Team Research Report

Nao-Team HTWK

January 2020

1. Team members and affiliations

- Rico Tilgner, M. Sc.
- Thomas Reinhardt, M. Sc.
- Stefan Seering, B. Sc.
- Tobias Kalbitz, M. Sc.
- Samuel Eckermann, M. Sc.
- Michael Wünsch, M. Sc.
- Florian Mewes, M. Sc.
- Tobias Jagla, M. Sc.
- Stephan Bischoff, B.Sc.
- Carolin Gümpel, B.Sc.
- Marvin Jenkel, Undergrad
- Andreas Kluge, M.Sc.
- Tobias Wieprich, M.Sc.
- Felix Loos, Undergrad

All team members are affiliated with HTWK Leipzig (Leipzig University of Applied Sciences).

2. Notable work and fields of interest

In Section Notable work and fields of interest we describe various areas of work where Nao-Team HTWK achieved significant progress. With regards to Section A.1 of the 2020 Rules we suggest the following changes in 2019 for consideration:

- our new and open sourced Walking Engine for the Nao v6
- a field border detection via a neural network (see Field Border Detection)
- a neural network ball hypothesis generator in the lower camera for hard situations which is a lot more reliable than an algorithmic one (see Hypothesis Lower Cam)

The following sections list our approaches to various modules.

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 team strategy gives a constant gameplay, which is suitable for testing and improving basic soccer skills. But to hold a competitive ability and to reach a humanlike soccer gameplay, a new approach is needed. Each player must be able to make a decision for an 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 team strategy can be a solution. The graduation work in [1] (language: German) shows the development of the above described dynamic team strategy. 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 team strategy is the development of a defensive and offensive position finding behavior.

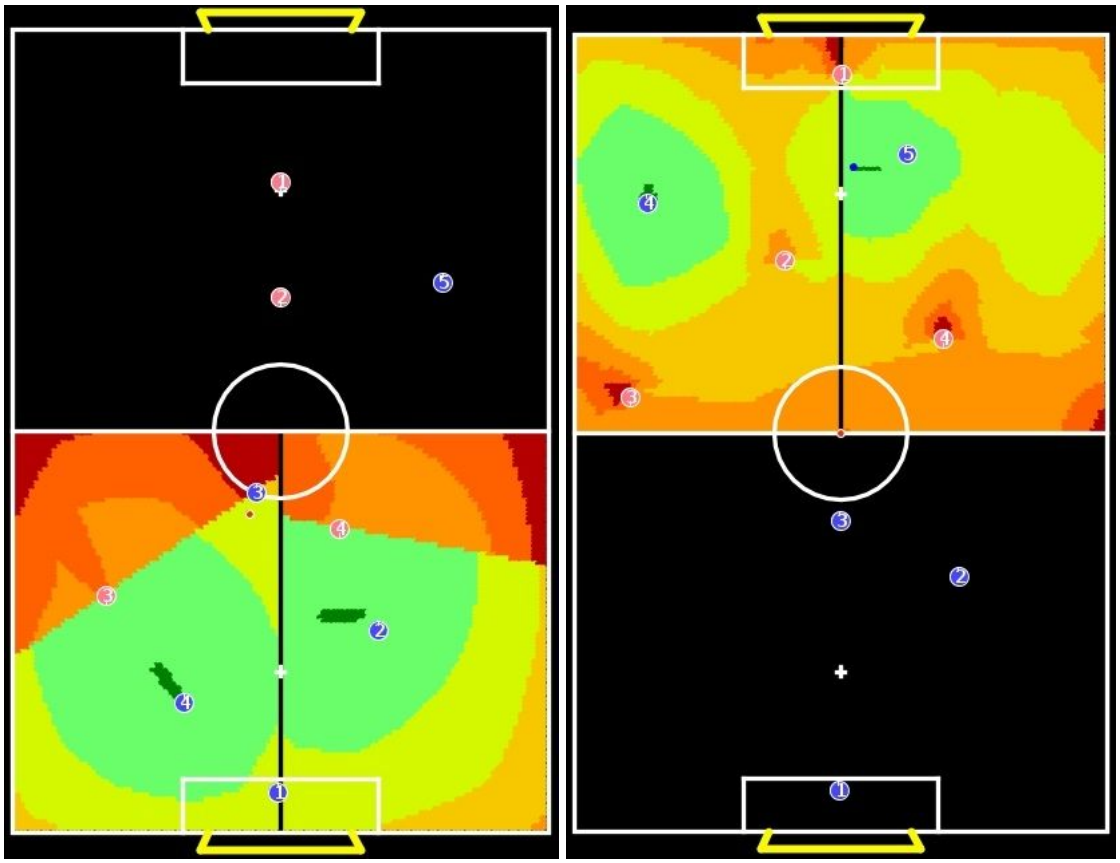

Figure 1: Offensive and Defensive Player Positioning

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.

