Introduction to DNT

Caitlin Lagrand
caitlin@dutchnaoteam.nl

First version:
September 1, 2018

Last update:
September 25, 2018

1 Introduction

The Dutch Nao Team is a student team of the University of Amsterdam consisting of Artificial Intelligence students, both Bachelor and Master, supported by a senior staff member, Arnoud Visser. It was founded in 2010 and competes in the Standard Platform League (SPL); a robot football league, in which all teams compete with identical Nao[1] robots to play football fully autonomously. The league was started to incentivise 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.

This document will give an introduction to the Dutch Nao Team, including working with Nao robots and creating your own football robot. Firstly, some background information about the team will be given in Sect. 2, followed by information about the Nao robots in Sect. 3. In Sect. 4 a detailed description of how to make a simple football playing robot will be given. Lastly, Sect. 5 will describe the current *DNT Framework*.

2 Team Information

The Dutch Nao Team debuted in the Standard Platform League (SPL) competition at the German Open 2010 [18]. Since their founding, the Dutch Nao Team has been qualified for the world cup competitions in Istanbul [17], Mexico City [16], Eindhoven [1], João Pessoa [3], Leipzig [2], Nagoya [5] and Montreal [6].

Besides the major RoboCup events, we have attended multiple GermanOpens, IranOpens, the Humanoid Soccer School 2013, the Mediterranean Open 2011, the Colombia Robotics week, TechFest 2015[2], the European Open 2016, the Robotic Hamburg Open Workshop 2016 and 2017. At the Benelux Conference on Artificial Intelligence 2016 the team received the award for best demonstration [4]. At the Iran Open 2017 the team received the Award in the Open Challenge with a presentation on our behaviour engine.

During the participation in the RoboCup, the Dutch Nao Team has provided its support or resources in bachelor & master theses[10, 11] and projects that led to publications on a large variety of topics. At the Maastricht University, a PhD student published a paper on learning a more stable gait[13], compared to the energy efficient gait from earlier work[12]. In an honours project a comparison was made on ball detection with classical image processing versus modern deep learning techniques[9]. The Dutch Nao Team extended the application of the Nao robot to the @Home league of the RoboCup: the Nao robot was used to help

[1] https://www.softbankrobotics.com/emea/en/robots/nao

[2] TechFest is Asia's largest science and technology fair with more than 165,000 people attending: http://techfest.org/

Figure 1: One of our Nao robots.

in a kitchen environment by finding a tomato and grabbing it from a table [8, 7]. Finally, the Dutch Nao Team has made the penalty shootout situation into a standalone demonstration [4] which it premiered at the Benelux Conference on Artificial Intelligence 2016[3] and won the first prize for best demonstration.

Earlier the Dutch Nao Team has published papers in the International Conference on Advanced Robotics [20], the Performance Metrics for Intelligent Systems Workshop [14], the RoboCup IranOpen Symposium [19], the RoboCup Symposium [15] and the international conferences as International Conference on Autonomous Robot Systems and Competitions [8].

3 Nao Robots

The Nao robot (Figure 1) is a programmable humanoid robot made by Softbank Robotics[4], 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 1 GB 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 localisation. The Nao has two high-definition (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, which is something that happens regularly during matches. The Nao also possesses four sonars, an infrared emitter and receiver, pressure sensors

[3] http://bnaic2016.cs.vu.nl/

[4] https://www.softbankrobotics.com/emea/en/robots/nao

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 its 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. It is also permitted for the goalie to hold the ball in its hands, but it is highly uncommon for teams to 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 fluent, since the joints of these robots have been worn out. In order to ensure a robust walk for every robot, the joints for each individual robot need to be calibrated.

3.1 V6 Nao

The V6 robot is the newest version of the Nao robot. This robot differs in a few aspects from the older versions. Most importantly, it has a 1.91 GHz Atom quad core processor, with 4 GM of RAM and 32 GB storage. Furthermore, it uses a newer version of Naoqi, in which some main components are removed. So far, we cannot use them for the competition, since our framework relies on those components. However, we can work with them for this introduction. You can recognise the V6 on their dark grey patches.

