UNIVERSITY OF AMSTERDAM
FACULTY OF SCIENCE
THE NETHERLANDS

DUTCH NAO TEAM

Technical Report

Authors:
Thomas WIGGERS
Douwe VAN DER WAL
Hidde LEKANNE GEZEGD DEPREZ
Wouter ZWERINK
Pieter KRONEMEIJER

Supervisor:
Arnoud VISSER

January 15, 2020

Abstract

In this Technical Report, the Dutch Nao Team lists its progress and activities in the past year with the previous report [1] as a starting point. Besides new developments this report also lists older developments when relevant. This year, progress has been made in developing vision, localisation modules and logging. For vision, line detection has been developed which was consequently used in the localisation module. To be able to have access to more data and faithfully compare different algorithms, a logging module has been created that can save sensory data during a game and can load it back into the framework later in order to evaluate the performance of new algorithms. Furthermore, the team has made the framework fully compatible with the V6 version of the NAO robot. The team has also participated in several events, namely the Robotic Hamburg Open Workshop (RoHOW) and the 2019 RoboCup in Sydney. Finally, several demonstrations and lectures on robotics and AI have been given at both schools and companies.

Contents

1 Introduction

The Dutch Nao Team consists of one alumni and twelve Artificial Intelligence students: eight Master students and four Bachelor students, who are supported by a senior staff member, Arnoud Visser. The team was founded in 2010 and competes in the RoboCup Standard Platform League (SPL), which is a robot football league in which all teams compete with identical robots to play football. The league was started to incentivise the development in robot science. Its current goal is to play against the human world champion of 2050, and win. Since all teams participating in the Standard Platform League are obliged to use identical robots, the focus of the league is solely software oriented rather than hardware oriented. The robots need to be able to play autonomously. This includes finding the ball, locating itself on the field, and making decisions on how to play next, as well as communicating with teammates and being able to walk.

Figure 1: Team photo in Sydney

2 Hardware

2.1 The Nao robot

The Nao robot is a programmable humanoid robot made by Softbank Robotics, formerly known as Aldebaran Robotics. Up until 2018, all versions 4.x and above of the Nao were equipped with the same computational hardware, only differing in their sensors or actuators. The release of the sixth version (V6) introduced a significant change in both hardware and proprietary software. This year we have made an effort to optimise our framework for V6, and have stopped supporting the V5 robots. They lack the computational power required and differ in the camera quality and software interfaces. These differences make maintaining the framework for both versions simultaneously exceedingly difficult hence our decision to to only use the V6 this year. This has allowed us to implement techniques that were previously too computationally heavy. Below, the V6 is detailed.

While the V6 hardware is significantly improved compared to previous versions, CPU resources are still scarce. This limits calculation-heavy tasks such as expensive pixel-wise image operations and deep neural network approaches. 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 whether the robot has fallen, which happens regularly during matches. The Nao also possesses four sonars, an infrared emitter and receiver, pressure sensors and tactile sensors. Except for the pressure sensors, these sensors are used less frequently than the cameras and inertial board, as they are more prone to breaking down, resulting in faulty measurements. The pressure sensors however become one of the most important sensors, since the new walking engine relies heavily on the pressure information coming from the feet for stable walking.

From the V4 and up the Nao robot has 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 while 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 is playing football. 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. Additionally, each robot's camera can shift inside its enclosure, resulting in a slight offset in the transformation with respect to the robot centre. To correct for this, the cameras can be calibrated.

The V6 is a 64-bit system and has a 1.91Ghz atom E3845 CPU which is a Quad core with one thread per core. Ram has increased to 4 Gb and storage has increased to 32Gb SSD over previous versions. Furthermore the WiFi connection has improved and the fingertips are more resistant to impact. With the added computing power, new approaches to issues like localisation and ball detection will be possible. This year we have focused on building modules that can take advantage of the robot improved computing capabilities.

This year the V6 robot has also been used in matches. The DCM replacement, called LoLA (Low Level Access) is now supported by our framework and V6 robots are fully functional.