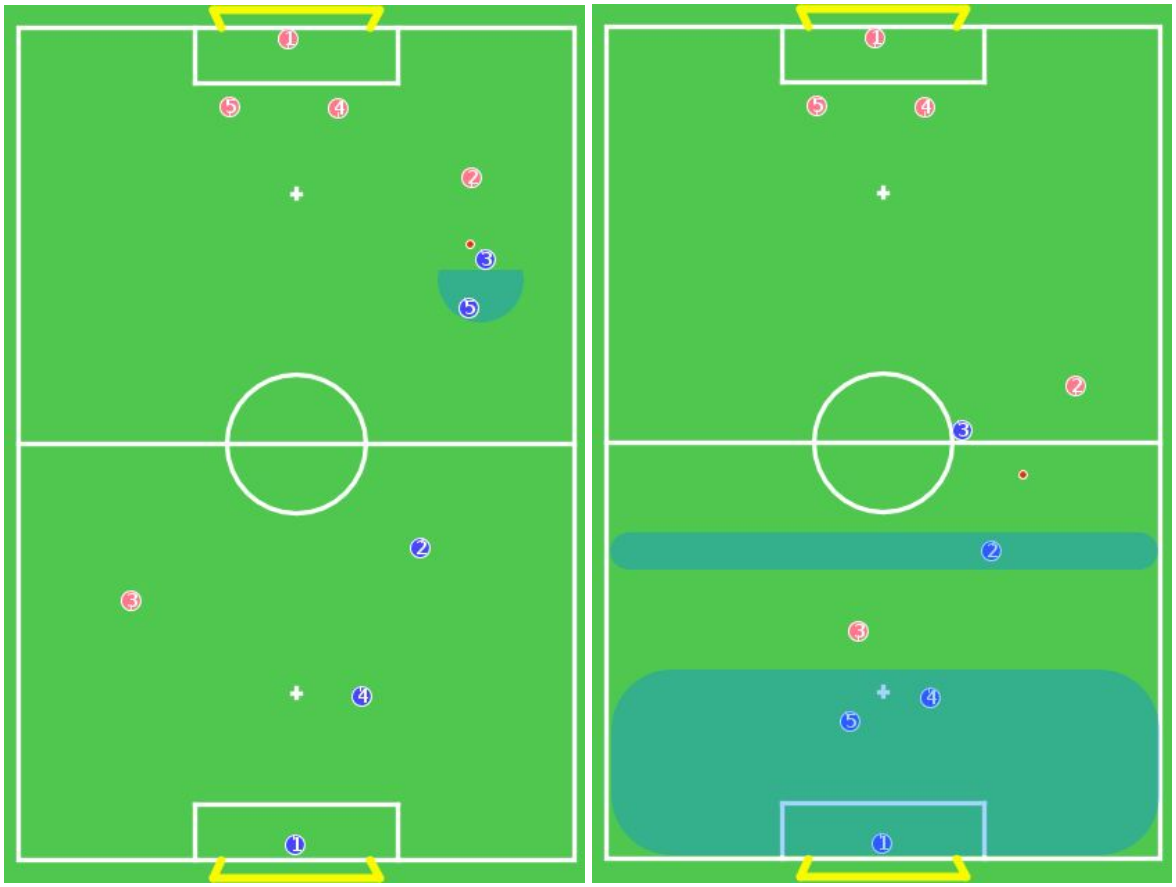

Figure 2: Range of shadow system, defined defender areas

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 react to long shots and to spread team players over the field. To have as many 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.

2.1.2. 2018 Strategy Improvements

The dribbling strategy from 2017 turned out to be a very difficult strategy to counter. But because of the “ad hoc” development during the robocup (to just have another strategy) it showed some flaws, which needed to be addressed. The behavior rework or improvements to overcome these flaws are content of the following sections.

2.1.2.1. Decision making: “Which robot goes to the ball?”

Since the introduction of our dynamic strategy in 2016 - the decision “Which robots goes to the ball?” was only decided by one factor: the distance between each robot and the ball. In addition, this calculation was executed by all robots using the current worldmodel information of each robot. In certain situations this lead to the misbehavior that two or more robots or no robot at all went to the ball (for example: noise on the position from the localization or the worldmodel from a robot was not updated because of network issues). These situations appeared more often with the dribbling strategy, because the shadow robot is always near the striker and the ball. To prevent this misbehavior the new concept has now three factors to influence the decision:

- the angle which the robot has to move to walk to the ball
- the distance which the robot has to move to the ball
- the angle which the robot has to move around the ball to dribble it into the goal

And to combine these factors into a single time value we are using the turning- and walking-velocity of our robots. The question “Which robot goes to the ball?” is now answered by the lowest time value a robot needs to position itself to dribble the ball into the goal.

We also introduced the concept of “trust” of each robot towards the team ball and use more accurate relative ball distances instead of team ball distances to decide upon the striker.

2.1.2.2. Positioning: Prevent running into the striker

Another misbehavior was observed from robots which are trying to position themselves according to the dynamic strategy. The evaluation for the optimal position did not include the position or the walking path of the current striker. When for example the striker loses the ball and an opponent kicks the ball into our field-side one of the defenders switches into the role of being the current striker and the last striker switches as well into a new role of being either a defender or the shadow. The defender’s as well as the shadow’s position is always behind the striker, which leads into an intersection between at least these two robots. To resolve this misbehavior the evaluation was improved by adding the current striker. So the optimal position is getting manipulated, that the robot either walks around the striker or waits near the field border until the striker has gone by.

