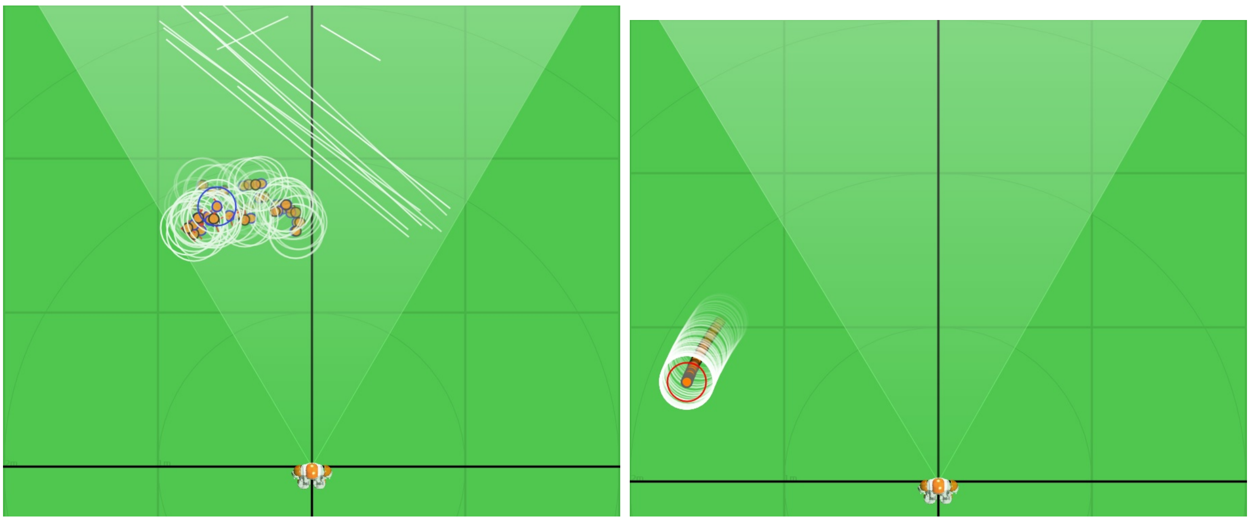

Figure 15: Multi-Target Tracker

The confidence in our orange ball detection in recent years allowed us a direct usage of ball information without a multi-target object model. To support the black and white ball detection, which has a lower recall, we implemented a relative multi-target world model. With an input of an object as a relative 2D position and a detection rate, this world model tracks all given objects separately. Because of that the object characteristics are not mixed up (e.g. velocity).

The multi-target tracker compares the new input's position with all currently stored objects. If the tracker decides a match, the stored object will be updated with the input data. There is no merging of old and new positions, meaning an update of an object overrides its position with the new input data. Further, at every frame the position of every stored object will be modified depending on the robot's odometry data.

An additional task of the tracker is to choose an object which is going to be used as the output data of the tracker. In case of the ball only one object can be chosen. For this decision every object holds two variables which are modified (increased or decreased) in every frame depending on whether it was detected or not. In case of not matching, a stored object is going to be held for a maximum of 4 seconds. This tracker is implemented for all kinds of objects on the field.

2.4 Infra-Red Data Transmission

Unreliability of the wifi network during most championship games prompted research into alternative data transmission methods. One of those is using the built-in infra-red transmitters and receivers to transmit complex data if a line-of-sight connection exists.

Data is sent in blocks of 15b with 8b/15b encoding through the LIRCD interface.

The protocol contains a 3b identifier for the Nao's jersey number, a 3b message identifier and 2b usable data, as well as a stop bit signalling the end of a transmission.

Transmissions of upto 3.6m are possible with acceptable error rates but a relatively low bandwidth.

This enables transmission of simple strategic information with low latencies between robots in close proximity.

2.5 Localization

Projections centered around the estimated camera attitude are sampled and evaluated based on the relative angles between the visible field lines. The most conforming projection is chosen and used to find a complete set of hypotheses of the player's position, from which the true position can be determined by using prior data. This method increases robustness of the localization in case of permanent camera movement (e.g. after a robot fell), fast head motions or external influences, e.g. in a fight with an opposing robot.

