Dutch Nao Team

Technical Report 2012
`http://www.dutchnaoteam.nl`

Faculty of Science, Universiteit van Amsterdam, The Netherlands

October 28, 2012

Contents

1 Introduction

The Dutch Nao Team consists of Artificial Intelligence students from the Universiteit van Amsterdam, supported by a senior staff-member. The Dutch Nao Team is the continuation of the Dutch Aibo Team; a cooperation of several Dutch universities who were active in the predecessor of the Standard Platform League [14, 17, 7]. The Dutch Aibo Team has been successful, both in the competition and with a number of publications [11, 6, 10, 9, 16, 8]. The Dutch Aibo Team always published their source-code online including a technical report about the innovations [15]. The Dutch Nao Team will continue this tradition. The Dutch Nao Team debuted in the SPL competition at the German Open 2010 [13].

The responsibilities have this year been distributed over our team as follows:

Supervisor: Dr. Arnoud Visser

Coordinator: Duncan ten Velthuis

Vice-coordinators: Camiel Verschoor, Auke Wiggers

Programmers: Auke Wiggers, Chiel Kooijman, Hessel van der Molen, Inge Becht, Maarten de Jonge, Michael Cabot, Richard Pronk, Sander Nugteren, Tijmen Blankevoort, Erik van Egmond, Anna Keune

Sponsors: Placeholder

2 Qualification

This year the Dutch Nao Team made the commitment to travel to Mexico City for the RoboCup 2012. In December a qualification document [5] and qualification video[1] were produced. The qualification document described the research that was to be performed in the upcoming months and how this was to be achieved.

3 Competitions

3.1 Iran Open

A total of six teams competed in the Iran Open Standard Platform League:

1. AUTMan (Iran), first place
2. MRL (Iran), second place
3. Berlin United (Germany), shared third place
4. Dutch Nao Team, shared third place
5. Kouretes (Greece)
6. DreamwingSP

[1]Available at `http://www.youtube.com/watch?v=laCXVb2dtPQ&list=PLCFE0DE27EB5C8C9D&index=1&feature=plpp_video`

3.1.1 Matches

In total 5 matches were played. Two matches were lost, with 0-6 to AUTMan and 0-2 to MRL. The match against Berlin United and Kourettes ended up in a tie (both 0-0), and the match against DreamingSP was won with 1-0. The main problem in the competitions was with the vision. The Naos recognised a ball often outside of the field. The Naos also had problems with connection to the network so communication could not be tested in the matches.

3.1.2 Developments

Due to the above mentioned problems the first steps towards a calibration tool were made for simple color calibration, as well as a bounding box implementation so that the Naos no longer could find a ball outside of the field.

3.1.3 Open Challenge

For the Open Challenge multiple projects were presented that were worked on by members of the Dutch Nao Team, such as a dynamic tree learning localisation approach and the human interface using the Kinect[1] The presentation ended up earning the third place of all the open challenge presentations

3.2 Dutch Open

In the week of april the 22^{nd}, the DNT traveled to Eindhoven to participate in the Dutch Open. There was no real SPL competition at the Open but some demo matches against SPQR were played.

3.3 RoboCup Mexico

Twenty-four teams from all over the world competed in the RoboCup 2011. We've played against a few of them:

1. rUNSwift, lost, 0-10
2. RoboCanes, won by distance in the penalty round
3. The Portugese team, match ended 1-1, lost by penalty round with 1-2

The biggest problem faces in Mexico was the difficulty with connecting to the network (something every team had to try and cope with) but mostly the fact that the state of the code was not so it could be run easily, resulting in crashes of the robots.

3.3.1 Developments

A lot fo the time onely fixes were made to the already existing code.

4 Program

The program used in the RoboCup 2011 is written solely in Python. It is compatible with Python versions 2.61. OpenCV for Python is used, mainly for image processing. The files used are all original files written by members of the Dutch Nao Team. Different files are used for different sections, for instance, motions are all specified in a single file and not in the file containing the main program.

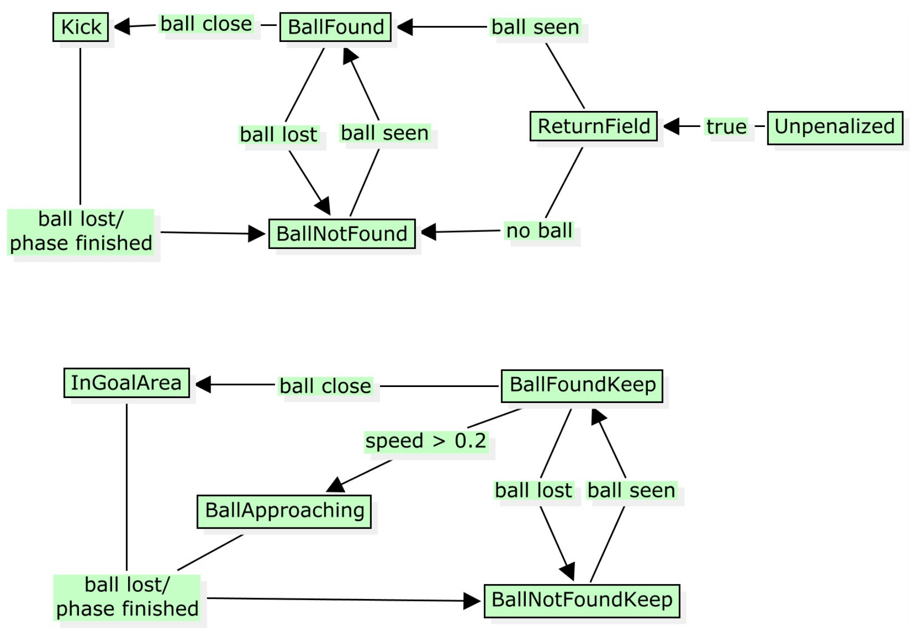


Figure 1: The phases for both player (top) and keeper (bottom) and conditions for transition between phases.

4.1 Architecture

The program solves the game problem using a double state machine in the file `soul.py`. At the highest level are the *states*, which are the game states as specified by the gamecontroller. At a lower level are the *phases*, which are specific situations during a game. Transitions between the states and phases are made based on external input (conditions). For state-transtitions, the external input is a message from the GameController and/or the button interface (see section 4.4). For phase-transitions, the external input is based on observations (see section 4.2).

In each of these phases, checks are made to see if conditions for transition are met (e.g. if in `BallFound` phase and ball close enough for a kick, go to `Kick` phase). An overview of these phases and conditions can be found in figure 1. An advantage of this approach is that the main problem, playing a game of soccer, is divided into smaller problems, which can be solved independently and locally. A second advantage is that phases can be tested without executing the entire program. In effect, the tests are less time-consuming. On the other hand, the Nao is not able to process circumstances that are not specified in a phase. The solution implemented is a straightforward one where phases can not be interrupted until they have finished. This would not, or less so, be the case if a single program covered the entire game problem. A short explanation of every phase in our program:

BallFound
Contains a findBall call to vision, which returns an x- and y-value representing the relative location of the ball (if found). Uses the found location to move Nao towards the ball. Transition conditions: Ball not found or ball close (close being close enough for a kick). How close this is depends on circumstances of the field (lighting changes interpretation of distance, how fast the Nao walks depends on the carpet, etc.)

BallFoundKeep
Similar to `BallFound`. The main differences are that the keeper does not walk towards the ball

and that old positions of the ball are tracked as well. It is possible to derive speed and direction of the ball using these old positions and some geometry. If two or more positions are known, a function through these points $f(y)$ will intersect with the Nao's y-axis in $(y, 0)$. The direction of the ball, which is the direction in which the Nao should dive, is determined by that specific y. An estimate of the speed at which the ball is moving towards the Nao (x-direction) is derived by taking two ballpositions and subtracting the x-positions. Transition conditions: Ball moving towards keeper (speed high enough) or ball close (low speed).

BallNotFound
When a player can not see the ball, the head moves around to scan for it. If, after a full scan, the ball is still not found, Nao turns around. Transition condition: Ball found.

BallNotFoundKeep
Similar to `BallNotFound`, the difference is that the keeper does not turn when a ball is not found. Transition condition: Ball found.

Kick
Player-only phase. Player is close to the ball and scans for goal. Player takes action if a goal is found, action depends on which goal is found. Angle towards goal is used as input for the kicking motion. Transition conditions: Ball lost or phase finished (ball kicked). A special transition was added later: If the ball somehow moves during the kick phase and is too far to kick, move to the BallFound phase.

BallApproaching
Keeper-only phase. Ball moves at high speed towards the keeper in a certain direction `dir`. Keeper takes action depending on `dir` (diving or sidestepping, left or right). Transition condition: Phase finished.

InGoalArea
Keeper-only phase. When the ball is in the goalarea, the keeper walks towards it, kicks it and walks back. As returning to the center of the goal is still impossible for us, this phase is not used in the current program.

Unpenalized
When a player or keeper is unpenalized, it moves to this phase first. It tells the player to walk 2 meters in x-direction (forward). Player/keeper then goes to phase ReturnField.