3.2 Working with Nao Robots

Working with robots takes special care. Always work with at least two people when working with the robot, since it is not that stable and can fall over a lot. Make sure that one person is always standing next to the robot to catch it when it falls and the other one behind the laptop to start and stop the code. Furthermore, when not using the robot, let it go to its rest position by pressing twice on its chest button. To undo this resting mode, again press twice on its chest button.

3.3 Installing NaoQi

Go to https://community.ald.softbankrobotics.com/en/resources/software/language/en-gb and ask one of the current team members for the login credentials. If you are using a V5 robot, download for your OS:
`[FORMER NAO VERSIONS] - Python NAOqi SDK` for Python.
`[FORMER NAO VERSIONS] - C++ NAOqi SDK` for C++.
When using a V6 download for your OS:
`[NAO V6 ONLY] - SDKs and documentation 2.8.3`

For Python check http://doc.aldebaran.com/2-1/dev/python/install_guide.html for how to install. If you get linking errors, please ask one of our team

members for a working SDK. Please note that it only works for Python 2.
For C++ check http://doc.aldebaran.com/2-1/dev/cpp/install_guide.html For C++, we want to use our own OpenCV and not the one from Naoqi, so after installing OpenCV (Sect. 4.1), do the following[5]:

- Once you have OpenCV installed for your system, you have to remove it from the SDK (`/path/to/SDKfolder/naoqi-sdk`). Do not remove the CMake configuration files.
- Remove all OpenCV libraries from the lib/ directory of the SDK. They will have the following format: libopencv_modulename.so, libcv.so, libhighui.so and libml.so
- Remove the opencv folder from the include folder

4 A Simple Football Playing Robot

We will implement a simple football playing robot that can detect the ball, track it, walk towards it and kick it. To make it easier, we will use an orange ball, instead of the black and white checkered ball. We will start implementing in Python to get familiar with the robots, followed by the same in C++.

4.1 Installing OpenCV

Since we are working with images, a computer vision library is very useful. OpenCV is one of the most used libraries and works for both Python and C++. Installing it, however, might cause some problems, therefore we always build it from source (at least on Ubuntu).
For Ubuntu follow the part after *Building OpenCV from source*.
For Windows and for Mac we have never tested it ourselves, since we all use Ubuntu, so no guarantees that the links for Windows and Mac work.

4.2 Python

As stated before, Naoqi only works with Python 2.

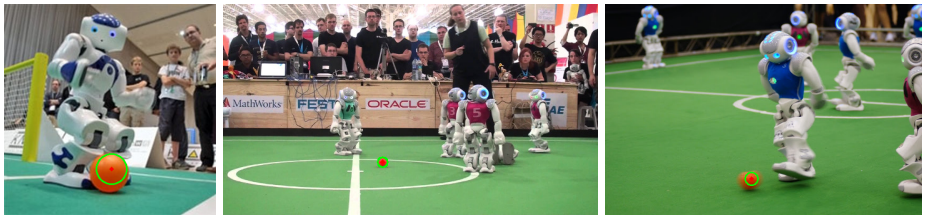
4.2.1 Detecting an Orange Ball

We are going start with detecting an orange ball in images. The file `orange_ball.py` contains the functions that can be used to detect the ball. Firstly, we are looking into detecting the ball in images obtained from a folder. To do this, complete the function that reads the images from a folder (`read_imgs`). Next, we loop over each image and try to detect a ball in the image. The function `detect_orange_ball` is the function that will handle the detection of the ball:

[5] http://doc.aldebaran.com/2-4/dev/cpp/examples/vision/opencv.html

(a) RGB (b) HSV (c) Mask

Figure 2: Example image in RGB (a), HSV (b) and masked with orange (c).