3 Framework

3.1 V6 Compatibility

Our framework was created in 2017. Its structure is based on [2] and is further elaborated in [3]. Moreover it uses Kahns algorithm [4] to link different modules in a proper sequential order within each module group.

3.2 Module Communication

This year the need arose for the ability to execute some modules at a different update frequency than others. Specifically the modules responsible for motion would benefit from a high constant frequency while other modules (vision for instance), could significantly slow the update frequency with computational expensive tasks. Additionally the computation needed for a task could be variable, depending on the situation the robot finds itself in. Besides this issue there was also the opportunity to reduce executing times by utilising the additional cores.

In order to make this possible, modules with strict dependencies were put together in module groups. Each module group is on a separate thread. This new system needed a method to exchange information between modules on separate threads. As different threads can have different update times, memory access needs to be safe and preferably managed without additional specifications inside each module.

Thus a new message system was proposed. In its new form any module can send an output messages which contains a pointer to a representation object. Every module in need of that message gets a copy of that message in their message queue. Entries in these queues can only be read and/or deleted, not modified. This queue has a maximum size after which the oldest message gets deleted. The module accessing it's queue can do so by popping messages. Within module groups, queues never get larger than size one. When modules need to communicate across threads, the amount of messages can increase depending on the difference in execution time.

3.3 Interface

Since our last report, the interface has been replaced by a qt-framework[1] because of its more extensive functionality compared to NanoGUI[2]. No changes in widget methodology have occurred.

All information comes directly from a robot and can be communicated over either WiFi or Ethernet. In order to make this possible, every representation needs to be serialisable. For this purpose the C++ Cereal library's[3] binary serialise function is used. This function limits header data to a minimum not even needing to communicate type information, reducing bandwidth usage to a minimum. However this comes at the cost of defining both save an load functions for every representation needed to be serialisable.

Additionally, compared to last year, communication is centralised. There is one communication module within the interface which receives all representations. These representations are then passed along to the relevant widgets following our thread safe module communication system.

[1]www.qt.io
[2]https://github.com/wjakob/nanogui
[3]https://uscilab.github.io/cereal/

With this new framework several visualisation methods have been developed for the: line detection, line model, particle filter and the behaviour engine. The line detection overlays the detected lines onto the live feed of top an bottom camera. The line model project the lines it the robot should on the camera feeds. The particle filter shows all the potential particles (in black) and the best particle (in red) onto a modelled field. The behaviour widget shows at each frame which behaviour has been chosen en which considerations it has within that behaviour.

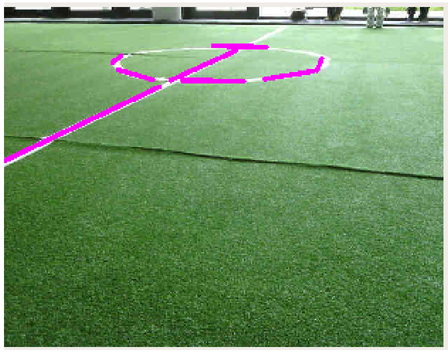


Figure 2: Detected Lines

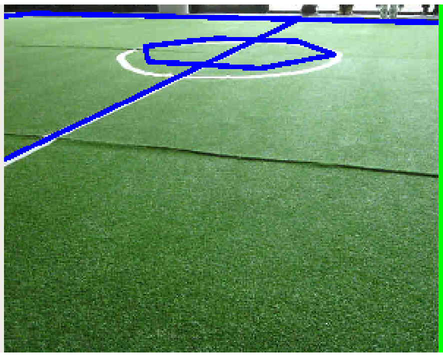


Figure 3: Line Model

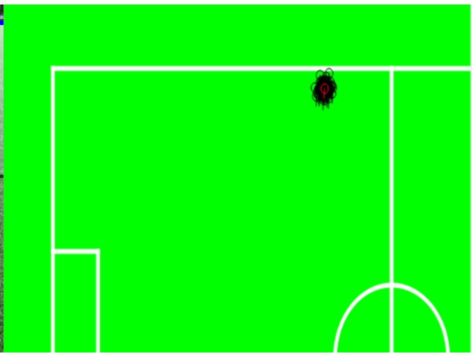
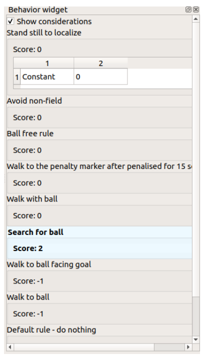