ReturnField
This phase can only be reached from the `Unpenalized` phase. While the player is walking, search the ball. Transition conditions: Ball found or end of walk.

Keepers behave differently from other players in the field. Therefore, a similar approach is used but with different names for the phases (e.g. `BallFoundKeep` instead of `BallFound`). This shows another disadvantage of our approach: When a bug is found or a change is made in the phases for regular players, it has to be changed twice, once for the normal phases and once for the keeper phases. However, different behaviour for the player and keeper can be made, where this is needed (e.g. emphasize movement along y-axis for keeper to protect the correct side of the goal, whereas a player will emphasize movement along the x-axis to reach the ball faster). A phase contains the following:

- A call to receive visual information and, if needed, further processing of this information
- One or more condition checks for transition to other phases
- One or more motion calls
- Optional: print statements for debugging purposes

The exception to this is the `Unpenalized` phase. This phase cannot be reached from any other phase when in the `Playing` state and it does not ask for visual input. It is reached when a player or keeper was in the `Penalized` state and is unpenalized. Its sole purpose is to make this player then walk two metres forward (in the direction it is facing, this should be towards the field) and thus prevent it from staying at the sideline.

As each phase is implemented as a Python function, it can be called independently. The system thus works as a loop calling the function corresponding to a game phase. It should be noted that the phase can be changed in any state but the corresponding function is only called in the Playing state. Also, a keeper that is penalized will reach the `Unpenalized` and `ReturnField` phase successively, meaning it turns into a regular player after being penalized.

Calls to other aspects of the program (vision and motion) are made through interfaces. These interfaces are specified in different files. DNT chose to use this approach instead of regular motion calls to make the main program (`soul`) easier to understand. An example: Function `StandUp()` in `motionInterface.py` checks the pose of the Nao and then takes action corresponding to this pose. Instead of checking the pose in the main program, this entire process is specified in a different file and can be initiated with a simple function call.

4.2 Vision

The ball- and goal-detection are based on color filters. Prior to a match calibration is needed to set the HSV-color-ranges of the red ball, yellow goal and blue goal. These ranges are then used to filter the image taken by the Nao resulting in a grayscale image. When detecting the ball the filtered image is smoothed to remove any noise. If the brightest pixel passes a threshold then this is assumed to be the ball. The position of the ball is calculated by projecting that pixel onto the ground using the location and orientation of the camera when the snapshot was taken. A small improvement of this ball tracking algorithm of last year is the use of bounding boxes to prevent searching the ball outside of the playing field, by filtering on the color of green (figure 2 shows this smalle improvement).

Figure 2: Using a bounding box approach to minimize the ball search space

This prevents that the Nao will find the ball outside of the field and in environmental noise(which has been often the case in previous matches).

The goal is detected by appending vertical lines in the filtered image to form blobs. These blobs are tested to see if any describes a goalpost. If two posts are found then the angle towards the goal is calculated.

All the vision components were set up so to run parallel with each other. This was removed in the latest release of the program to prevent overheating of the Nao's head.

4.2.1 Blob and Nao detection

In order to recognize features of the environment, a scan is done for blobs of colour in the snapshots the Naos make while walking. To do this, first each picture is corrected for varying lighting conditions. This is done by taking a sample snapshot and finding the pixel that most approximates white. Luckily, since the robots are playing a game of soccer, there will almost always be a white field line somewhere in the snapshot for sampling. The white point is then assumed to be white and all other colors are corrected to make this pure white. This will give consistently coloured snapshots to work with.

Then, the pictures are scanned for colours in a certain range (for example, yellowish colours for the yellow goal). These color ranges are pre-set according to tests, due to correcting these colours, the pictures are consistent enough to make this assumption. A new black and white picture is made representing this scan, with white pixels denoting a positive pixel and black denoting a negative. This gives a black image with the shape of the object that is scanned for in white, and usually some noise. Then each line of the image is checked for lines of one or more white pixels. These pixels represent a blob on a certain line, line blobs. The data on all of these line blobs is stored along with a unique ID value. Due to this line blobs can be easily re used.

Once the entire image is scanned, this will result in a database filled with line blobs. The next step is to convert these line blobs into two-dimensional blobs. This is done by looping over all the line blobs and checking if any of the line blobs, on a line below the current line, are within the horizontal range of the current line blob. If this is the case these line blobs are mapped to one ID value. Once this is done, all the blobs with the same ID are found and the new maximum edges are calculated. This is stored as a new blob with a new ID, along with this data.

To detect a Nao, all insignificantly sized blobs are discarded; this filters most of the less significant noise. Sometimes, faraway Naos are also discarded during this step; however, this is not a real problem since opponents this far from our goal are no immediate threat. This does leave the bigger noise groups in the picture. To determine which of these objects are actual Naos, a check is done on three lines above and two below each blob. If the detected blob does indeed belong to a Nao, each of these lines should be (approximately) white.

4.2.2 Goal Recognition Algorithm

The following algorithm is based on the shape of the goal. The goal is defined by its two poles. The poles in the image are rectangular with a length and a width defined by thresholds. Goal recognition is performed on a color filtered image. The color can be either blue or yellow. In short, to check if the poles are present, the algorithm searches the image in vertical direction for lines long enough to be poles. Then it checks if the lines are adjacent and the width is wide enough to be a goal.

Algorithm step-by-step (the threshold t is 20)

- Filter the image based on color.
    - Form a dict of vertical lines.
    - Iterate from top down steps of ten ($0.5t$) pixels, start at ten.

  - When a true pixel is found go back nine $(0.5t - 1)$ pixels (remember the step of ten).
  - Start iterating by one pixel until a true pixel is found.
  - Then count until twenty (t) pixels are found in a row or until a false pixel is found.
  - When a false pixel is found retry at the last known pixel (ten $(0.5t)$ further).
  - When a true pixel is found do the same but bottom up .
  - Remember starting and ending pixel.
  - If no pixels is found there is no need to start from bottom up.
  - Bottom up searching cannot go past the starting pixel.

- Group the vertical lines together.
  - Iterate from left to right.
  - If the left and right one are adjacent and do not differ in length by more than 50%
  - If a group is found and large enough remember it and find another group.
  - Return the two largest groups.

4.3 Motion

The first steps towards designing a dynamic kick were made this year. A multiple of modules have been constructed to solve this problem [2]. Using a forward kinematics module it now is possible to find the position of a limp relative to one of the legs. This forward kinematics module also gives the possibility to locate the Center of Mass and is used in a Center of Mass based P-controller to keep the Center of Mass above the Support Polygon. A second balancer uses online data in the foot sensors to correct outside interferences from other Naos during a competition.

Another critical part of dynamic motion that has been mastered is the Inverse Kinematics. This has been approached by the damped least squares method, in which the desired position is approached over multiple iterations. Due to time constraints the module is written in C++ instead of our usual Python but uses a Python SWIG wrapper to easily communicate with the rest of the Python code.

At the moment all of the motions used in matches are still keyframe motions. The kick and back kick are given an angle (angle to the goal) as input. The result is an open-loop motion kicking the ball in the specified angle. The keeper movements are of our own design. We use the walk engine by Aldebaran which is a stable omnidirectional walk. Get-up motions are improved versions of the motions provided in Choreographe.

4.4 Communication

At first, the program was tuned to listen to a specific IP address. It would receive the input stream, parse it, and store the information in a GameController object. Recieving the data was done by the *refresh()* method, and the object had several get methods to acquire the data from the object. The refresh method returned true if it had recieved data, and false if it hadn't.

In the Rome Open, the program was made to display the right LEDs on the chest and feet, depending on the current team color and the game state (green for playing, red for penalized, etc.).

After the Rome Open, the button interface was implemented and combined with the game controller to make a state controller. This is a threaded module which is called to get the game state according to the buttons and game controller (if game controller is present).

Furthermore, the *controlledRefresh* method was implemented, which calls the refresh method repeatedly during a specified interval to see if the game controller is still online (default 4 seconds). If it is, it returns true. If the connection timed out, it returns false, and the state controller starts listening to the buttoninterface, before trying to connect with the game controller again.

In Istanbul it occured, that some teams were broadcasting their own game controllers on the same IP address, which caused problems as the program would listen to all of them. The solution to this was making the program listen to game controllers broadcasting a match with our team number in it only.

Communication between Naos has been proven to work successfully using built-in methods provided by Aldebaran. It is possible to store a string in memory of a Nao. Other Nao's can receive this specific string if the IP adress is known. This communication will be used to 'tell' other Naos how close to the ball they are. Visual information, once processed, can be sent through use of a (not yet determined) protocol, e.g. 'Ball 0.500 0.300 0.583' would mean a ball is at relative position (0.5, 0.3) at distance 0.583 meters.