(a) (b) (c)

Figure 3: Example results of the red ball detector.

given an image (RGB), it returns the x and y coordinates of the centre of the ball, together with the radius (r).

Since we want to detect an orange ball, and the ball is about the only orange thing on the field, the easiest way to proceed is to look for an orange circle in the field. Because we are working with colours, it is easier to work in HSV colourspace instead of RGB[6]. Firstly, we will apply a mask (`mask_img`) such that everything in the image that is orange will become white and the rest black. Figure 2 shows an example of the different colourspaces and the mask.

Next, we will try to detect circles in this mask image (`detect_circles`). This can be done using build in functions in OpenCV such as Hough Circles[7] or a blob detector[8]. Make sure to tune the parameters a bit to be able to detect balls in different circumstances. Furthermore, some additional checks such as the percentage of orange within the circle can be added to avoid false positives. Figure 3 shows some results of the orange ball detector.

[6] https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/

[7] https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html

[8] https://www.learnopencv.com/blob-detection-using-opencv-python-c/

Getting images from the webcam. Instead of detecting the ball on previously obtained images, we can also obtain images from our webcam. Try to rewrite your code to obtain images from your laptop and detect balls in them.

Getting images from the robot. The next step is to use the camera of the robot. Checkout the documentation of the *Video Device* or some sample code and try to implement this for the orange ball detector.

4.2.2 Tracking an Orange Ball

After detecting the orange ball, we want to do something with this information. We will first look into tracking the ball with the robots head. To be able to control the joints of the robots, we need to create a `motion proxy`[9]. Complete the function `start_proxies` but ignore the other proxies for now. The robot has 25 joints that can be set separately.[10] Let's start with horizontal tracking: moving the head left or right. The joint needed for this movement is the `HeadYaw`. Simply move the head towards the left if the centre of the ball is in the left part of the image and move it right if the ball is on the right part of the image using the function `changeAngles`[11]. Vertical tracking, moving the head up or down, is basically the same and can be done using the `HeadPitch` to move the head up if the ball is in the upper part of the image, and down if it is in the lower part of the image.

4.2.3 Walking towards the Ball

The final step to let a robot play football is to walk towards the ball. For walking, we need the same `motion proxy` that we used for the head motion, but instead of setting all joints separately, we can use the functions for locomotion[12]. Specifically, we can use the `moveToward` function which let the robot walk with the given (normalised) velocity. We can let the robot turn towards the ball by checking the x coordinate of the ball and setting the theta velocity accordingly. Similarly, we can walk towards the ball by setting the velocity in the x direction of the robot.

4.2.4 Kicking the Ball

Now that it is possible to dribble with the ball, the next step is to kick the ball. You can use the provided function in `kick.py` to perform a kick. Make sure to take extra care of the robot and catch him if he falls over, since the kick is

[9] http://doc.aldebaran.com/2-1/naoqi/motion/almotion.html
[10] http://doc.aldebaran.com/2-1/family/robots/joints_robot.html
[11] http://doc.aldebaran.com/2-1/naoqi/motion/control-joint-api.html#control-joint-api
[12] http://doc.aldebaran.com/2-1/naoqi/motion/control-walk-api.html#control-walk-api

not perfect. Let the robot kick the ball if it is close enough. Before and after kicking, let the robot go to its `StandInit` position using the posture proxy[13].

4.2.5 Search for the Ball

One thing that you might have noticed so far, is that the robot does not do anything specific if it does not see the ball. To search for the ball, we will use the head of the robot. Let the robot move its head randomly, or in a specific pattern, to search for the ball and see how far you get with playing football.

4.3 C++

Now that we have a simple football playing robot in Python, we can do the same in C++. Firstly, make sure that you've installed Naoqi for C++ correctly with your own OpenCV (Sect. 3.3).