Figure 4: Particles


Behavior widget
✓ Show considerations
Stand still to localize
Score: 0

	1	2
1	Constant	0

Avoid non-field
Score: 0
Ball free rule
Score: 0
Walk to the penalty marker after penalised for 15 s
Score: 0
Walk with ball
Score: 0
Search for ball
Score: 2
Walk to ball facing goal
Score: -1
Walk to ball
Score: -1
Default rule - do nothing

Figure 5: Behaviour engine

3.4 Recording Robot Sensor Data

This year a module has been created to store sensor data, and another to load the data back into the framework. The data storage is handled by the C++ Cereal library. This solves several problems that limited both development of new modules and testing of new modules.
First of all, the data saved comes directly from the framework executed during a match. This ensures the data recorded is identical to the data received during a match making detailed evaluation possible. Additionally, data can be recorded every n frames, to save disk space and computational power. This may not be handy in match evaluation, but greatly enhanced our ability to generate data for machine learning. Its second function is to test algorithmic or parameter changes as a robot can be simulated using the recorded sensor data. This greatly shortens the development process for non-behavioural changes as no physical robot is required.

3.5 Remote Compiling

As the framework grows, both due to complexity and number of modules, the compiling time increases too. To decrease the wait for compiling and increase productivity, a remote compiling script called Mainframer[4] has been made compatible with the framework. The script runs on one of the RobolabWS workstation[5] located in the Intelligent Robotics Lab. This workstation contains a 10 core Xeon processor (e5-2640 v4), which outperforms most laptops. The overhead of syncing the files is minimal, as only changed files are synced.

4 Motion

4.1 Walking Engine

Last year a stripped down version of the walking engine from B-Human [5] was converted to the DNT framework, with the intention to expand it with features deemed necessary after seeing it in action.
The only feature that was added to the walking engine in the previous year, is an in-walk kick. Originally, the in-walk kick was implemented in the previous year when the motion engine was originally ported to the DNT framework. However, after some use it appeared to be broken, as it sometimes made the robot stick out its leg forward in a rather uncontrolled manner, resulting in a fall and heavy wear on the hip gears. To remedy this, the whole kick algorithm was rewritten in a more structured way. Now, the robot is able to take a big step with one of its legs without falling over, resulting in a quick kick motion.

4.2 Interpolation engine

While last years report rightly stated that "all motions ran smoothly", there were still some problems when switching between motions. Sometimes, the robot was unable to finish one motion correctly before another one was started, resulting in sudden spasms. For example, after the robot got up from a fallen position, it would very quickly move its legs a centimetre or two to get in the initial position for the next motion, e.g., walking. While this rarely destabilised the robot enough for it to fall over, such motions are not good for the gears.
While attempting to solve this problem, many small things were changed, including a revision of the default stand motion, a change in stiffness management, an update to the way different robot stances are detected, and a restructure of the interpolation engine. While all updates improved code quality and introduced safeguards against spastic behaviour, the revision of the default stand motion finally fixed the spastic behaviour completely, as this removed the need to interpolate between any motions, since they all end and start from the same position.

4.3 Fall damage protection

Often when robots fall on the field, a cracking sound is heard coming from hip, leg or arm joints. This is likely caused by motors not being strong enough to take the full force of the fall of a robot. For example, when a robot falls forward, the motors in the shoulder joints are not suited for catching the robot and breaking its fall. Since there is no need at all for any joints to have any stiffness when the robot is certain to fall, a 'reflex' system was put in place that releases stiffness at every joint when the robot upper body passes a certain angle. Joints can move more freely this way, preventing motors taking damage from big impacts, and hopefully saving some trips to France for repairs.

