University of Amsterdam
Faculty of Science
The Netherlands

Dutch Nao Team

Technical Report

Authors:
Caitlin Lagrand
Michiel van der Meer
Jonathan Gerbscheid
Thomas Groot
Sébastien Negrijn Patrick de Kok

Supervisor:
Arnoud Visser

October 14 2016

Abstract

In this Technical Report, the Dutch Nao Team presents its progress from the past year, and an outline of the developed modules. In 2016, the team went to RoboCup European Open in Eindhoven and the 2016 RoboCup World Finals in Leipzig. The team used BHuman's framework in development, in addition to two new modules. A ball detector was developed to account for the new ball in the Standard Platform League, and a new behaviour engine was created in combination with a behaviour designer.

Figure 1: Team photo Leipzig.

1 Introduction

The Dutch Nao Team consists of seven Artificial Intelligence students, three Master and four Bachelor, supported by a senior staff member. It was founded in 2010 and competes in the Standard Platform League (SPL); a robot football league, in which all teams compete with identical robots to play football fully autonomously. The league was started to incentivize the development in robot science. Its current goal is to play against the human world champion in 2050. In the Standard Platform League, all teams use identical robots, which shifts the focus to software development rather than hardware development. The robots are non-player controlled, and have to be able to play football by themselves. This includes seeing the ball, locating itself and making decisions on how to play next, as well as communicating with teammates.

2 Hardware and framework

2.1 NAO

The Nao robot is a programmable humanoid robot made by Softbank Robotics, formerly known as Aldebaran Robotics. The CPU used by the robot is a 1.6 GHz Intel Atom single physical core which is hyper-threaded, combined with 1GB of RAM and 8 GB of storage. Since there are only two logical cores, CPU resources are scarce, which limits calculation heavy tasks such as ball detection and localization. The Nao has two HD cameras and an inertial board that are both used for the competition. The HD cameras are located in the head of the Nao, one is aimed forward to look for

far away objects and the other is aimed downwards for looking at objects close to the feet of the Nao. The inertial board is used for determining if the robot has fallen, something that happens regularly during matches. The Nao also possesses a sonar, an infrared emitter and receiver, pressure sensors and tactile sensors. These sensors are however less frequently used than the cameras and inertial board because they are not as essential to the footballing capabilities of the robot. However, what is essential in order for a robot to play football is movement options. Depending on the version the Nao Robot has either 14, 21 or 25 degrees of freedom in its joints. The joints in the legs allow for stable bipedal movement and various kicking motions, the arms are generally used for helping the robot stand up again after falling and for stability whilst walking, although it is allowed for a goalie to hold the ball in its hands teams rarely make use of this. Even though every robot is supposed to be the same, individual differences are noticeable when the robots walk, the movement of older robots is less stable and fluid. In order to ensure a robust walk for every robot the joints for each individual robot need to be calibrated.

2.2 B-Human

The development of a DNT framework would be very time and resource intensive and was therefore not a possibility. This is why we decided to use the BHuman framework. BHuman offers a module based system, in which each module can be easily integrated with the BHuman core. This core offers low level functionality, which allowed us to focus on higher level modules. BHuman is also well documented, making it a favorite among beginning teams for the RoboCup. BHuman also includes a program called SimRobot which is separate from the framework itself. SimRobot allows us to calibrate robots for color, white balance, joint offsets and camera offsets. Using SimRobot while the robot is on the field we can check the workings of the various modules in case we need to debug. For instance we can see what behaviours the robot is making using the behaviour engine, or see if the robot is having trouble recognising the ball in certain areas. All these features make SimRobot a vital part of the BHuman framework and allows us to work much more efficiently. The main modules created for this year's events are outlined in the following sections. Our team forked the BHuman 2015 Code Release repository[1] and used this as a basis for both RoboCup events. The modules created for this year's events are outlined in the following sections.

3 Detecting the ball

This year the color of the ball was changed from orange to a blocked black and white pattern. This ment that an entirely new ball detector had to be created. We tried multiple approaches including a Viola-Jones Classifier [Viola and Jones, 2004] from OpenCV[2] and cognitive image processing [Ras, 2015]. The Viola-Jones approach did not classify as well as we had hoped, but more importantly was too slow to be useful in a match even if the training set would be improved. A new ball detector was created using regular expressions. To use regular expressions, each pixel is classified as either black, white, green or null, the image is converted to an array of characters corresponding to each color: b, w, g or n. A regex like Figure 2 is applied to each column of pixels in the image to match patterns which are potentially balls. To speed up the process this process is not applied to every column, instead Bhuman's scanlines are used as columns. Multiple regular expressions have been written, in Figure 2 two expressions can be seen, a simple one, a, that is used when lighting conditions are worse and a more refined one, b, that works better in good lighting conditions. Regex a only checks

[1] `https://github.com/bhuman/BHumanCodeRelease`

[2] `http://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html`

(a) `b.{0,5}w|w.{0,5}b`

(b) `w+[ng]{0,3}b+[ng]{0,3})+w*[ng]{0,3}g+`

Figure 2: The regular expressions used for the ball detector.