4.5 File architecture

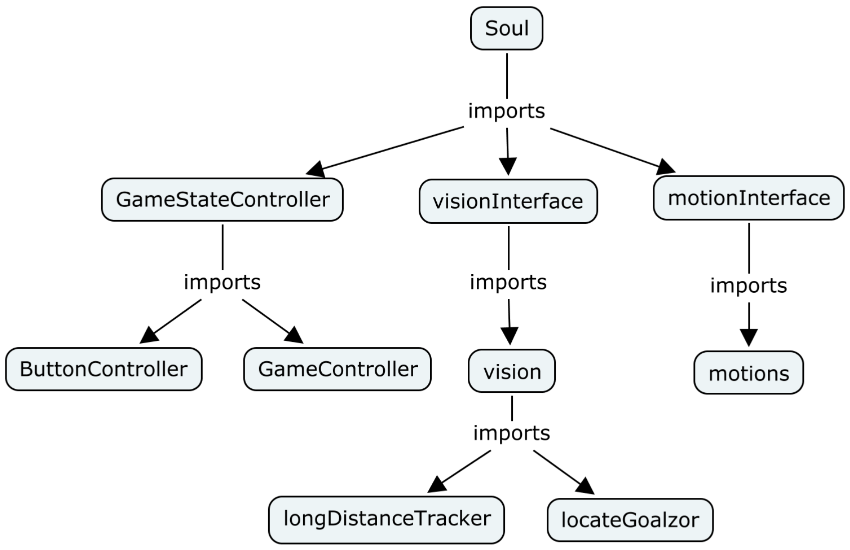


Figure 3: Hierarchy of files as used in the RoboCup 2011

The file hierarchy (see figure 3) that is used prevents the main program (`soul.py`) from becoming a chaos. The contents of the files are described in more detail in Appendix 12.5. Motions are specified in a single file. For different visual tasks, different files are used. This is mainly to make testing easier (tasks can be tested and programs can be altered simultaneously without trouble due to incompatible versions). To make matters even easier, interfaces (`motionInterface` and `visionInterface`) are used. These interfaces are used because almost every vision task starts with a snapshot and conversion of this image. It would not be aesthetically pleasing to make these function calls in the main program every time a vision call is made. The vision interface is well used; the motion interface is implemented merely for completeness (as vision is comparable to motion, both are straightforward tasks that do not alter the flow of the main program directly, unlike the game controller and button interface). The game controller and button interface are each specified in a file to make testing and altering individual functions easier.

4.6 Simulation

At the Universiteit van Amsterdam a physical realistic model of an Aldebaran Nao robot inside the simulation environment USARsim [4] is developed. A major advantage of this simulator is that uses the local installation of NaoQiSDK to process commands to the Nao robot, which means that the same python code can be used for the real robot and the simulated robot.

It was even considered to use a NaoQi system image inside a virtual machine to circumvent the usage of a local NaoQiSDK, but private discussions with Aldebaran employees[2] on the RoboCup revealed that Aldeberan itself is busy with this development for their open source release in Fall 2011.

USARsim is a simulation environment based on the Unreal Engine, which enables easily modification of the environment around the robot (in contrast with Cogmation's NaoSim) and create realistic lighting condition.

In the previous version of USARsim (based on UT2004) a wide variety of robot models was available, including a Sony Qrio and Aibo. The model of the Nao robot is the first type of walking robot in the new version (based on Unreal Development Kit).

First a Nao model with two joints (T2) was developed, which was gradually upgraded to a model with 21 joints (T21). The different parts are now nicely scaled, as illustrated in Fig. 4.

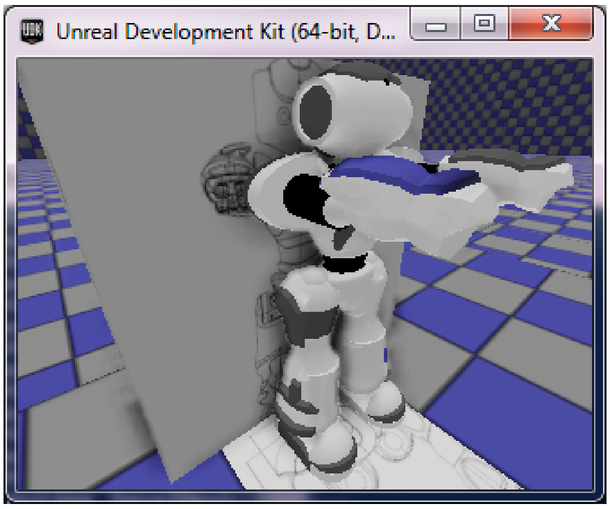


Figure 4: A model of an Aldebaran Nao robot in USARsim, including a scaled drawing from the Aldebaran documentation

Important for a physical realistic simulation is the calibration of the joints' dynamic behaviors. The dynamic behaviors are configured with parameters such as the mass distribution, the inertia and the gravity.

Several tests related with gravity, forces and constraints have been performed to validate basic assumptions about the physics of the simulated world. A nice feature of the UDK-model is that the double constraint for the HipYawPitch joint is seemlessly incorporated into the model.

With the center of mass close to the hip, representing a mass distribution according to Aldebaran's specifications, the robot currently can stand up and sit-down. Even more complex behaviors as the Tai Chi and Thriller dance can be executed without loosing balance.

Currently the walking behavior is fine tuned by letting the real and simulated robot walk an eight shaped figure. In the near future the complexity of the tests will be increased, including more complex movements of the Dutch Nao Team.

[2]Manuel Rolland and Cedric Gestes

The development of this open source simulation of a Nao robot not only valuable inside the Standard Platform League, but should also be interesting for the Soccer Simulation League and the @ Home League. Outside the RoboCup community this simulation could be valuable for Human-Robot Interaction research.

4.7 Tools

Some various tools have been developed to ease preparation before a match.

4.7.1 Calibation Tool

One thing that proved to be a problem in tournaments was color calibration. Previously this was done manually by taking samples of an image and kind of estimating what the color range would be. This led to a lot of inaccuracy and difficulty in finding the ball and goalposts. This has been alleviated with a graphical calibration tool written in C++ with Qt. It allows the user to load an image and select an area, either through dragging rectangles or a magic wand tool. It then ouputs the HSV range (minimum and maximum values for the hue, saturation and value) of the select area(s) (see image **??**).

4.7.2 Preparation Script

To ease the preparation for a match, a Python script has been made to find all Nao robots on the network using Avahi and automatically push code and a generated naoinfo.txt.

5 Installation Guide

5.1 Requirements

5.1.1 Program

The following items are required to run our program `soul.py`:

- Nao
- NaoQi 1.10.52
- Computer (Windows/Linux)
- Secure Shell Client (Windows) / SSH (Linux)
- TurtoiseSVN (Windows) / SVN (Linux)
- Link to repository[3]

5.1.2 Simulation

The simulation environment consist of two parts: UsarSim which represents the SoccerField (or another world) where one or multiple robots can be spawned and UsarNaoQi which represents the robot (and translates NaoQi commands into usarsim commands). For UsarSim an installation guide is available[4]. For UsarNaoQi an usage guide is available[5].

[3] `http://code.google.com/p/dnt-rules/`
[4] `http://sourceforge.net/apps/mediawiki/usarsim/index.php?title=Installation`
[5] `http://nao.sandern.com/doku.php?id=usage`

The UDK version of UsarSim can be downloaded from sourceforge with the command `git clone git://usarsim.git.sourceforge.net/gitroot/usarsim/usarsim`, for instance from a Cygwin shell. To allow the usage of the two cameras of the Nao a tool is needed, which is (not yet) available under git. An update of this tool which uses the video hook (EasyHook) native to UDK to get access of the camera images is available for download[6]. This tool should be installed in the directory USARTools.

UsarNaoQi is available for download at `http://svn.sandern.com/nao/trunk/usarnaoqi`. The code requires an installation of Visual Studio 2008 and an installation of NaoQiSDK. The location of the NaoQiSDK installation should be specified in the environment variable `AL_DIR`.

The latest version of UsarNaoQi is tested with NaoQiSDK 1.10.52.

5.2 Guide

5.2.1 Program

Follow these steps to install and run the program `soul.py`:

1. Download the program from our repository with a SVN program.
2. Move downloaded files from the previous location to the home folder of the Nao (Linux environment).
3. Navigate to folder 'home\naoinfo' (Windows) or '/home/naoinfo/' (Linux).
4. Create a file 'naoinfo.txt', line 1 is the playernumber, line 2 the teamnumber. Line 1 should be 1 if Nao is a keeper.
5. Navigate to 'home\naoqi\lib' (Windows) or '/home/naoqi/lib' (Linux)
6. Call 'python loader.py' from command line.
7. Nao will say that you have to press a footbumper to initiate football mode. Press the bumper.

In this release, there is no safe way to stop the program. Rebooting the Nao or forcing a stop in command line both work but are not pretty solutions to the problem.

5.2.2 Simulation

To start the UsarSim including support for the cameras the following command should be given from the UDK directory: `Binaries\Win64\udk.exe RoboSoccer?game=USARBotAPI.ImageServerGame -log -d3d11"`.

To spawn a Nao in this RoboSoccer environment the command `usarnaoqi.exe` should be given. This program loads its default configuration from the file defaultconfig.xml.

The robot can now be controlled and monitored by any control program, including Aldebaran's Choregraphe and TelePathe (see Fig. 5). The Nao is visible with the default name 'UsarSim1' in the connection windows of both Aldebaran programs.