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 is 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 2018 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 2017 version (Caffe based). Please be aware that it does not contain a pretrained net for the ball detection. 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 (except Tensorflow Light 2.x for the neural networks)
- 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 2018. 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.

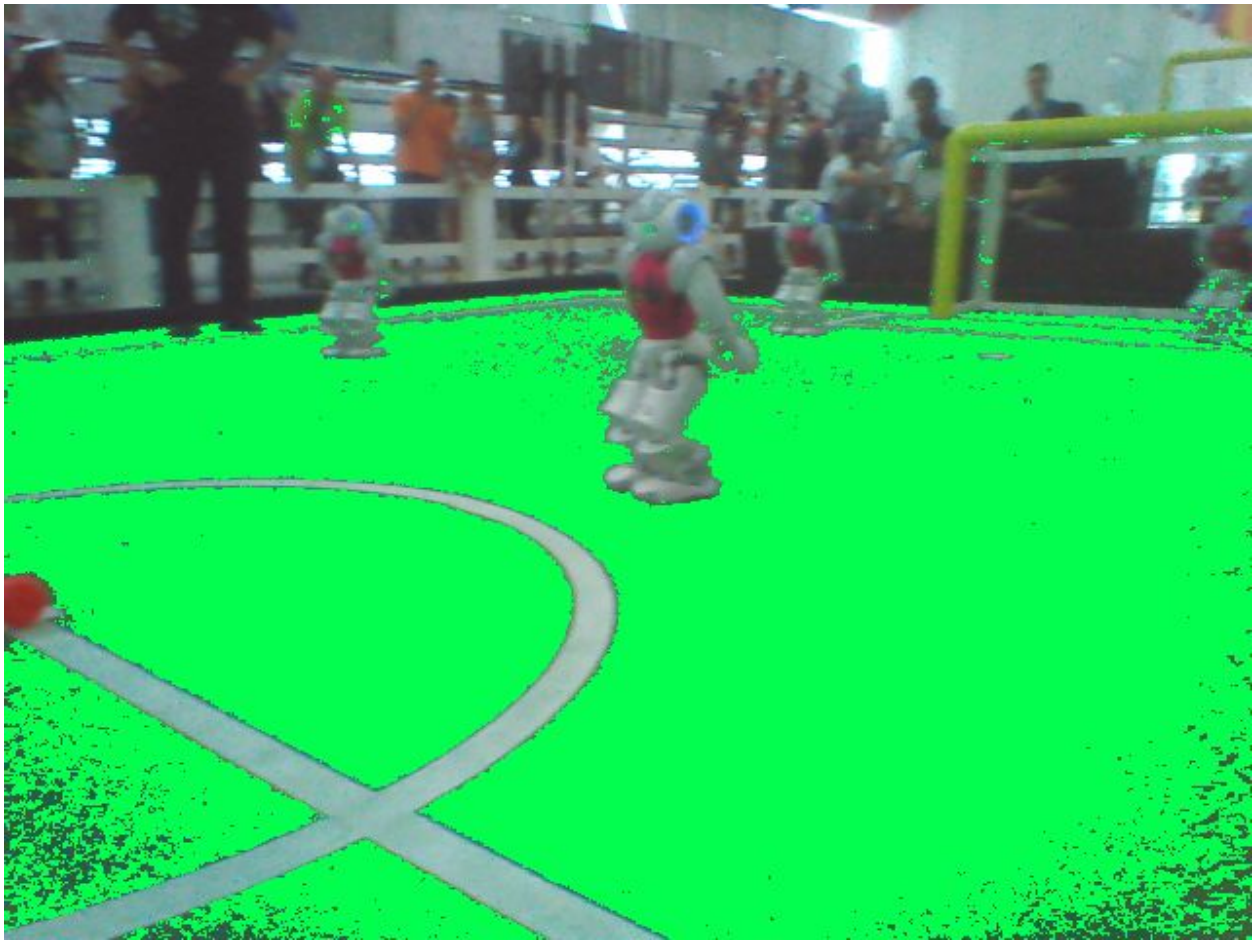

Figure 3: Automatically detected field color

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.