for either a black pixel followed by noise and a white pixel or a white pixel followed by noise and a black pixel. Regex b searches for a repeated pattern of white and black pixels, containing 0 to 3 green or noise pixels followed by at least one green pixel to signify the end of the ball area.
Both of these methods result in a lot of false positives, detecting for example the goal or other robots as balls (see Figure 3). Filtering these potential ball spots is therefore a essential step in the detection process. The first step is eliminating false positives using a minimal ball size. When the regex matches to a potential ball, the detector uses the found location in the original image to expand the ballspot horizontally. This combined with the original vertical matched line forms a cross through the potential ball. The x and y coordinates given by center of the two lines gives the center of ball and the length of the lines is used to calculate the ball size. The next step is to ignore candidates that are either inside of a robot, part of the jersey, not on the field or have an unrealistic radius. The resulting ballspots are fairly accurate and the entire process does not take a lot of computational power, the downside to this detection method however is that it is quite sensitive to color and white balance calibration and that not all false positives are filtered.

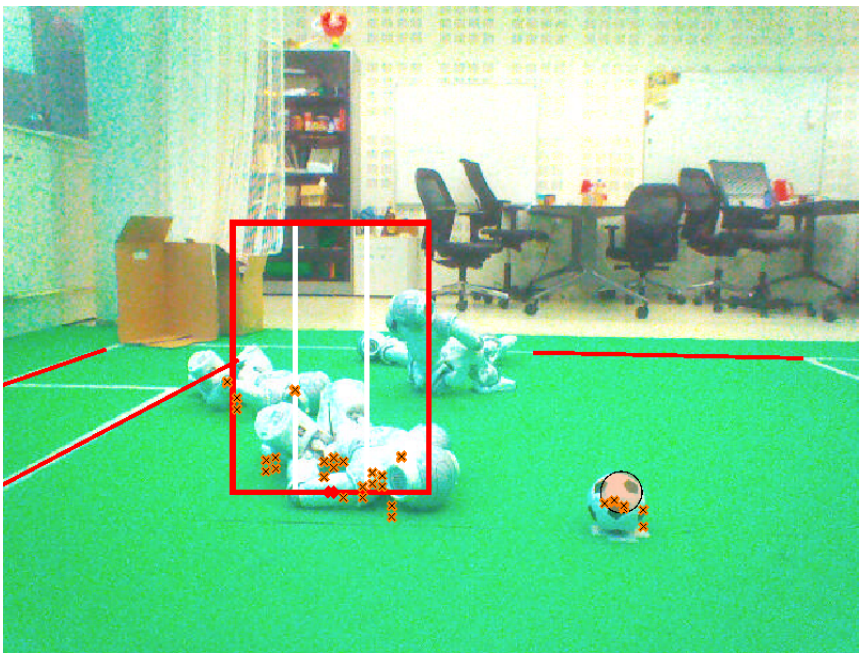

Figure 3: Ball detection seen through the eyes of the robot.

4 Behaviour engine

After having worked with BHuman's CABSL behaviour engine, the team decided that a more intuitive method of defining behaviours would make for a great addition to our code. The idea

Figure 4: Graph in the Behaviour designer

behind this new engine came from the game Guild Wars 2, which adopts an axis-based method for making decisions. It was shown at the Game Developers Conference in 2015[3].

4.1 Axis-method

Instead of the Finite State Machine approach that CABSL uses, the new engine uses decisions, called axes, which gain score depending on environmental factors. Consequently, the highest scoring axis wins, and its action is performed. The environmental factors that an axis is based on can be any method that is available in the system. Various frequently used methods are wrapped into a small library, simplifying the search for suitable functions. Each axis thus depends on a number of environmental factors, we call these factors considerations. Considerations are paired with a graph, this graph determines the final score of a consideration from the score returned by the environmental functions and enables us to describe complex relations intuitively. These environmental functions are collectively called the utility function of the consideration. For example when we are deciding if we have to start searching for the ball, we might consider how much time has passed since the last time we have seen the ball. We don't want to immediately start searching because we might just not recognize correctly the ball even though it is in sight. We also would not want to keep searching if we haven't seen the ball in a long time, as in that case it would likely be better to return to a starting position. This entire behaviour can be described easily using a graph like in Figure 4. All consideration scores are multiplied and normalized, as seen in Figure 5 (1) and the action of the decision with the highest score, chosen with Figure 5 (2), is executed.
Since each axis' score has to be calculated in order to find the highest, some optimization was necessary to speed up this process. By means of a utility score, a designer can specify a usefulness among the different axes, which can in turn be used to eliminate a possible axis before its entire score is calculated.