The simulated robot can also be controlled with the Dutch Nao Team software by typing `python loader.py` on the commandline.

[6]`http://nao.sandern.com/downloads/ImageServer_Beta3.zip`

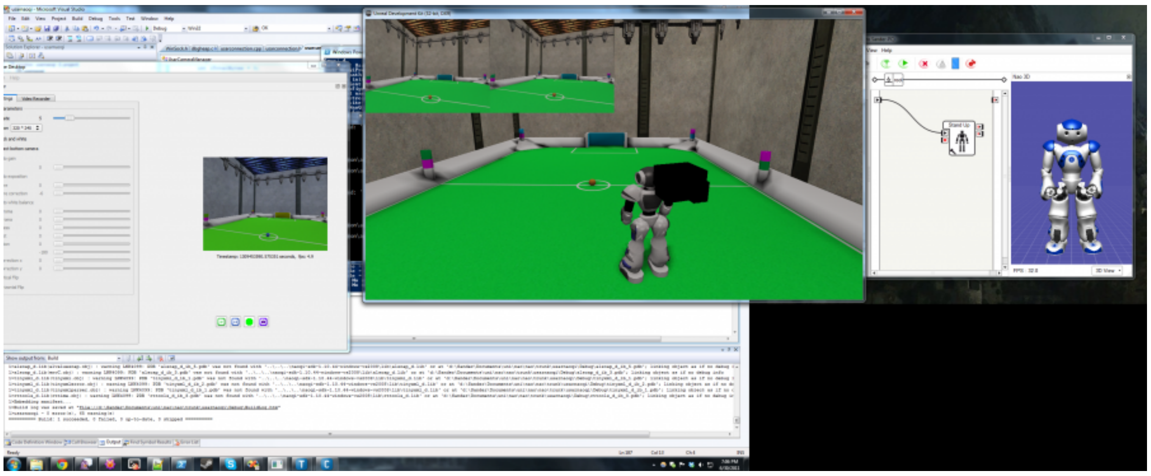


Figure 5: The Nao robot in the RoboSoccer environment of USARsim, including the view of both cameras in the upper left corner.

6 Sponsors

The Dutch Nao Team consists of students and most of them do not have enough money to pay for the travel to the Open competitions or the World RoboCup. Therefore, it is essential to get some sponsors. There are many companies that are eager to get some brand awareness and they are prepared to pay for it. It costs time and energy to find the right people from the right companies, but if it is done well, it might lead to a fine collaboration.

6.1 Preparations

There are a few things that the team needs to make some decisions on, before somebody starts contacting companies. Firstly, its necessary to know what the team has to offer; banner on the team's website, logo on the t-shirts or on the Nao's etc... Secondly, when the list is ready with all the possibilities that a company may use, prices for the individual options have to be determined. Sometimes people may ask whether there are packages (a few options packed together for a better price) that can be bought. Therefore, it is recommended to make some small packages (consisting of two or three options) for an attractive price, some normal packages and one large package, which contains everything that is offered by the team. This is due to the fact that every company has a different amount of money that they can spend on sponsoring. A larger company may be willing to pay more in contrary with smaller companies, that will try to get a better deal. After making the packages it is recommended to prepare some contracts. This is to prevent that the company has to wait. It is up to the person who takes care of the sponsoring to know whether to go along with the negotiations or not. When the company is very small, they will not be able to pay as much money as the other companies, but every amount can be helpful fot the team. So it is good to consider your decision about the sponsor.

Another task is to find the companies that might be interested in sponsoring the team. The first choices should be Information Technology (IT) companies, because those are the companies that are primiraly looking for IT students. The internet is the best place to look for them and also to find another businesses that might want to collaborate with the team.

6.2 Contacting companies

When all the preparations are finished one can start contacting the companies that have been found. It is highly recommended that every company communicates with one person. This in order to prevent double deals. It also looks unprofessional if more than one person try to contact one person in a company. When a company is interested one should negotiate about what the company wants exactly. Then the contract can be set up and signed.

7 Public Relations

7.1 Getting started

The Dutch Nao Team is one year old now, which means the Public Relations (PR) department started from scratch, with no script, site or reference to a tournament the Dutch Nao Team had participated in. First of all, the team started with the most general concern of PR: launching the website. Given a website should appeal to a wide variety of people so it should not only describe technical terms, but also introduce the RoboCup and ourselves on a basic level. In addition, the site had to be a clear and easy to use, so everyone can find what they are looking for. To achieve this, a simple menu was made based on the subjects one might wonder about. Furthermore, the language of the site was English instead of Dutch, because of the international character of the RoboCup competition.

Secondly, the communication advisor of the UvA, drs. M. van de Laar, was contacted, who came up with a general script voor PR, which is described in the following section.

7.2 Making the script

7.2.1 The script in general

Public Relations is all about improving reputation and image. To achieve this, you will need a script revolving the following questions:

- Audience: Who does the team want to reach?
- Objective: What does the team want to achieve?
- Message: What does the team want to communicate?

7.2.2 Audience

The RoboCup is not very well known outside the robotics community, therefore, choosing a target audience was difficult. The goal was to reach (prospective) students in the teams field to show the possibilities of robotics and perhaps even to recruit new members. Another goal was to promote the RoboCup in general to a bigger audience. The team wanted to address the latter because it would help reaching prospective students and perhaps even sponsors. It was also important to keep the robotics section of the UvA up to date, because without them there would not even be the RoboLab.

7.2.3 Objective

In the Netherlands only a few people are familiar with the competition. The objective is thus to promote the reputation of the DNT and promoting the RoboCup. However, it is neither the task nor the responsibility of the DNT to provide a campaign for the RoboCup in general. Nonetheless, the ignorance is something to keep in mind when promoting the team.

7.2.4 Message

In short, the message for every group, comes down to:

- (Prospective) Students: See what the Dutch Nao Team does in their field and join!
- Public in general: Learn about the possibilities of robotics!
- UvA: Follow the progress of the team!

7.3 Achievements

For the past year we have mainly been working on the website and a general view of PR. The biggest obstacle was to decide where to start. Trying to reach our entire audience at once was too much to handle and on top of that we needed different approaches for each one.

- For our junior prospective students we gave a demonstration at the RoboCup Junior in science center NEMO. This was done to show youngsters the possibilities of robotics.
- During the RoboCup tournaments everybody wore printed t-shirts to promote the team and improve our reputation within the league.
- At the Science Park there were posters of the Dutch Nao Team with information about us, our demonstrations, our progress and of course a reference to our website!
- To keep contact with our supporters a Facebook-page and a Twitter-account were made. Through these, people from all over the world can follow our progress, thoughts and experiences.
- And finally, to address the bigger audience, we made a press release about the participation of the DNT in the World RoboCup in Istanbul. We chose this event for the press release because a world cup generally is newsworthy for its international character. This particular press release can be found in Appendix 12.4.

7.4 Points of improvement for next year

Next year it would be very wise to start with expanding the script to a much more detailed version. Last year we fiddled about for too long because our ideas were not specific enough and also: sending general messages to an unspecific audience is not efficient or functional at all.

In addition, the goal to promote the reputation of the RoboCup in general is a job on its own, so it is something to bear in mind when you are writing to people outside the robotics community, but it is definitely not up to us to manage their PR!

On top of the press releases and emails, it is time to make a new, more professional website now that we have finally accomplished something and want it to last.

Furthermore, it is a new years resolution to attend more events for speeches as well as demonstrations. And last, but not least, we would like to refer to the following site: `http://www.freepublicity.nu/`. Unfortunately we ran across this website at the end of this past year, but you can find lots of tips and trics there for the PR department.

8 Travel

The Dutch Nao Team travelled this year to Iran and Mexico to participate in competitions. For every journey, several arrangements had been made. This section describes, which arrangements were made.

8.1 Teheran

This was a special journey as the Dutch Nao Team got invited by the organizing party (Qazvin University) and the costs for travel and stay were fully covered by them. Therefore, only the flight had to be booked and a visa had to be requested. It is important to do these two on time as both require some time. For the visa, the required documents have to be sent to the organizing committee, which will respond to you when the visa can be requested at the consulate in Den Haag. The organizing committee will make an official invitation letter for the participating members. To request the visa in Den Haag all participating members have to be present at the consulate. Requesting a visa can take up to one day. The flight was booked after a confirmation of the organizing committee was received. Before leaving, a document [12] of all important facts about the country was sent to all participating members. This was to make sure that participating members were informed about the background of the country, possible required vaccinations, laws and uses of the inhabitants and advice about staying in the country. In Iran, the stay, transport, food, tours of the city were all organized by the committee.

8.2 Mexico

This journey was to the RoboCup2012 where all international teams participated in a robot football World Cup tournament. The flight and hotel were booked quite late causing the price of the flight to be higher. A nice hotel was found nearby the location of the event (name of the hotel is El Diplomatico (located near the World Trade Center where the RoboCup was held). Breakfast and lunch was bought from the nearby supermarket. Dinner was bought and consumed at nearby restaurants. We traveled mostly by foot to the Robocup revenue, and sometimges using public transport. The first week of the journey was spend sightseeing around Mexico-city.