[4]https://github.com/buildfoundation/mainframer
[5]https://github.com/IntelligentRoboticsLab/robolabws

5 Vision

5.1 Localisation

This year we have added localisation. This allows the robot to keep track of its own position on the field. While the need for this information is high, it is also very difficult to obtain. The robots movements can be very erratic and the video feed is blurred by the motion from walking. This causes our robots to quickly lose track of their position.

For localisation we have implemented a particle filter, following the method described in [6]. At the start of the match we know our position, and by combining odometry and visual cues from line detection we can try to infer our position. This is done by first transposing the particles according to the odometry calculations. Then a number of particles are sampled by adding generated random noise from a normal distribution. Finally all particles are evaluated using the lines from the line detection and the lines a particle should see, given its position on the field. While the robot performs this update it stops walking to prevent motion blur from affecting its line detection. The best particle is then selected as the current position.

The robot may re-localise after the penalised state. In this state the robot is placed on its own half on a specific location on the sidelines. Because the actual side the robot is placed on is determined by the referee, we initialise a particle on both sides, and let the robot determine on which side it was placed.

5.2 Green Detection

Previous implementations of field border detection relied on the definition of the field as a volume in the HSV colour space. This approach was sufficiently accurate to not miss any lines on the field and not detect any objects outside the field, provided that the definition of the volume was configured well. This configuration presented a major flaw in the approach, as 6 different parameters[6] needed to be configured at every new field and upon every change in lighting. These parameters are difficult to tune and this method could not handle natural lighting conditions which were present at the RoboCup of 2019.

In order resolve these issues, a new approach to colour detection was proposed. Instead of defining a fixed colour space, the image would be normalised on the colour green. This normalisation is achieved through the formula $Gc = \frac{Green}{Green+Blue+Red}$, where Gc represents a value between $0 - 1$ indicating how strong the green channel is relative to the other channels and therefor provides a measure of green chromaticity.

In this colour space anything black, white or grey is expected to have a value of roughly $\frac{1}{3}$ as all channels are expected to be of equal value. Green pixels should thus have a value significantly more than $\frac{1}{3}$.
Dark pixels represent a challenge for this approach due to noise having a stronger effect on their pixel intensities. This can cause the green chromaticity to change erratically. As the Gc value of dark pixels can not be trusted, dark pixels are filtered out before the Gc value is computed. They are classified as non-green pixels.

[6]Hue_min, Hue_max, Saturation_min, Saturation_max, Value_min, Value_max

The SPL rules do not define the colour of the field beyond "green". To allow the green detector to work robustly on fields of varying shades of green, the threshold on the Gc channel is determined dynamically. Under the assumption that the most prevalent colour in the image is that of the field, the Gc value of this colour should be apparent in a plot of the Gc channel. To discretely compute the value, it is defined to be a normal distribution with as mean the peak of a histogram of the Gc channel values for all values significantly above $\frac{1}{3}$.

5.3 Field Detection

Field detection is used to reduce noise, save computational power and extract localisation features; noise reduction is achieved by masking objects outside of the field; computational power is saved by no analysing things outside the field; and the edge of the field can be used to further enhance localisation.

The green detector is used to feed detect pixels where the field might be present. Several statistical methods are applied in this module to filter out noise[1]. Besides the new colour detection, an additional method was proposed to deal with erroneous field edge detections. This method filters field edge detections by assuming they must form a convex hull. A convex hull is defined as the outer edge of a convex shape containing all the field edge detections. By overwriting field edge detections when they deviate heavily from this convex shape, outliers can be corrected.

5.4 Line Detection

To aid localisation, the team has implemented a line detector to detect the white field lines. It begins by filtering the pixels that are unlikely to be part of a field line, resulting in a binary image. The green pixels from the green detector and insufficiently bright pixels are filtered out.

Lines that need to be detected can have varying widths throughout the image, a few pixels in the distance to over forty pixels wide at the bottom of the image. To get around this, the line detector detects edges. These edges are found using a fast approximation to the Canny filter, which checks the four neighbours of a potential white pixel to decide if it is an edge pixel. This approximation works well as long as the number of gaps within lines is small.