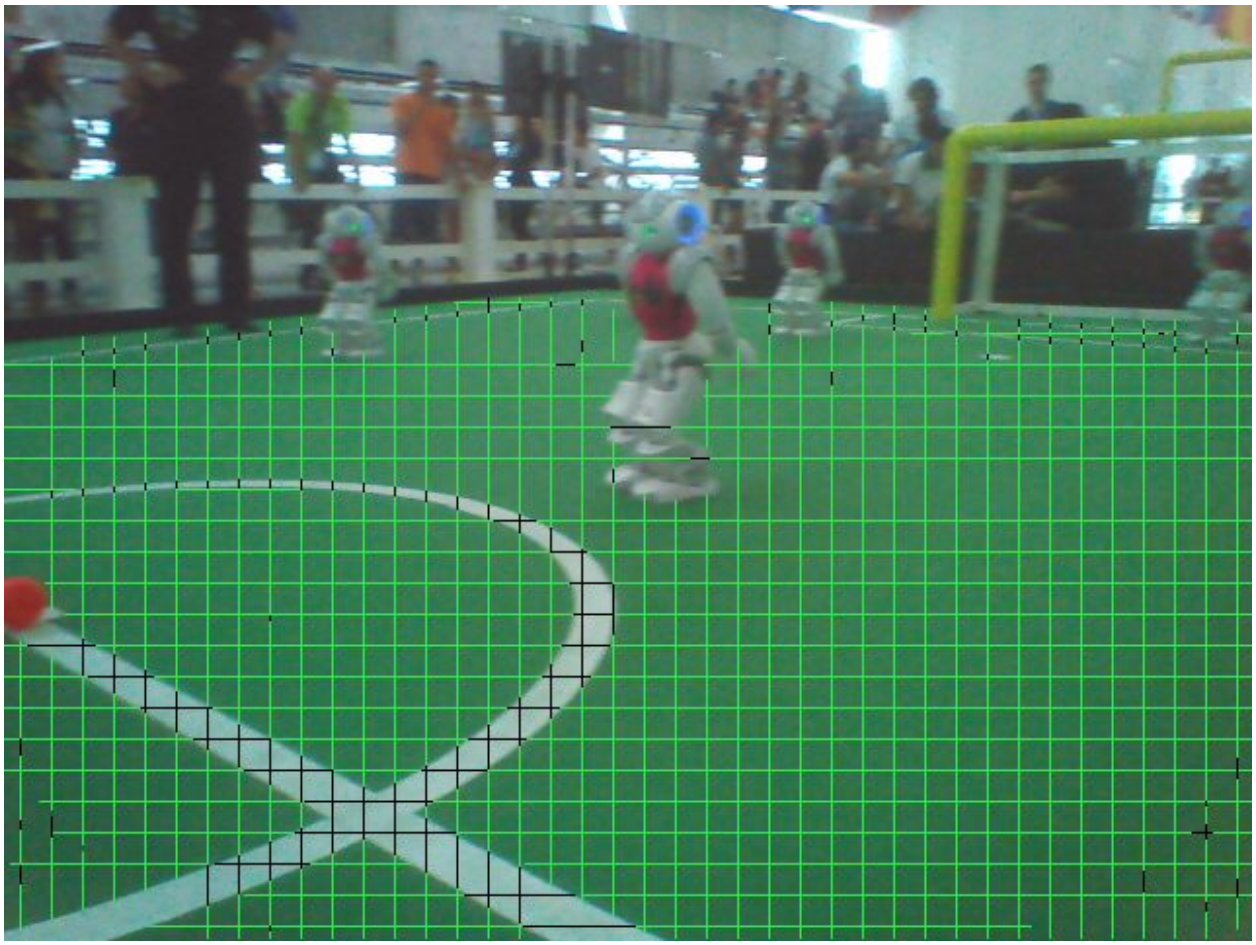

Figure 4: Scanline Classification

2.2.1.4. Field Border Detection

The field border detection was reimplemented for RoboCup 2019 in Sydney using a deep neural network for the upper camera image. Input is a 40x30 YUV422 image, processed through 4 adapted Inceptionv2 modules. The output is a 40 element tensor which represents the field border of the input image.

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 precisely detect the inner and outer edge of the center circle.