8.3 Tips for the next journey

This section describes the advice to the organizers of next year. The following points are important to take into account for next year:

- Have strict deadlines for participating students. Pay and sign up deadlines are both important as without money you cannot book the flight and hotel.
- Start on time with booking the flight to reduce costs.
- Find a hostel/hotel near the location of the event.
- Write a travel information document. So all the important information bundled together.
- Communicate clearly with the students participating in the team.

9 Public Relations

10 Evaluation and Future Improvements

The Dutch Nao Teams second year has made some progress from the previous year. The first steps towards closed loop motions has been explored, although not yet implemented into the code at this point. Some tools were developed to ease the setup before a game, making it less of a hassle to ready the Naos before the match.

10.1 Future work

The Dutch Nao Team would like to further develop clodes loop motions, either by making a closed loop gait or create a fully dynamic kick. Falling during a kick was one of the main problems in this years RoboCup. A faster gait would be an important improvement because this teams robots are currently the slowest in the competition.

Further the idea of rewriting all code into C++ is a possibility.

Line detection was not further explored this year, but would be a good to add to the arsenal.

10.1.1 Line Detection

To find a line on the field, an image is filtered on white pixels (lines are white). In this filtered image, the Houghlines algorithm, found in OpenCV [3, p. 156], can find lines. Using (intensity) thresholds, lines with corresponding accumulator value greater than the threshold are returned. It is also possible to input a minimal required length for a line. When a line is partially invisible, the algorithm can reconstruct the line. An additional parameter is the maximum length a gap between lines can be before its not considered to be the same line. Unfortunately, the Houghlines algorithm returns a lot of redundant lines. This is caused by the different thicknesses of lines in a 3D world projected onto a 2D plane. A different threshold for lines further away than lines close by is needed, as currently, the same line close by looks thinner when its further away. The problem that that causes is that the algorithm finds multiple lines on a single line simply because it is so thick that it is not considered to be the same one. To reconstruct these thicker lines, lines that are close to each other, with the same angle, are appended to form a single line. Corners of the field and T-junctions can be found using the data the Houghlines algorithm returns. The Nao can use these as landmarks for localization. The angles found for T-junctions will also contribute to the calibration of the camera, because the 2 lines should always be perpendicular in the real world. The angle between the two lines that form a T-junction found through this algorithm will not always be 90 degrees. Therefore, this angle can be used to recalibrate the camera in order to maximize accuracy of localization.

11 Acknowledgements

The Dutch Nao Team likes to thank the following people:

- Sander van Noort for his contribution to the simulation.
- The University of Amsterdam for their support, the new lab, the contributions and experience we get from this project.
- All our Artificial Intelligence teachers for when they took RoboCup into account.
- Adwin Verschoor of Cena-Werk[7] providing us with full technical support.
- The Italian and Iranian National Committee for their hospitality and travel support
- RoboCup Junior for a great event
- Marthy van de Laar for her help with Public Relations
- The Standard Platform League for blessing us with the Robocup.

Last but not least we would like to thank: Sander Latour, Patrick de Kok, Chiel Kooijmans, our parents and everybody that has helped for their contributions.

[7]Visit him at `http://www.cenawerk.nl` for computer support

References

[1] V. et al., “Technical Report DNT”, University of Amsterdam, 2011.

[2] I. G. Becht, M. de Jonge and R. Pronk, “A Dynamic Kick for the Nao Robot”, project report, July 2012.

[3] G. Bradski and A. Kaehler, *Learning OpenCV*, O’Reilly Media Inc., 2008.

[4] S. Carpin, M. Lewis, J. Wang, S. Balakirsky and C. Scrapper, “USARSim: a robot simulator for research and education”, in “Proceedings of the 2007 IEEE Conference on Robotics and Automation”, pp. 1400–1405, 2007.

[5] Dutch Nao Team, “Team Description for RoboCup 2012”, january 2012.

[6] M. Liem, A. Visser and F. Groen, “A Hybrid Algorithm for Tracking and Following People using a Robotic Dog”, in “Proceedings of Third International Conference on HumanRobot Interaction (HRI 2008)”, pp. 185–192, 2008.

[7] S. Oomes, P. Jonker, M. Poel, A. Visser and M. Wiering, “Dutch AIBO Team at RoboCup 2004”, in “Proceedings CD of the 8th RoboCup International Symposiums”, July 2004.

[8] B. Slamet and A. Visser, “Purposeful perception by attention-steered robots”, in “Proceedings of the 17th Dutch-Belgian Artificial Intelligence Conference”, pp. 209–215, October 2005.

[9] D. van Soest, M. de Greef amd Jrgen Sturm and A. Visser, “Autonomous Color Learning in an Artificial Environment”, in “Proceedings of the 18th Dutch-Belgian Artificial Intelligence Conference”, pp. 299–306, October 2006.

[10] J. Sturm, P. van Rossum and A. Visser, “Panoramic Localization in the 4-Legged League: Removing the dependence on artificial landmarks”, in “RoboCup 2006: Robot Soccer World Cup X”, pp. 185–192, October 2007.

[11] J. Sturm and A. Visser, “An appearance-based visual compass for mobile robots”, in “Robotics and Autonomous Systems”, pp. 536–545, 2009.

[12] C. R. Verschoor, “Iran Travel Document”, Information document, March 2012.

[13] A. Visser, R. Iepsma, M. van Bellen, R. K. Gupta and B. Khalesi, “Dutch Nao Team - Team Description Paper - Standard Platform League - German Open 2010”, January 2010.

[14] A. Visser, P. van Rossum, J. Westra, J. Sturm, D. van Soest and M. de Greef, “Dutch AIBO Team at RoboCup 2006”, in “Proceedings CD of the 10th RoboCup International Symposium”, June 2006.

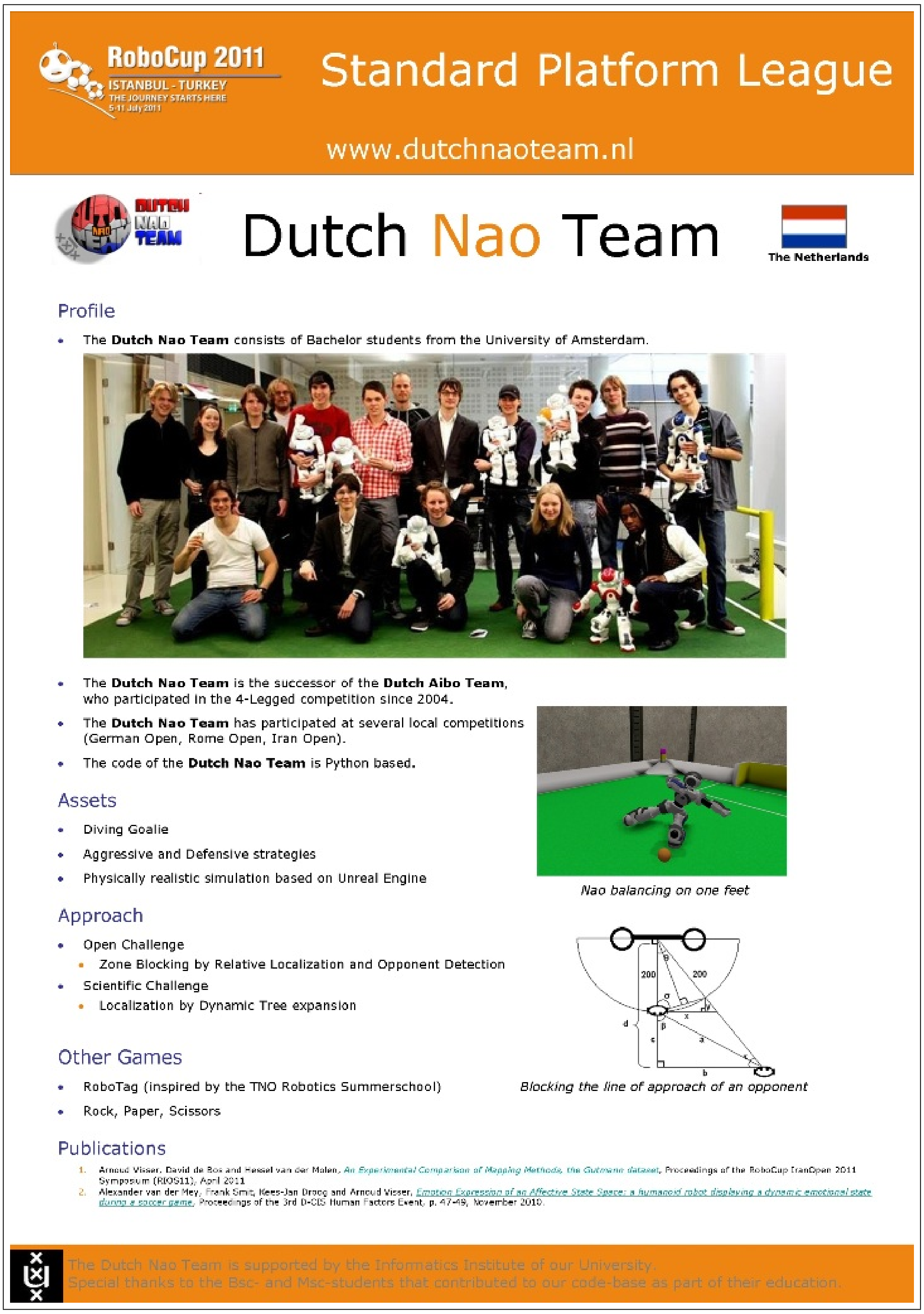
[15] A. Visser, P. van Rossum, J. Westra, J. Sturm, D. van Soest and M. de Greef, “Dutch AIBO Team at RoboCup 2006”, Team description paper for the 10th RoboCup International Competition, June 2006, Bremen, Germany, June 2006.