Segments are extracted from the edge map using a simple segment detector. It begins by finding a white pixel and sets this to be the seed. From the seed, it seeks out another white pixel within a small range. If no such pixel is present the seed is removed and a new seed is found. If a second pixel has been found the seed now has a direction. New points are added by finding white pixels near the newest point in the correct direction, which is recomputed upon adding points. If no more points are found, the segment gets accepted if it has a sufficient number of points in which case a segment is computed from the points.

After the segments have been found, the line detector will attempt to merge them. This will cause the two edges of a single line to be combined and will additionally make the algorithm more robust to small gaps breaking up lines.

To determine whether segments should be merged, their angles are first compared. If their angle is similar, the distance between the segments is computed. Segments which are close together and have similar angles are then merged if the resulting segment has enough white in it.Once no more segments can be merged, the line detector outputs the detected segments as pairs of points.

5.5 Expected Ball Size

In order to speed up the ball detector, we have reduced the amount of candidate regions we consider. By taking into account the current posture of the robot, we can compute the position and orientation of the camera's with respect to the floor. Given that the field is a flat surface and the ball has a fixed size, we can predict the size of the ball in some region of the image. This allows us to filter out a large amount of incorrectly sized ball candidates. We further reduce the number of candidates by checking if the candidate contains enough white, and we move the window to the centroid of the white pixels. This has the effect of improving the position of the candidate on the ball, and reduces the amount of overlap needed in the candidates generation. Finally, we check if a blob is present before we run the actual classifier. By significantly reducing the amount of times the classification is performed, we increase the update frequency of the ball detector which improves the robot's ability to keep track of the location of the ball.

The expected ball size is computed using the position of the foot the robot is standing on. From there we apply forward kinematics, as described in [1] in section 4.2, to compute the angle θ and height h of the camera. The camera intrinsics such as vertical view angle and the number of rows in the image are known constants. We interpolate linearly across the image using

$$\theta_c = h + \theta + \text{view\_angle} \cdot \left(\frac{y}{\text{rows}} - 0.5 \right)$$

where y is the vertical index of pixel at which we would like to estimate the ball size. The expected size of the ball in pixels is then computed as

$$x = \arctan\left(\frac{\sin(\theta_c) \cdot \text{obj\_size}}{2h - \text{obj\_size}} \right) \cdot \frac{2}{\text{view\_angle} \cdot \text{num\_rows}}$$

5.6 Ball Model

The use of the new V6 robots has caused some problems with ball detection. The cameras have an increased dynamic range, which significantly changes the representation of colour's in the image. Unfortunately the Cascade classifier we use is trained using images captured by the older V5. This, combined with the darker colour of some parts of the plastic shell, has caused a large amount of false positives to be detected by the ball detector. The erratic detection of these false positives in other robots leads the robot to move towards those in stead of the actual ball. Therefore we have begun work on a ball model. This model should interpret the detection's from the ball detector, and judge whether they are likely to be the actual ball or not. Furthermore, when the robot does not see the ball for some time, it could still have some idea of where it might be.

To implement this, we considered two solutions. The first is the simplest approach, using a simple threshold. If the number of ball detection's in the same region is not larger than the threshold, the position of the ball model is not updated to these new detection's. This attempts to filter out erratic false positives, as the actual ball is often detected multiple times. The second approach is to use Kalman filtering [7]. This is a much more sophisticated approach which is very effective in reducing the noise using multiple measurements.

However, when testing these techniques, we encountered difficulties with validating the implementation and tuning the parameters. Because we have no ground truth data available, we cannot properly evaluate the effectiveness of these solutions. Therefore we were not able to successfully use these techniques in the RoboCup.

6 Sound

6.1 Whistle Detection

A whistle sound signals the transition from the get ready state to the playing state. Not hearing this sound results in a 15 second delay, not starting to play until an additional message is send of the network. Detecting this signal is crucial in the early stages of every match played.