To resolve the field symmetries introduced by the use of two identical yellow goals since 2012, we analyze features in the surroundings of the field. By weighting the hypotheses of the localization according to how well they match to these features, the symmetry can be resolved. The features of the surroundings are updated continuously. More information can be found in [2].

2.6 Walking Engine

Until the beginning of 2010 we used closed-loop walking motions evolved through a genetic algorithm. These motions were fast but not omni-directional (eventhough walking along a curve was possible). This was a big disadvantage at the German Open 2010, so we decided to develop a completely different walking engine. Since Robocup 2010, our walking engine is based on a parameterizable walking model similar to [1] and is supported by a newly developed balancing algorithm. The big advantage of this system is full omni-directional capability and the ability to make fast direction changes whilst still being very stable.
The current walking engine was tuned for stability and speed manually and achieves forward speeds of upto 370 mm/s.

2.7 NaoControl

NaoControl is a monitoring program for our robots. It provides a virtual playing field showing the robot's and the ball's location. The Naos send their own and the ball's supposed position and an estimated localization quality to the program. With this, we can easily control whether the localization is fine or not. Also, the robot's rotation, field of vision and the current state of the strategy including its destination is displayed.

Figure 16: Screenshots of our NaoControl application.

Next to this, it is possible to show the actual images of the Naos' webcams. Those can be the real pictures or the segmented ones. We are able to send commands to the robots for testing. Additionally we are able to edit options live on the robot and in near future. We will be able to trace functions in the software. The virtual playing field and its lines can be well customized, so adaption to new dimensions causes no problems.

NaoControl is yet still in progress. In the near future it will be enhanced with simulation tasks. New playing strategies will be developed and tested with the assistance of NaoControl. For this purpose it provides simulated robot-behavior.

2.8 Motion Editor

The NaoMotionEditor is a replacement of Aldebaran's Choregraphe. The main purpose is to capture key frames directly from the robot, manipulate them and interpolate with a Groovy scripting engine between them. There exist already predefined Groovy scripts which define a linear and a smooth interpolation between two frames. These captured motions are saved in a XML file and can be later exported to a team dependent file format. To manipulate frames exist a variety of predefined operations like duplicate, mirror, move and show frames. The architecture of the editor is designed to add new functionality fast, so new requirements and features can be added on demand.

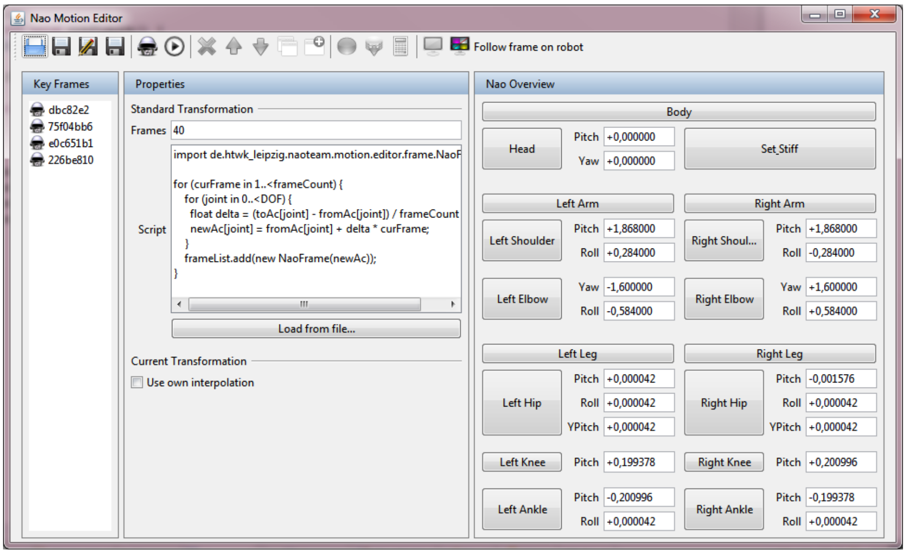

Figure 17: Example of a goalkeeper motion.