[16] A. Visser, J. Sturm and F. Groen, “Robot companion localization at home and in the office”, in “Proceecings of the 18th Dutch-Belgian Artificial Intelligence Conference”, pp. 347–354, October 2006.

[17] N. Wijngaards, F. Dignum, P. Jonker, T. de Ridder, A. Visser, S. Leijnen, J. Sturm and S. Weers, “Dutch AIBO Team at RoboCup 2005”, in “Proceedings CD of the 9th RoboCup International Symposium”, July 2005.

12 Appendix

12.1 Team Poster



RoboCup 2011
ISTANBUL - TURKEY
THE JOURNEY STARTS HERE
5-11 July 2011

Standard Platform League

www.dutchnaoteam.nl

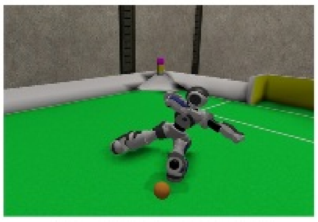
Dutch Nao Team

The Netherlands

Profile

- The **Dutch Nao Team** consists of Bachelor students from the University of Amsterdam.

- The **Dutch Nao Team** is the successor of the **Dutch Aibo Team**, who participated in the 4-Legged competition since 2004.
- The **Dutch Nao Team** has participated at several local competitions (German Open, Rome Open, Iran Open).
- The code of the **Dutch Nao Team** is Python based.



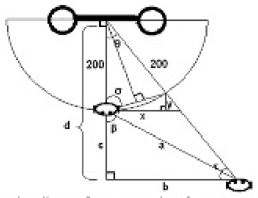
Nao balancing on one feet

Assets

- Diving Goalie
- Aggressive and Defensive strategies
- Physically realistic simulation based on Unreal Engine

Approach

- Open Challenge
  - Zone Blocking by Relative Localization and Opponent Detection
- Scientific Challenge
  - Localization by Dynamic Tree expansion



Blocking the line of approach of an opponent

Other Games

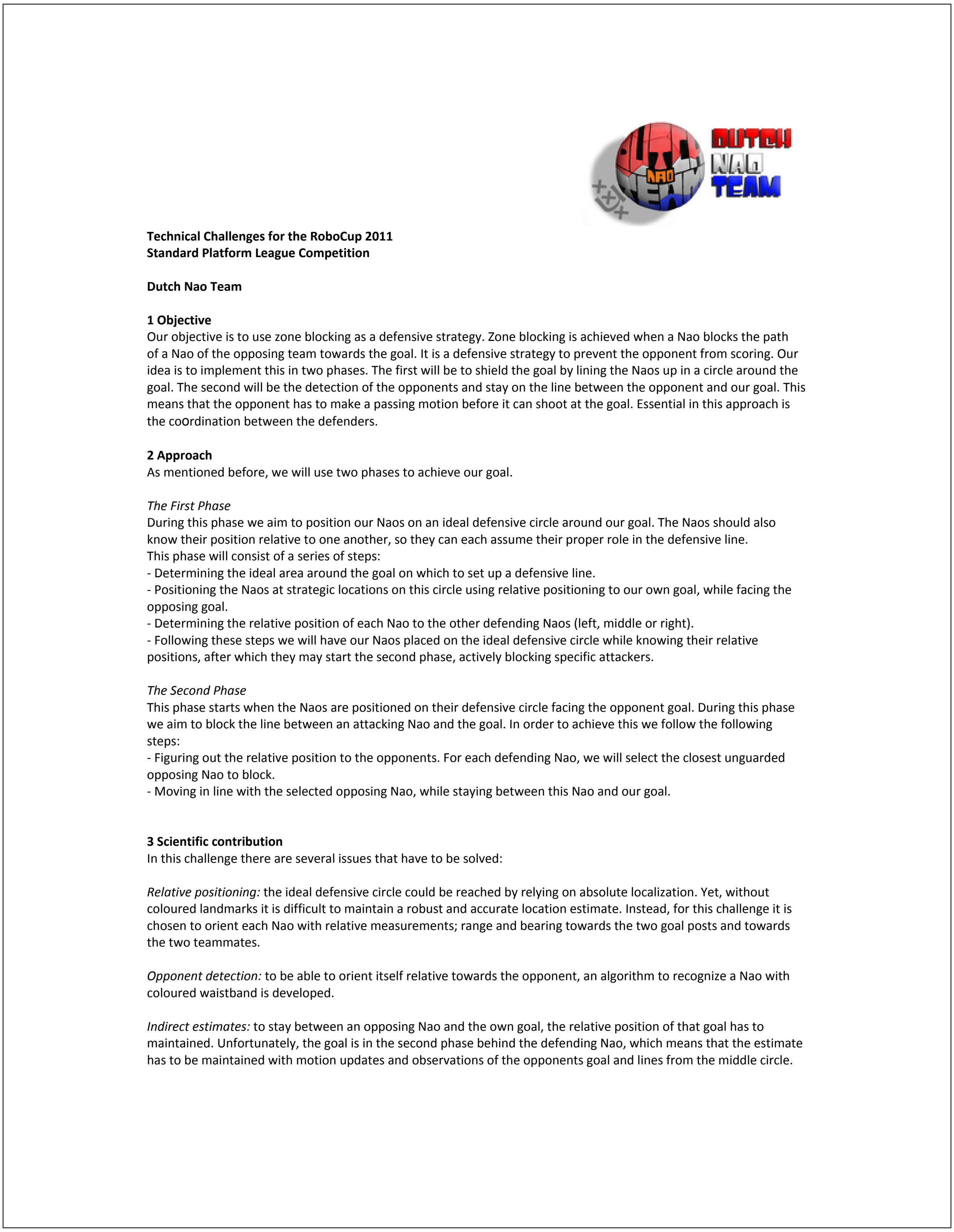
- RoboTag (inspired by the TNO Robotics Summerschool)
- Rock, Paper, Scissors

Publications

1. Arnoud Visser, David de Bos and Hessel van der Molen, *An Experimental Comparison of Mapping Methods, the Gutmann dataset*, Proceedings of the RoboCup IranOpen 2011 Symposium (RIOS11), April 2011
2. Alexander van der Mey, Frank Smit, Kees-Jan Droog and Arnoud Visser, *Emotion Expression of an Affective State Space: a humanoid robot displaying a dynamic emotional state during a soccer game*, Proceedings of the 3rd D-CIS Human Factors Event, p. 47-49, November 2010.

The Dutch Nao Team is supported by the Informatics Institute of our University.
Special thanks to the Bsc- and Msc-students that contributed to our code-base as part of their education.

12.2 Open Challenge Document June



Technical Challenges for the RoboCup 2011
Standard Platform League Competition

Dutch Nao Team

1 Objective
Our objective is to use zone blocking as a defensive strategy. Zone blocking is achieved when a Nao blocks the path of a Nao of the opposing team towards the goal. It is a defensive strategy to prevent the opponent from scoring. Our idea is to implement this in two phases. The first will be to shield the goal by lining the Naos up in a circle around the goal. The second will be the detection of the opponents and stay on the line between the opponent and our goal. This means that the opponent has to make a passing motion before it can shoot at the goal. Essential in this approach is the coördination between the defenders.

2 Approach
As mentioned before, we will use two phases to achieve our goal.

The First Phase
During this phase we aim to position our Naos on an ideal defensive circle around our goal. The Naos should also know their position relative to one another, so they can each assume their proper role in the defensive line.
This phase will consist of a series of steps:
- Determining the ideal area around the goal on which to set up a defensive line.
- Positioning the Naos at strategic locations on this circle using relative positioning to our own goal, while facing the opposing goal.
- Determining the relative position of each Nao to the other defending Naos (left, middle or right).
- Following these steps we will have our Naos placed on the ideal defensive circle while knowing their relative positions, after which they may start the second phase, actively blocking specific attackers.

The Second Phase
This phase starts when the Naos are positioned on their defensive circle facing the opponent goal. During this phase we aim to block the line between an attacking Nao and the goal. In order to achieve this we follow the following steps:
- Figuring out the relative position to the opponents. For each defending Nao, we will select the closest unguarded opposing Nao to block.
- Moving in line with the selected opposing Nao, while staying between this Nao and our goal.