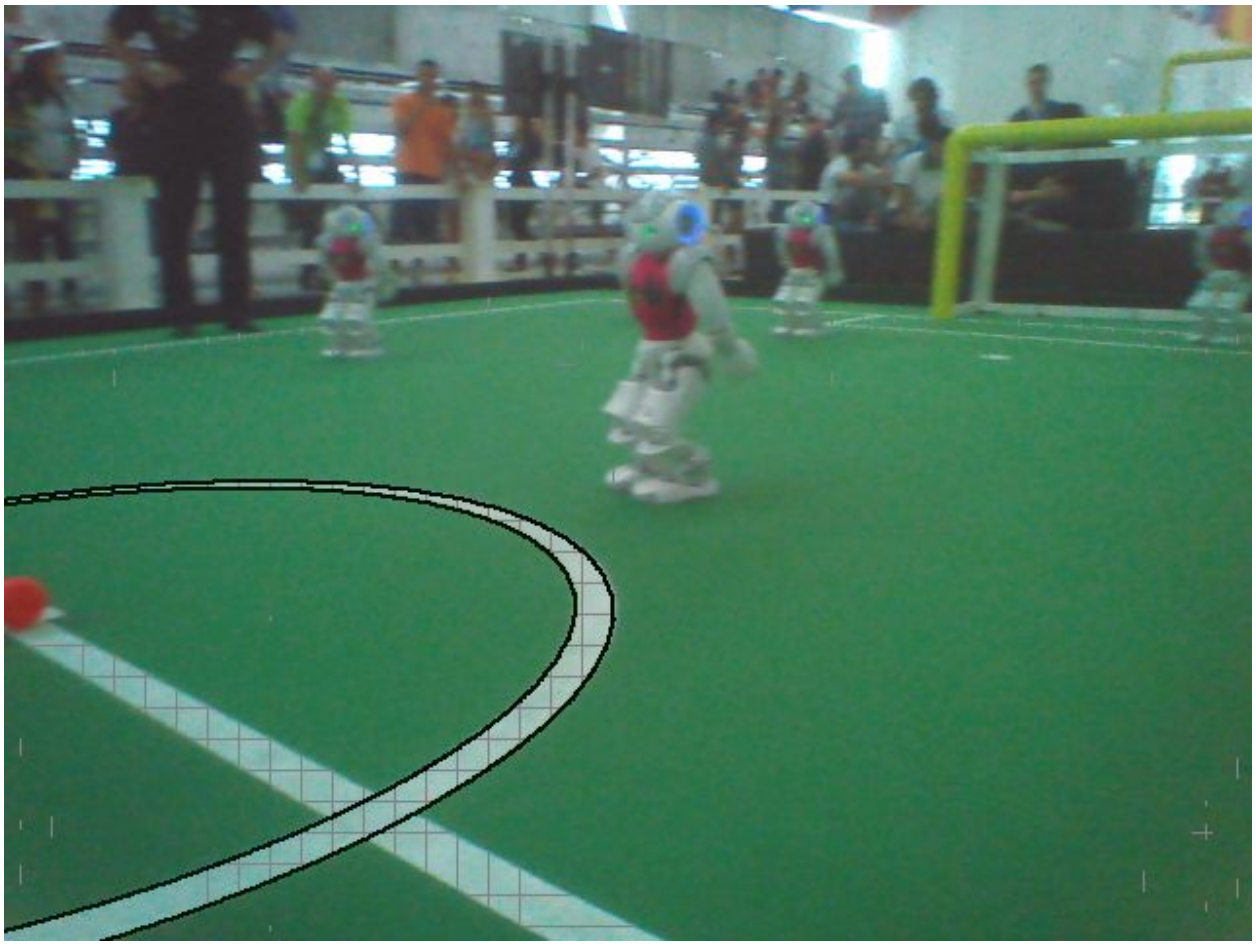

Figure 7: Center Circle Detection

2.2.1.7. Near Obstacle Detection

This 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 a small neural network.

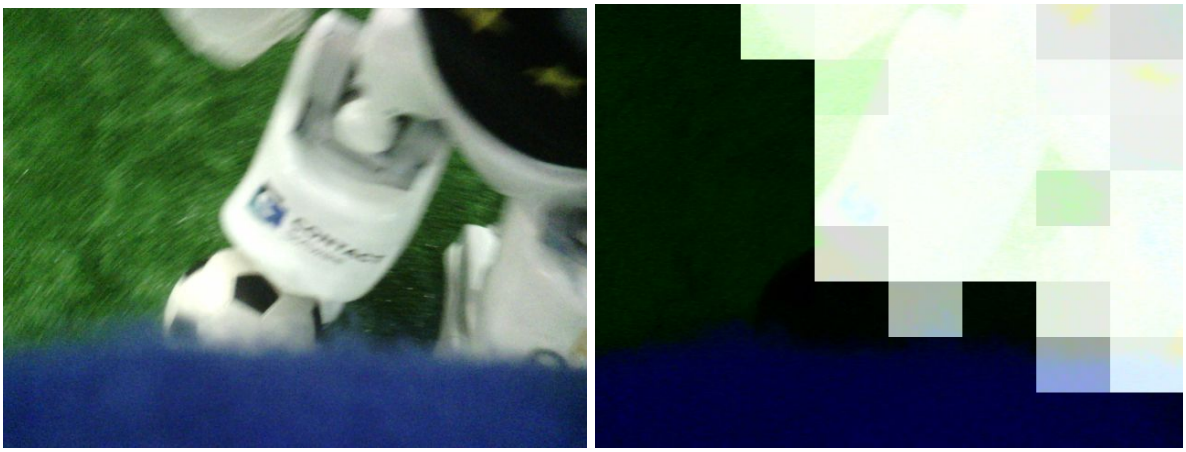

Figure 8: input (left) and output (right) of the near obstacle detection

2.2.2. Black and White Ball Detection

As a development in difficulty the SPL changed the orange street hockey ball that has been used since 2010 to a ball with a black and white classic soccer ball pattern. Our 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 (Upper Camera)

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.