If we look at the header file (`orange_ball.h`), a struct `Ball` is defined that contains the x, y and radius of the ball and an enum class for the camera type. Let's start again by detecting an orange ball in images. Complete all functions needed to do this the same way as in Python. To test the code, we need to compile it. Since we are going to use Naoqi, we need compile with `qibuild`. Let's first checkout the `CMakeLists.txt`:

```
cmake_minimum_required(VERSION 3.5.1)
set(CMAKE_CXX_STANDARD 14)

# Naoqi needs g++ 4.8
set(CMAKE_CXX_COMPILER g++-4.8)

# Project name
project(Balldetector)

# This include enable you to use qibuild framework
find_package(OpenCV REQUIRED)
find_package(qibuild)

# Create executable
qi_create_bin(BallDetector main.cpp orange_ball.cpp ball_tracker.cpp)

# Tell CMake the naoqi dependencies
qi_use_lib(BallDetector ALCOMMON ALPROXIES)

# Link with opencv
target_link_libraries(BallDetector ${OpenCV_LIBS})
```

[13] http://doc.aldebaran.com/2-1/naoqi/motion/alrobotposture-api.html

```
#include "orange_ball.h"
#include "ball_tracker.h"

enum class Method {
  ball_detector,
  ball_tracker
};

int main() {
  Method method = Method::ball_detector;
  if (method == Method::ball_detector) {
    OrangeBall orange_ball;
    orange_ball.detect();
  } else if (method == Method::ball_tracker) {
    BallTracker ball_tracker;
    ball_tracker.track();
  }
  return 0;
}
```

Figure 4: `main.cpp`

This file makes sure that we compile with `g++ 4.8`, OpenCV, `qibuild` and the Naoqi dependencies. The files that will be compiled are `main.cpp`, `orange_ball.cpp` and `ball_tracker.cpp` resulting in the executable `BallDetector`. Our `main.cpp` (Figure 4) is pretty simple: depending of the method, it will start the ball detector or the ball tracker.

To compile our code, we need to do the following in the same folder as `CMakeLists.txt` (thus the `cpp` folder):

- A file `qiproject.xml` should exist
- `qibuild init`
- `qibuild add-config mytoolchain -t mytoolchain --default`, where `mytoolchain` is the name of the toolchain you created at installing the sdk for cpp
- `qibuild configure`
- `qibuild make`

This will create a folder `build-mytoolchain/` in which we can find the executable `BallDetector` in the subfolder `sdk/bin/`. Test your code by running the executable `./BallDetector`.

Now, obtain images from your webcam and the robot and test all cases.

After finishing the ball detector in C++, we can continue with the ball tracker. Checkout the ball tracker files (`ball_tracker.cpp` and `ball_tracker.h` and set the method in `main.cpp` to `ball_tracker`. Complete the functions in the ball tracker, as you did in Python.

5 DNT Framework

Will be here some day!

References

[1] Patrick de Kok, Nicolò Girardi, Amogh Gudi, Chiel Kooijman, Georgios Methenitis, Sébastien Negrijn, Nikolaas Steenbergen, Duncan ten Velthuis, Camiel Verschoor, Auke Wiggers, and Arnoud Visser. Team Description for RoboCup 2013 in Eindhoven, the Netherlands. Proceedings of the 17th RoboCup International Symposium, May 2013.

[2] Patrick de Kok, Sébastien Negrijn, Mustafa Karaalioğlu, Caitlin Lagrand, Michiel van der Meer, Jonathan Gerbscheid, Thomas Groot, and Arnoud Visser. Dutch Nao Team Team Qualification Document for RoboCup 2016 - Leipzig, Germany, November 2014.

[3] Patrick de Kok, Duncan ten Velthuis, Niels Backer, Jasper van Eck, Fabian Voorter, Arnoud Visser, Jijju Thomas, Gabriel Delgado Lopes, Gabriëlle Ras, and Nico Roos. Dutch Nao Team team description for RoboCup 2014 - João Pessoa, Brasil, 2014.

[4] Caitlin Lagrand, Patrick de Kok, Sebastien Negrijn, Michiel van der Meer, and Arnoud Visser. Autonomous robot soccer matches. In *BNAIC2016 Proceedings*, pages 237–238, 2016.