To keep track of the game's progress, some sort of finite state machine was needed to make the

[3] `http://www.gdcvault.com/play/1021848/Building-a-Better-Centaur-AI`

$$\sigma(a) = a + a(1-a)\left(1 - \frac{1}{\| C(d)\|}\right) \tag{1}$$

$$d = \arg\max_{d \in D} \left\{ u(d) \prod_{c,a \in C(d)} \sigma(a(c, Sensors)) \right\} \tag{2}$$

Figure 5: Equations for calculating decision scores where u is the utility score, a is the score of the consideration, norm $\| C(d)\|$ is the amount of considerations of the decision d, D is the set of all decisions and $\sigma(a)$ is the normalized function score

robots able to respond to different game states. In our engine, each game state is seen as an Event, which can be raised or lowered. For example, in the set state, the Events GameSet will be raised, and after transitioning to a playing state, the GameSet event will be lowered and the GamePlaying event will be raised. Included in this method of raising and lowering events is the swapping of decisions, which made the robots able to only consider decisions relevant to the current game state.

4.2 Designer

In addition to the new behaviour engine, the team also created a web-based designer for the behaviours, which makes it easier to both create new and update existing behaviours. An example of a simple striker can be seen in Figure 6. After loading in a C++ file containing the various axes, JavaScript was used to parse and visualize all available data in a web browser. This overview is editable, after which the changes are converted to C++ code, ready to plug into the behaviour engine.

To simplify the process of designing the normalization graph, we used splines as a method of visualizing the functions. Splines are able to interpolate linearly or quadratically between knots that can be specified in the user, creating an intuitive yet mathematically correct way of designing normalization graphs.

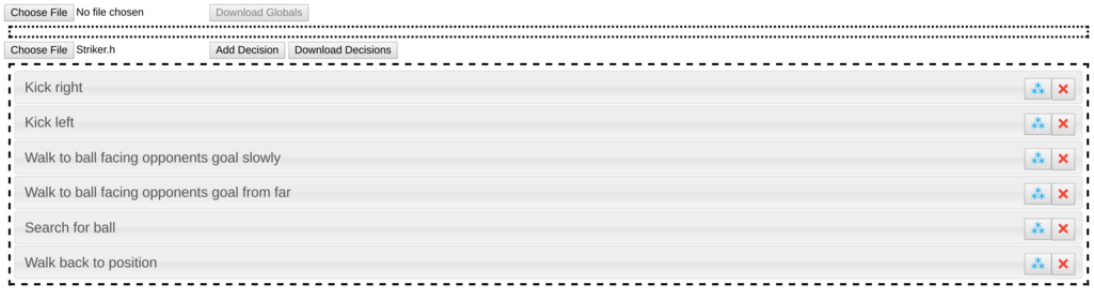

Figure 6: Actions in the Behaviour designer

5 Results

5.1 European Open (Eindhoven)

The European Open at Eindhoven was the first competition the team competed in this year. The new behaviour engine was not finished yet, so the behaviour engine of B-Human was used during this competition. The first match was lost against B-Human (0-7). A lot was learned from this match and improvements to the ball detector and behaviours were made for the second match against Nao Devils. The second match was also lost (2-0), but again a lot was learned. The next match resulted in a tie against HULKs (0-0). A penalty shoot-out was needed to get a winner. Unfortunately, we had never practiced this and the HULKs won.

5.2 RoboCup Leipzig

The Dutch Nao Team qualified for the RoboCup 2016 at Leipzig. The new behaviour engine was used during this competition, as well as an improved ball detector. The first match was lost against UT Austin Villa (6-0). The biggest problem encountered during this match was the ball detector. It was detecting too many things as a ball, for instance field lines and other robots. This was fixed by filtering out false positives. During the next match, against Linköping Humanoids, nothing happened. There was a bug in the behaviour engine and the robots did not do anything. Luckily, the robots from Linköping did not move as well, which resulted in a tie (0-0). The last match from the Robin Pool was against SPQR, who had also lost with 6-0 and a tie against Linköping. This match again resulted in a tie, so a penalty shoot-out was needed to determine the second and third place. SPQR scored one out of three and we did not manage to score, so SPQR became second and we were third in the group. The next match was a knock-out game against DAInamite. After being equally strong during the match, the the score was 0-0. Even after the penalty shoot-out, both teams had an equal score of 2-2. A sudden death shoot-out followed, in which DAInamite scored, while our striker could not find the ball.

6 Conclusion

In the future, we look forward to welcoming more members. The coming year our team will continue to work with the BHuman framework, improving our own modules, such as the ball detector and behaviour engine. In addition, the team would like to create a new localization method. However, some modules, like the motion engine, will likely not change. Replacing additional modules might spread the workload which could cause our work to go unfinished. If we want to produce results, improving what we have would be our best option in most cases. The localization method is an exception to this, as we had issues with the current module from the BHuman framework. Creating our own method would also enable us to debug more easily, as we would have a better understanding of the code.

We gained a lot of experience from this year's RoboCup and even though we did not perform very well, everyone in our team learned a lot from their own projects. Now that all of us have more experience we will be able to work with the Nao platform more comfortably and hopefully place well in the next RoboCup.

References

Gabriëlle EH Ras. Cognitive image processing for humanoid soccer in dynamic environments. *Bachelor thesis, Maastricht University*, 2015.

Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.