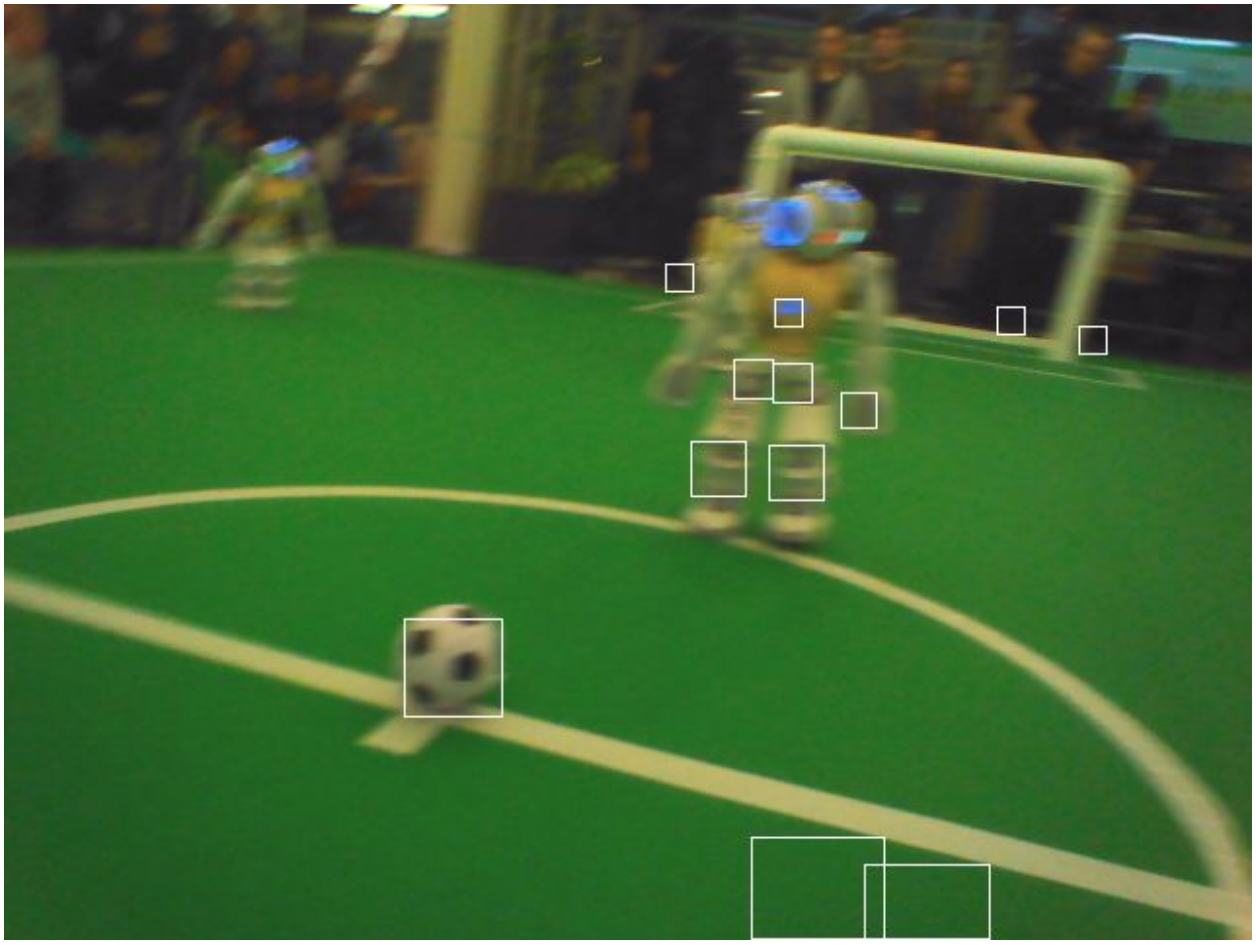

Figure 10: Ball Hypotheses (Upper Cam)

2.2.2.2. Hypothesis Generation (Lower Camera)

The lower camera hypothesis generation struggled a lot with images where the ball was clamped between the legs of the robot and with the darker grey color used in the Nao v6. Hypothesis generation was changed to a neural network which creates exactly one hypothesis. Input is a downsampled 40x30 YUV422 image, processed through 4 adapted Inceptionv2 modules. The hypothesis generator outputs a (x,y) position in the image where best hypotheses is expected. Afterwards this position is processed through the hypotheses classification.

Figure 11: Ball Hypotheses (Lower Cam)

2.2.2.3. Hypotheses Classification

The second part of the ball detection is the classification of the generated hypotheses with a deep convolutional neural network (CNN). We use Tensorflow Light as deep-learning framework and ported it for the Nao.

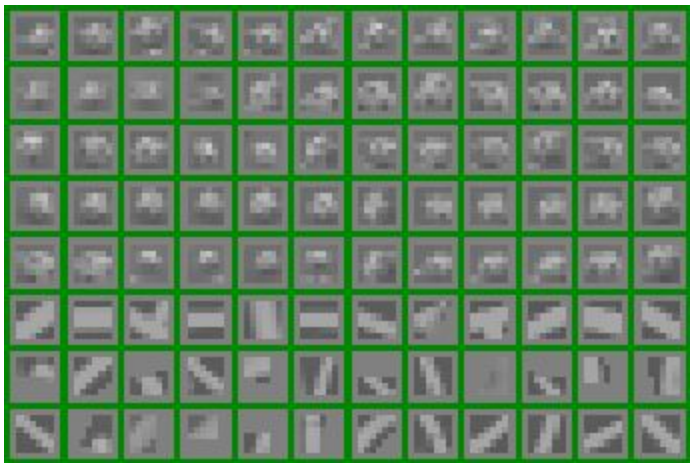

Figure 12: Hypotheses

We classify the 16 best hypotheses given by the first stage using a CNN. We use 12x12 px inputs with 2 convolutions with batch norm connected through average pooling and a final fully connected layer.

2.2.2.4. Robot Detection

We improved our new robot detection from [2] by now sharing the CNN with the ball and penalty spot detection. The shared detection allowed us to evaluate more hypothesis in each frame than before.