2.9 Architecture

2.9.1 NIO Framework

Our NIO (Nao Input Output) Framework is an independent piece of software that runs on Nao robots and extends the Aldebaran Robotics NaoQI framework.

The motivations for creating our own framework:

- Inconsistency of NaoQI's API
- Very limited debugging capabilities of NaoQI framework
- No need for a time intensive NaoQI restart after changes to parts of the software (e.g. motions, strategy)
- No thread safety of certain NaoQI calls
- Lots of NaoQI functionality we actually don't need

The basic functionality of our NIO Framework is defined by a Unix Domain Socket client server pair. We have built a simple C++ module for the NaoQI framework that exports a subset of the NaoQI calls through the socket to the requesting process. This module is compiled as a shared library and will be linked against our actual kernel (core executable of our framework). Our exported API calls are kept very simple and performant, the subset is small and threadsafe. On the other hand, NIO consists of a series of subsystems. Each subsystem is generally independent of the others and serves one single aspect.

2.9.2 Multiple agent system to determine the optimal short term strategy

Often a robot has to face difficult decisions like: Should I turn around the ball and shoot, dribble it in a big arc, or do a side-kick?
To facilitate situations like that and avoid big decision-trees or long if-else-chains, we introduced a multiple agent system into our architecture in 2016.

Our robots will now get simple commands from a team strategy module, e.g. "move the ball to the goal" or "walk to position x,y". These commands are interpreted by many agents running in parallel.

Each agent first determines, whether he can fulfill the command, and then computes how long it would take to do so. It then sends this result, including what it would like to do to fulfill the task, to an arbitrator which chooses the agent most suitable to do the work while ignoring commands from all other agents.

As an example, there could be 2 agents able to fulfill a command like "move the ball to the goal": One that tries to dribble the ball and one that does a straight shot.
If the goal is at an angle, the agent wanting to do a straight shot would have to move around the ball, then shoot. This would take a certain amount of time and also bring with it some risks like losing the ball or missing the goal in the shot.
The other agent, the dribble agent, would determine how long it would take to dribble the ball into the goal, also factoring in a possible ball loss or long duels with opposing robots.

Both estimates can now be evaluated and the movements of the best agent can be executed.

The system is also easily expandible: Say we want to add an agent that does a side-kick. It would also just have to determine if it can fulfill the order (e.g. if the ball is near enough to the goal, so a weak side-kick would suffice) and then determine how long it would take to do the task in a similar way to the straight kick agent, except that it needs to be aligned at a different angle to the ball.
It can also send the data to the arbitrator and will be chosen as soon as it is the most efficient agent.

This method doesn't require changes to already existing agents when a new agent is introduced, and it also provides a simple way to weight all options a robot has to fulfill its task.

2.9.3 Intra-robot communication

All the modules of our new architecture, e.g. the team strategy module, the agents, and the arbitrator need to communicate with each other in a fast, efficient, and thread-safe way.

This is implemented using 0MQ for queued communiation and a lightweight component based upon the pthread library for queue-less last-is-best communication.

2.9.4 Wiimote control

We have developed a Wiimote remote-controlled Nao movement interface for several testing purposes. This enables us to play against our developed Nao game strategy or to efficiently test new motions, which for instance have been set up by our own motion editor. The Wiimote is conneted via Bluetooth socket to a Bluetooth-enabled host machine that is within the same subnet as the Nao that will be controlled remotely. On the Nao-side a server application listens for incoming instructions that are sent from the host machine. With the help of our software, the host machine can even route multiple Wiimote connections to various Nao robots.

References

[1] Sven Behnke. Online trajectory generation for omnidirectional biped walking. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1597 – 1603, May 2006.

[2] Samuel Eckermann. Verbesserung der Selbstlokalisierung im Roboterfußball mittels optischer Umgebungsmerkmale. Bachelor thesis, HTWK Leipzig, 2012.

[3] Florian Mewes. Entwicklung einer dynamischen Spielstrategie auf der humanoiden Roboterplattform NAO. Bachelor thesis, HTWK Leipzig, 2014.

[4] Florian Mewes. Objekterkennung und Zuordnung im Roboterfußball (SPL) - Roboter. Master thesis, HTWK Leipzig, 2017.