[5] Caitlin Lagrand, Sébastien Negrijn, Patrick de Kok, Michiel van der Meer, Douwe van der Wal, Pieter Kronemeijer, and Arnoud Visser. Team qualification document for robocup 2017, nagoya, japan. Technical report, University of Amsterdam, Science Park 904, Amsterdam, The Netherlands, 2016.

[6] Caitlin Lagrand, Sébastien Negrijn, Michiel van der Meer, Douwe van der Wal, Linda Petrini, Hidde Lekanne Deprez, Pieter Kronemeijer, Santhosh Kumar Rajamanickam, Jier Nzuanzu, Lukas Jelinek, and Arnoud Visser. Team qualification document for robocup 2018, montreal, canada. Technical report, University of Amsterdam, Science Park 904, Amsterdam, The Netherlands, 2018.

[7] Caitlin Lagrand and Michiel van der Meer. The roasted tomato challenge. *Amsterdam Science*, 05:4, April 2017.

[8] Caitlin Lagrand, Michiel van der Meer, and Arnoud Visser. The roasted tomato challenge for a humanoid robot. In *Proceedings of the IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Bragança, Portugal, 2016.

[9] Caitlin Lagrand, Douwe van der Wal, and Pieter Kronemeijer. Detecting a checkered black and white football. honour's project report, Universiteit van Amsterdam, February 2017.

[10] Caitlin G. Lagrand. Learning a robot to score a penalty – minimal reward reinforcement learning. Bachelor's thesis, Universiteit van Amsterdam, June 2017.

[11] Sébastien Negrijn. Exploiting symmetries to relocalise in robocup soccer. Master's thesis, Universiteit van Amsterdam, December 2017.

[12] Z. Sun and N. Roos. An energy efficient dynamic gait for a nao robot. In *2014 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 267–272, May 2014.

[13] Zhenglong Sun and Nico Roos. A controller for improving lateral stability in a dynamically stable gait. In Lazaros Iliadis and Ilias Maglogiannis, editors, *Artificial Intelligence Applications and Innovations*, pages 371–383, Cham, 2016. Springer International Publishing.

[14] Sander van Noort and Arnoud Visser. Validation of the dynamics of an humanoid robot in usarsim. In *Proceedings of the Performance Metrics for Intelligent Systems Workshop (PerMIS'12)*, 2012.

[15] Sander van Noort and Arnoud Visser. Extending virtual robots towards robocup soccer simulation and @home. In *RoboCup 2012: Robot Soccer World Cup XVI*, pages 332–343. Springer, 2013.

[16] C. Verschoor, D. ten Velthuis, A. Wiggers, M. Cabot, A. Keune, S. Nugteren, H. van Egmond, H. van der Molen, R. Rozeboom, I. Becht, M. de Jonge, R. Pronk, C. Kooijman, and A. Visser. Dutch Nao Team – Team Description for RoboCup 2012 - Mexico City. In *Proceedings CD of the 16th RoboCup Symposium*, June 2012.

[17] C. Verschoor, D. ten Velthuis, A. Wiggers, M. Cabot, A. Keune, S. Nugteren, H. van Egmond, H. van der Molen, R. Rozeboom, I. Becht, M. de Jonge, R. Pronk, C. Kooijman, and Arnoud Visser. Dutch Nao Team – Team Description for RoboCup 2011 - Istanbul. In *Proceedings CD of the 15th RoboCup Symposium*, January 2011.

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

[19] Arnoud Visser, David de Bos, and Hessel van der Molen. An experimental comparison of mapping methods, the gutmann dataset. In *Proceedings of the RoboCup IranOpen 2011 Symposium (RIOS11)*, 2011.

[20] Auke Wiggers and Arnoud Visser. Discovering reoccurring motifs to predict opponent behavior. In *Proceedings of the 16th International Conference on Advanced Robotics*, Montevideo, Uruguay, 2013.