In order to do this, sound is recorded with the two microphones on the head of the robot. After recording for about half a second, the sound is analysed on high pitched sounds indicating the sound of a whistle. To detect the pitch a fast Fourier transform[8] (FFT) is performed on the recorded sound. A high enough pitch observed frequently enough is considered a whistle sound. The height and frequency are set manually according to the performance on a set of prerecorded whistle sounds

7 Results

7.1 Robotic Hamburg Open Workshop (RoHOW)

RoHOW[7] is an annual open workshop for SPL teams, organised by the German team `HULKs`[8]. It took place from the 30th of November until the 2nd of December 2018. During the event, test games are played, algorithms and ideas about challenges are shared, and lots of coding is done. Furthermore, the more experienced members of most teams gave presentations or workshops about a broad range of subjects, the V6 however, took the spotlight for most of the workshops. Several issues like writing thread-safe code, using the integrated GPU and general V6 bugs were discussed. Overall, the atmosphere is relaxed without the pressure of competitive games, which makes it a great opportunity for new team members to get to know the flow of working with robots and being in the SPL.

7.2 RoboCup Sydney

The Dutch Nao Team qualified for the RoboCup 2019 (2-8 June 2018) in Sydney with a video[9] and a qualification paper [9]. This was the first event with the new walking engine and the field detector.
During the first round, the Dutch Nao Team played $2 - 0$ against `Starkit`. Both goals were after the start signal while we were still correctly localised. After that the team played $0 - 0$ against `RoboEireann` where we couldn't score because the opponent intercepted us after the start signal, after which our robots couldn't properly localise. Finally we lost $0 : 2$ against `Nomadz`.
After this, the team had enough points (4) to try qualify for the major league. A draw was played with score $0 - 0$ against `Camellia Dragons`, leading to penalties, which our team won with $1 - 0$. This win qualified the team for the major league. Both games in this league were lost; $0 : 9$ against

[7]https://rohow.de/2018/en/
[8]https://www.hulks.de/
[9]`https://www.youtube.com/watch?v=dOELKOFmX90&feature=emb_logo`

the `Nao Devils` and 0 : 6 against `TJArk`. Besides the opponents having more extensive code bases, both matches were played with natural lighting, which made the ball-detector fail.

7.3 Foundation

In 2016, the Dutch Nao Team started a foundation, in order to be able to receive money from companies in a transparent way.
In 2018-2019, the board of the foundation consisted of the following members:

1. Chair: Caitlin Lagrand
2. Vice-chair: Sébastien Negrijn
3. Secretary: Pieter Kronemeijer
4. Treasurer: Douwe van der Wal

7.4 Public events

The past year, the team put a significant amount of effort into providing demonstrations, lectures and workshops. As the year before, all the funds raised by these events were used to fund the trip to the RoboCup.

In addition to the events scheduled by the team, the University of Amsterdam also invites us to their public events. In doing so, the team normally uses this opportunity to interact with young kids, upcoming students or alumni as a way of educating the public about robots and AI.

Events of this year were:

1. 6-10-2018: Science park open day, where the team reached out to the broader community by giving demo's at a stand.
2. 07-01-2019: Visitation of participants of the "First Lego League" where the team gave a talk and demo.
3. 18-02-2019: DNT open day for bachelor students, where bachelors of Artificial intelligence were introduced to the Dutch Nao Team.
4. 01-03-2019 until 03-03-2019 Rodeo: where the team attended a workshop weekend featuring the standard platform league, organised by the NAo Devils
5. 04-04-2019: visitation of highschool students interested in robots, they got a demo and a talk about robotics.
6. 05-04-2019 until 06-04-2019: Ernst and Young Hackaton, where the team participated in the 24-hour hackaton to come up with innovative solutions using machine learning.
7. 03-05-2019: teensevent, where the team gave a presentation to teenagers about robotics and artificial intelligence.
8. 13-05-2019: Elementary school Poseidon demo, where the team gave an educational presentation, in cooperation with researcher Thomas Mensin, to children from the sixth grade.