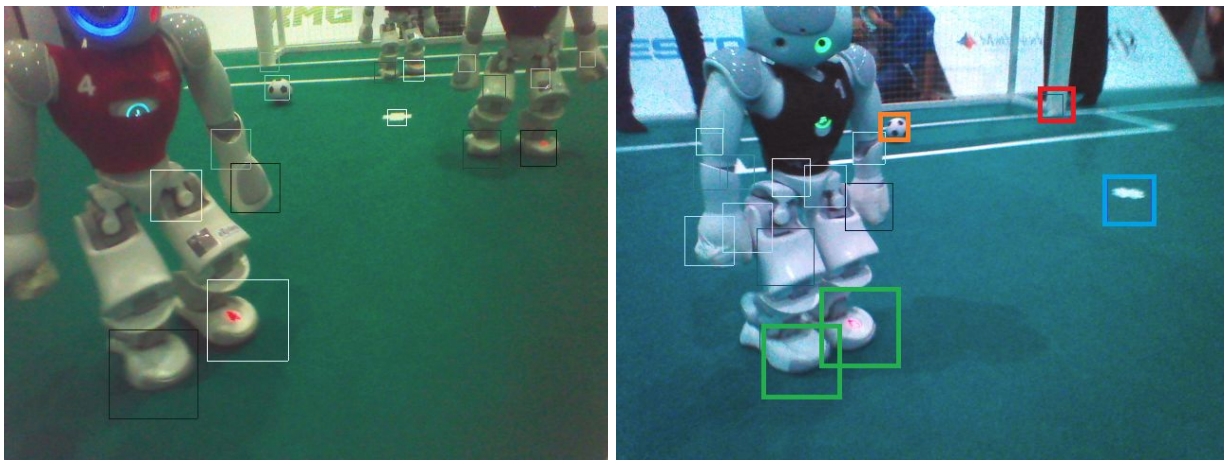

Figure 13: Object Hypotheses

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%.

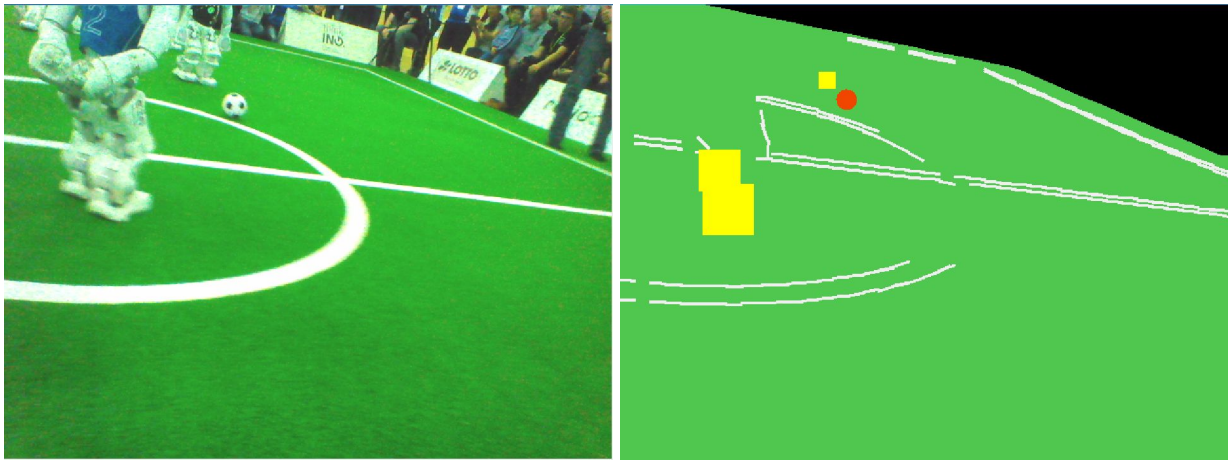

Figure 14: Original and classified image

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 multitarget worldmodel, the two feet of one robot are merged into one robot object.

2.3. Relative Multi-Target World Model

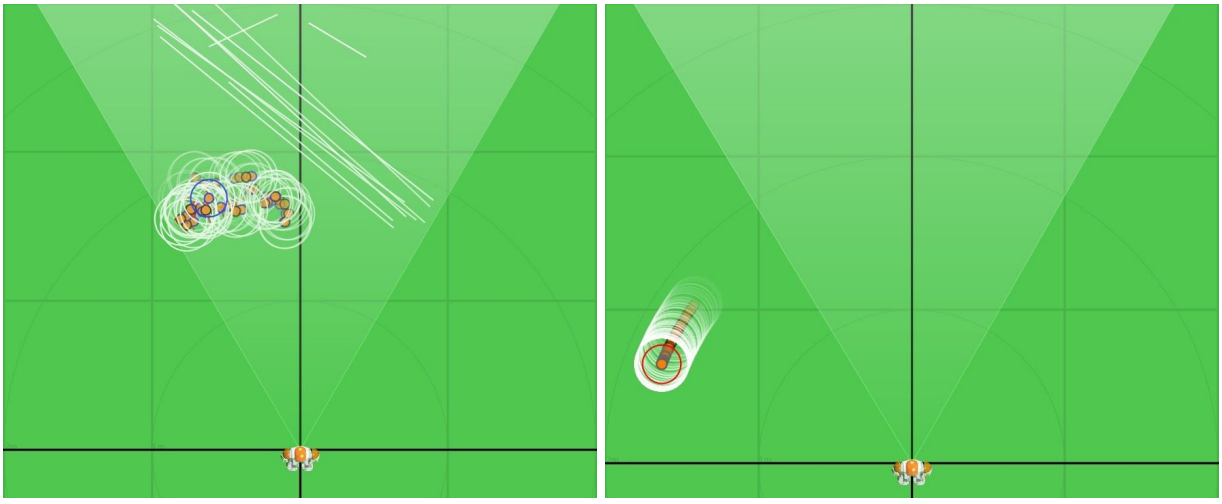

Figure 15: Multi-Target Tracker

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. 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.