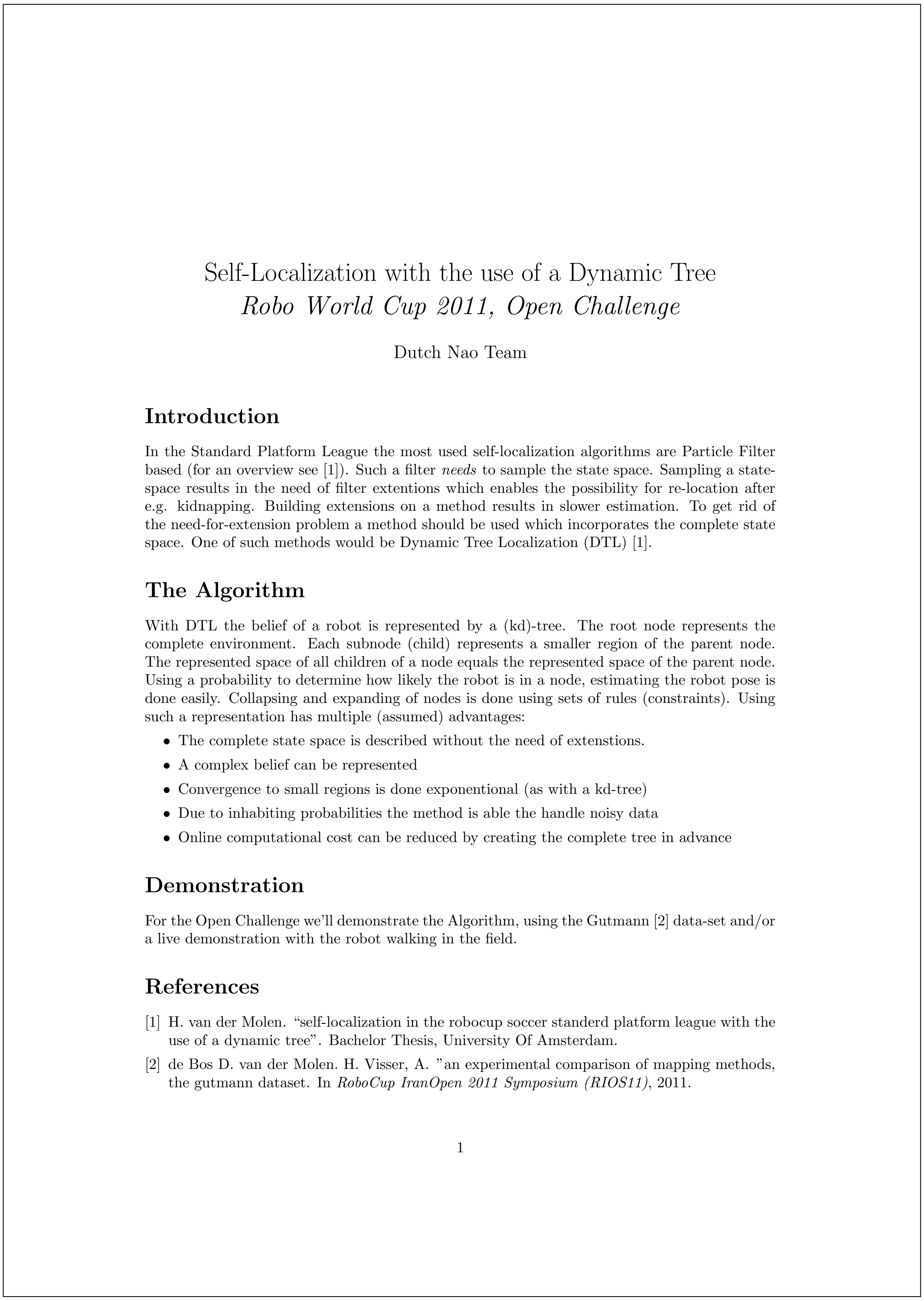
3 Scientific contribution
In this challenge there are several issues that have to be solved:

Relative positioning: the ideal defensive circle could be reached by relying on absolute localization. Yet, without coloured landmarks it is difficult to maintain a robust and accurate location estimate. Instead, for this challenge it is chosen to orient each Nao with relative measurements; range and bearing towards the two goal posts and towards the two teammates.

Opponent detection: to be able to orient itself relative towards the opponent, an algorithm to recognize a Nao with coloured waistband is developed.

Indirect estimates: to stay between an opposing Nao and the own goal, the relative position of that goal has to maintained. Unfortunately, the goal is in the second phase behind the defending Nao, which means that the estimate has to be maintained with motion updates and observations of the opponents goal and lines from the middle circle.

12.3 Open Challenge Document July



Self-Localization with the use of a Dynamic Tree
Robo World Cup 2011, Open Challenge

Dutch Nao Team

Introduction

In the Standard Platform League the most used self-localization algorithms are Particle Filter based (for an overview see [1]). Such a filter *needs* to sample the state space. Sampling a state-space results in the need of filter extentions which enables the possibility for re-location after e.g. kidnapping. Building extensions on a method results in slower estimation. To get rid of the need-for-extension problem a method should be used which incorporates the complete state space. One of such methods would be Dynamic Tree Localization (DTL) [1].

The Algorithm

With DTL the belief of a robot is represented by a (kd)-tree. The root node represents the complete environment. Each subnode (child) represents a smaller region of the parent node. The represented space of all children of a node equals the represented space of the parent node. Using a probability to determine how likely the robot is in a node, estimating the robot pose is done easily. Collapsing and expanding of nodes is done using sets of rules (constraints). Using such a representation has multiple (assumed) advantages:

- The complete state space is described without the need of extenstions.
- A complex belief can be represented
- Convergence to small regions is done exponentional (as with a kd-tree)
- Due to inhabiting probabilities the method is able the handle noisy data
- Online computational cost can be reduced by creating the complete tree in advance

Demonstration

For the Open Challenge we'll demonstrate the Algorithm, using the Gutmann [2] data-set and/or a live demonstration with the robot walking in the field.

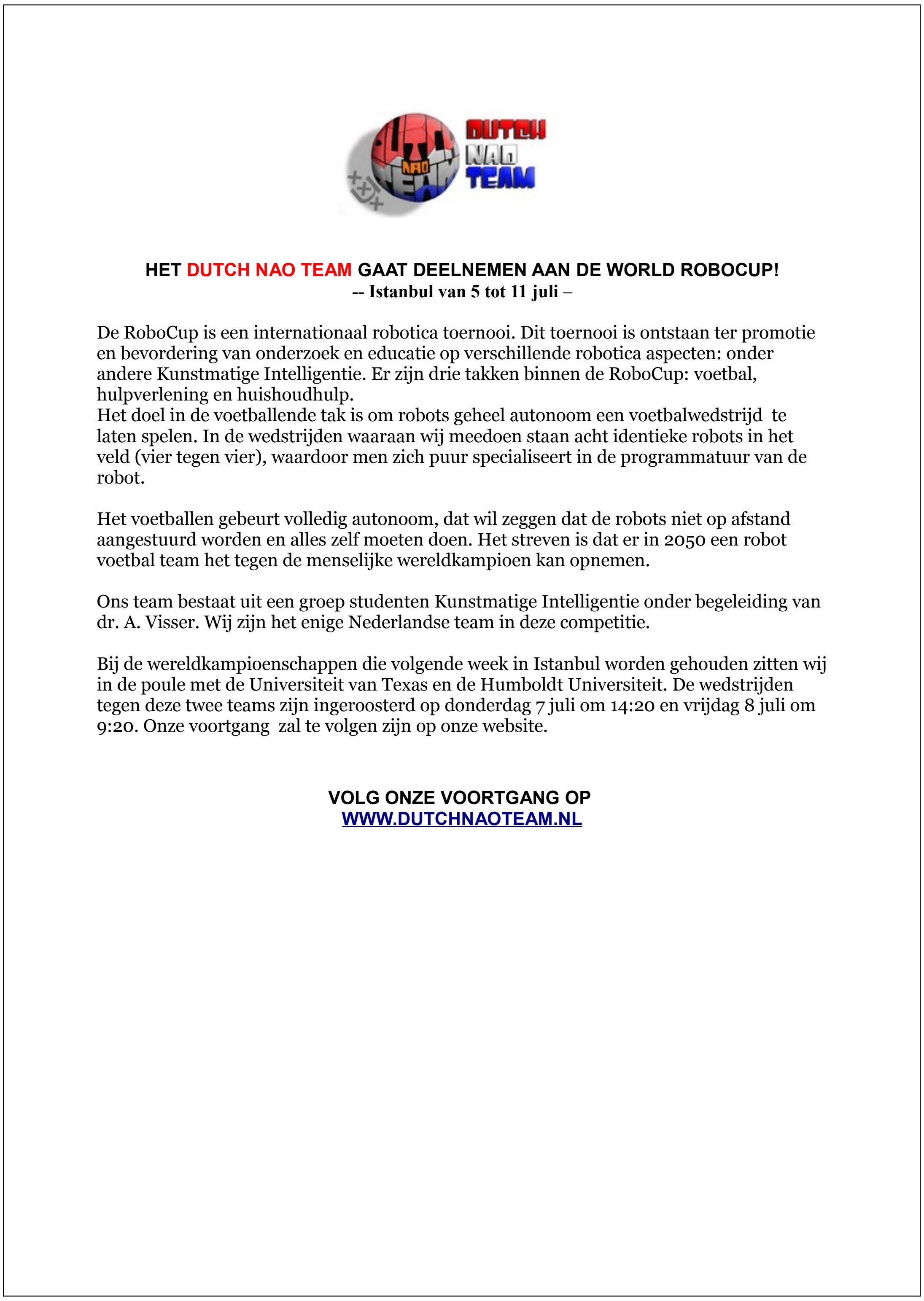
References

[1] H. van der Molen. "self-localization in the robocup soccer standerd platform league with the use of a dynamic tree". Bachelor Thesis, University Of Amsterdam.