9. 22-05-2019: Demo municipality council of Amsterdam, where the team gave a demo and a talk about artificial intelligence to the municipality of Amsterdam.
10. 03-06-2019: PA consulting workshop, where the team organised a workshop for the winners of the Raspberry Pi challenge 2019.
11. 04-06-2019: Where the team gave a talk about our organisation to an Indian delegation.
12. 07-06-2019: VIA parent evening, where the team gave a presentation about artificial intelligence to parents invited by the study organisation VIA.
13. 15-06-2019: University day, where scientists present new insights to the general public, including our team.
14. 20-06-2019: Workshop with the youngest team of the Raspberry Pi challenge.

To accommodate several workshop events this year, tutorials and exercises for specific age groups were written to bring students in contact with Choreographe[10].

8 Contributions

List of people working on the additions mentioned in this report.

- **Caitlin Lagrand**: worked on: interface, ball detection, whistle detection, collision detection, behaviours
- **Douwe van der Wal**: worked on: saving robot inputs, remote compiling
- **Hidde Lekanne gezegd Deprez**: worked on: field detector, interface optimizations
- **Jier Nzuanzu** : worked on: behavior engine interface
- **Santhosh Kumar Rajamanickam**: worked on: whistle detection
- **Pieter Kronemeijer**: worked on: everything motion related
- **Sébastien Negrijn**: worked on: localisation, updating to V6
- **Thomas Wiggers**: worked on: frame rate limiter, ball model, masking body in lower camera
- **Wouter Zwerink**: worked on: line detection

9 Conclusion

This year, we have achieved a very important milestone. We scored our first goal against another team in the RoboCup. This marks a moment in our teams history for which the team has worked very hard. This year we have managed to complete the fundamental components required to score in an actual match. While there are many areas to be improved, we have built a complete foundation capable of playing football and from which we can improve for next year. With the migration to the V6 platform we will explore new approaches and better technique. Our team looks forward continuing research and we hope to do even better next year.

References

[1] Hidde Lekanne gezegd Deprez et al. *Dutch Nao Team - Technical Report*. Tech. rep. University of Amsterdam, Dec. 31, 2018. URL: `http://www.dutchnaoteam.nl/wp-content/uploads/2019/01/dnt_techreport_2018.pdf`. published (cit. on pp. 2, 8, 9).

[2] Elizabeth Mamantov et al. "Robograms: A lightweight message passing architecture for robocup soccer". In: *Robot Soccer World Cup*. Springer. 2014, pp. 306–317 (cit. on p. 4).

[3] Douwe van der Wal, Pieter Kronemeijer, and Caitlin Lagrand. *Dutch Nao Team - Technical Report*. Tech. rep. University of Amsterdam, Dec. 31, 2017. URL: `http://www.dutchnaoteam.nl/wp-content/uploads/2018/01/dnt_techreport_2017.pdf`. published (cit. on p. 4).

[4] Arthur B Kahn. "Topological sorting of large networks". In: *Communications of the ACM* 5.11 (1962), pp. 558–562 (cit. on p. 4).

[5] Colin Graf and Thomas Röfer. "A center of mass observing 3D-LIPM gait for the RoboCup Standard Platform League humanoid". In: *Robot Soccer World Cup*. Springer. 2011, pp. 102–113 (cit. on p. 6).

[6] S. Thrun et al. *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents series. MIT Press, 2005. ISBN: 9780262201629. URL: `https://books.google.nl/books?id=2Zn6AQAAQBAJ` (cit. on p. 7).

[7] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45 (cit. on p. 9).

[8] James W Cooley and John W Tukey. "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of computation* 19.90 (1965), pp. 297–301 (cit. on p. 10).

[9] Pieter Kronemeijer et al. *Team Qualification Document for RoboCup 2019, Sydney, Australia*. Tech. rep. Science Park 904, Amsterdam, The Netherlands: University of Amsterdam, Jan. 15, 2019. URL: `http://www.dutchnaoteam.nl/wp-content/uploads/2019/01/DNT_TeamQualificationDocument2019.pdf`. published (cit. on p. 10).

[10] Emmanuel Pot et al. "Choregraphe: a graphical tool for humanoid robot programming". In: *Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on*. IEEE. 2009, pp. 46–51 (cit. on p. 12).