2.5. Walking Engine

Our 2019 walking engine is available as Open Source at:
https://github.com/NaoHTWK/HTWKMotion

In 2018 we introduced a completely redeveloped walking engine. It's based on predicting support foot switches and uses direct gyro ankle balancing. In 2019 we added additional stability control by tracking the body pitch and gyro pitch errors. The algorithm dynamically changes the step duration and forward velocity to compensate for unstable situations. For example when the robot tilts backwards it increases the step duration and walks slower (or stops) until the upper body angle error is small enough again. This new stability controller enabled us to significantly improve our stability while walking in 2019 and our robots fell around 75% less often than in 2018.

Also we tuned the upper body forward shifting motion by adding an additional swing compensation function to generate more smooth and energy efficient locomotion and less camera vibration while walking.

With these two improvements we achieved also a slightly faster default forward walking velocity of 28 cm/s and sidewards velocity of 35 cm/s in 2019. It should be possible to walk 20-30% faster but this is not recommended because of possible joint overheating issues and due to increased gear abrasion over time.

2.6. NaoControl

NaoControl is a monitoring program for our robots. It provides a virtual playing field showing the robots 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 the near future. We will be able to trace functions in the software. The virtual playing field and its lines can be well customized, so adaptation 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.

2.7. 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 an 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.

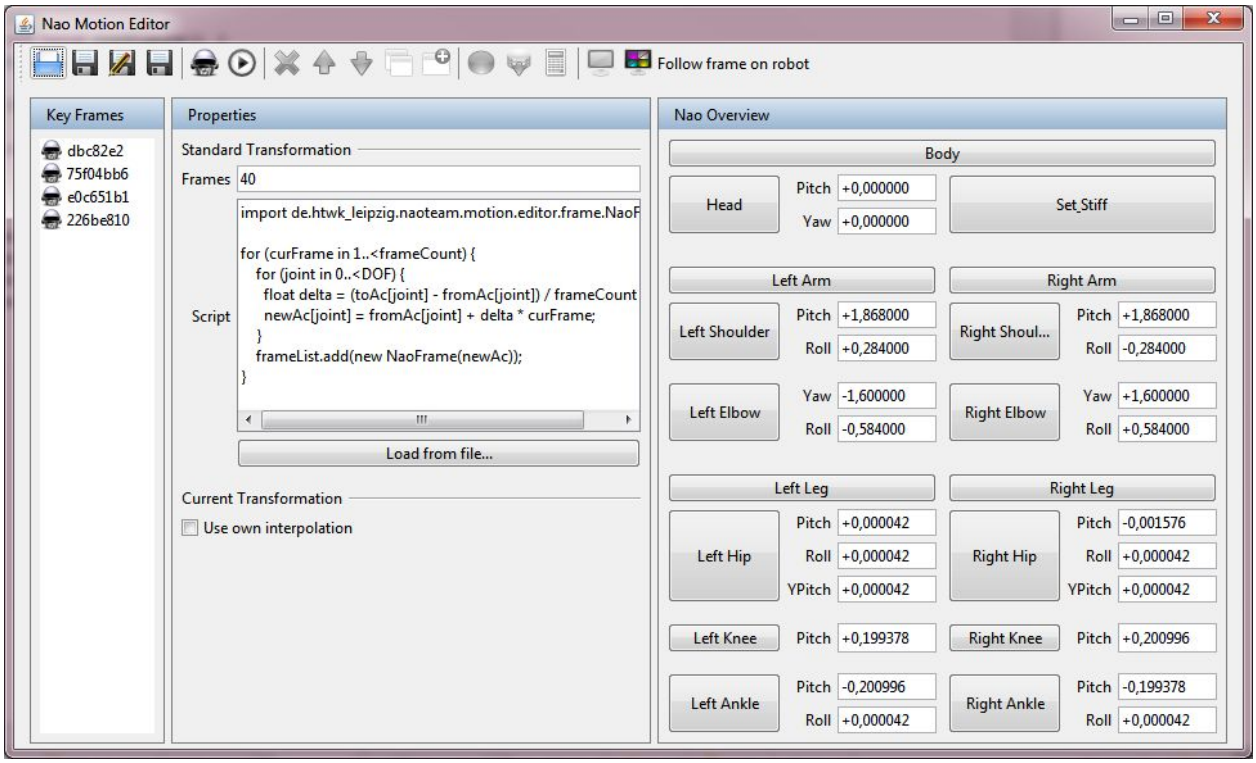

Figure 17: Example of a goalkeeper motion.

2.8. Architecture

2.8.1. 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 thread safe. On the other hand, NIO consists of a series of subsystems. Each subsystem is generally independent of the others and serves one single aspect.

2.8.2. 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 expandable: 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 weigh all options a robot has to fulfill its task.

2.8.3. 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 a lightweight component based upon the pthread library for queue-less last-is-best communication.

2.8.4. 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 connected 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.

3. References

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

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