[2] de Bos D. van der Molen. H. Visser, A. "an experimental comparison of mapping methods, the gutmann dataset. In *RoboCup IranOpen 2011 Symposium (RIOS11)*, 2011.

1

12.4 Press release



HET DUTCH NAO TEAM GAAT DEELNEMEN AAN DE WORLD ROBOCUP!

-- Istanbul van 5 tot 11 juli –

De RoboCup is een internationaal robotica toernooi. Dit toernooi is ontstaan ter promotie en bevordering van onderzoek en educatie op verschillende robotica aspecten: onder andere Kunstmatige Intelligentie. Er zijn drie takken binnen de RoboCup: voetbal, hulpverlening en huishoudhulp.
Het doel in de voetballende tak is om robots geheel autonoom een voetbalwedstrijd te laten spelen. In de wedstrijden waaraan wij meedoen staan acht identieke robots in het veld (vier tegen vier), waardoor men zich puur specialiseert in de programmatuur van de robot.

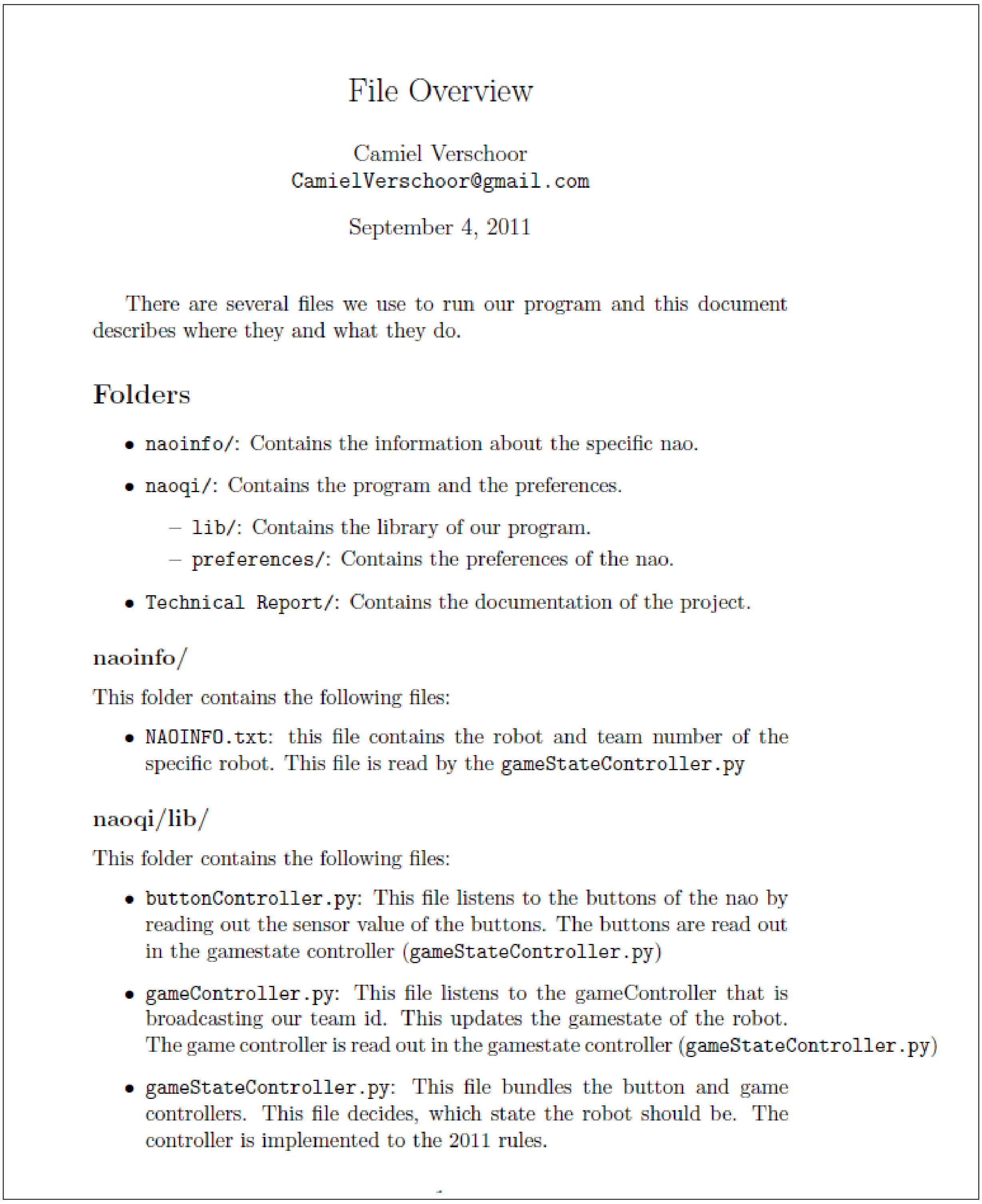
Het voetballen gebeurt volledig autonoom, dat wil zeggen dat de robots niet op afstand aangestuurd worden en alles zelf moeten doen. Het streven is dat er in 2050 een robot voetbal team het tegen de menselijke wereldkampioen kan opnemen.

Ons team bestaat uit een groep studenten Kunstmatige Intelligentie onder begeleiding van dr. A. Visser. Wij zijn het enige Nederlandse team in deze competitie.

Bij de wereldkampioenschappen die volgende week in Istanbul worden gehouden zitten wij in de poule met de Universiteit van Texas en de Humboldt Universiteit. De wedstrijden tegen deze twee teams zijn ingeroosterd op donderdag 7 juli om 14:20 en vrijdag 8 juli om 9:20. Onze voortgang zal te volgen zijn op onze website.

VOLG ONZE VOORTGANG OP
WWW.DUTCHNAOTEAM.NL

12.5 File Overview



File Overview

Camiel Verschoor
CamielVerschoor@gmail.com

September 4, 2011

There are several files we use to run our program and this document describes where they and what they do.

Folders

- `naoinfo/`: Contains the information about the specific nao.
- `naoqi/`: Contains the program and the preferences.
  - `lib/`: Contains the library of our program.
  - `preferences/`: Contains the preferences of the nao.
- `Technical Report/`: Contains the documentation of the project.

naoinfo/

This folder contains the following files:

- `NAOINFO.txt`: this file contains the robot and team number of the specific robot. This file is read by the `gameStateController.py`

naoqi/lib/

This folder contains the following files:

- `buttonController.py`: This file listens to the buttons of the nao by reading out the sensor value of the buttons. The buttons are read out in the gamestate controller (`gameStateController.py`)
- `gameController.py`: This file listens to the gameController that is broadcasting our team id. This updates the gamestate of the robot. The game controller is read out in the gamestate controller (`gameStateController.py`)
- `gameStateController.py`: This file bundles the button and game controllers. This file decides, which state the robot should be. The controller is implemented to the 2011 rules.

- `loader.py`: This file loads the football program if a foot bumper pressed. The file is automatically loads on start up. The file launches the football soul (`soul.py`).
- `locateGoal.py`: The old function to locate the goal. This function is called by the vision interface (`visionInterface.py`).
- `locateGoalzor.py`: The new function to locate the goal. This function is called by the vision interface (`visionInterface.py`).
- `longDistanceTracker.py`: The function to locate the ball. This function is called by the vision interface (`visionInterface.py`).
- `motionInterface.py`: This file is the interface of the motions (`motions.py`). This was created to have a overview of all the motion functions our nao can perform.
- `motions.py`: This file contains all the motions the nao can perform and reads out poses of the nao. The functions are called by the `motionInterface.py`.
- `soul.py`: This is our main program and is started up by the loader (`loader.py`). It bundles the program together and makes the calls to all the inferfaces.
- `testGoalzor.py`: This file was created to test the new function to locate the goal (`locateGoalzor.py`).
- `vision.py`: Contains the function to subscribe and unsubscribe to the cam of the nao. It also contains the function to take a image of the camera. The functions are called by the `visionInterface.py`
- `visionInterface.py`: This file is the interface, which bundles all the vision functions together. This interface calls the function in the specific files. For example when using the findGoal() function the interface calls `locateGoalzor` and returns the output to the main program (`soul.py`. It is the connection between the vision functions and the main program.

naoqi/preferences/

- `autoload.ini`: This file contains all the modules the nao has to load on start up. The file `loader.py` is linked in here as well. On start up the file loader.py is loaded, which can start the football program.