University of Amsterdam
Faculty of Science
The Netherlands

Dutch Nao Team

Team Report 2025

Students:
Vivienne Jansen
Amanda Jansen
Mark Honkoop
Julia Blaauboer
Macha Meijer
Marilene Oud
Rick van der Veen
Fiona Nagelhout
Harold Ruiter
Fyor Klein Gunnewiek
Kim Verkuijl
Dário Xavier Catarrinho
Gijs de Jong

Supervisor:
Arnoud Visser

January 6, 2026

Abstract

In this Technical Report, the Dutch Nao Team lists its progress and activities in the past academic year with the previous report [1] from 2024 as a starting point. Besides new developments, this report also lists older developments when relevant.

Contents

1 Introduction

The Dutch Nao Team (DNT) is a student robotics team from the University of Amsterdam (UvA) that competes in the RoboCup. RoboCup is an international robot football competition that began as a scientific initiative, aiming for autonomous humanoid robots to defeat the human world champion in football by 2050 under official FIFA rules [2]. Within the RoboCup, the team competes in the Standard Platform League (SPL). In this league, teams play matches using identical NAO robots, so the focus is entirely on software development [3].

DNT was founded in 2010 [4]. The team emerged after the SPL switched from AIBO robots to NAO robots in 2008. Before that transition, the group had been active since 2003 as the Dutch AIBO Team [5]. This year, the Dutch Nao Team consists of 5 Bachelor's, and 15 Master's Artificial Intelligence (AI) students, and 2 Master's Computer Science (CS) students, who are supported by a senior staff member, Arnoud Visser.

Figure 1: Team photo taken at the Robotics Hamburg Open Workshop (RoHOW) 2025.

The aim of this report is to present the main organizational and technical developments of the team over the past year. Chapter 2 describes the structure of the team. For the technical developments, Chapters 3, 4 and 5 present the projects related to vision, motion, and behavior, respectively. Chapter 6 covers the workshops organized by the team, as well as the events the team has attended. Chapter 7 outlines the plans for 2026. Chapter 8 provides an overview of the contributions of every team member, and Chapter 9 concludes the report. This year, Chapter 10 is dedicated to special acknowledgments for the contributions of all team members throughout the years.

2 Team Structure

For a team to function well, it is required to not only have people working on the technical tasks, but also on organizational tasks. Last year, a new team structure was introduced with the aim of improving the transparency and more clearly dividing the roles and tasks of each member [1].

2.1 Board and Management

The Dutch Nao Team is managed by two internal groups: the **board** and the **management** team. The board is the legal entity behind the Dutch Nao Team. Its primary tasks are handling the financial and administrative duties related to the team. This year, the board consisted of four members:

- The **Treasurer** is responsible for making the yearly budget and other financial related tasks. This year, Stephan Visser served as the treasurer until September, after which Kim Verkuijl took on the role.
- The **Secretary** handles the communication between the team and external parties. This year, Joost Weerheim served as the secretary until September, after which Julia de Vries took on the role.
- The **President** is responsible for the legality of the actions of the board, leads the board meetings and helps wherever needed. This year Lasse van Iterson served as the president until September, after which Mark Honkoop took on the role.
- The **Vice-President** assists the president with their duties, and ensures that the board is working smoothly by helping out wherever is needed. Dário Catarrinho served as vice-president in the previous year, and will continue to do so next year.

Last year, a new organizational structure called management had been introduced. The main goal of management was to decrease the growing workload of the board that came from the rapidly increasing number of new team members. The past year, management consisted of the same four members as last year. Harold Ruiter served as the **team leader**, Marina Orozco González as the **operational lead**, and Macha Meijer and Gijs de Jong as the **AI lead** and **software lead**, respectively.

2.2 Tech teams

Within the team there are two groups that work on different areas of the framework. Those are the software team and the AI team, which are led by the software and AI leads. These teams work independently from each other, and every member of the Dutch Nao Team is part of one of the teams. The software team focuses mostly on the framework, such as the behavior engine, robot communication, and line detection. The AI team focuses on computer vision related tasks, such as ball and field-mark detection, but also on RL behaviors.

The teams work on multiple smaller projects. Depending on the scope of the projects, one or more members of the team are assigned to it. Which members work on these projects is usually based on their interests. Once a week, all members of the team meet up to update each other on the progress, discuss any issues that came up, and ask others for help if needed.

2.3 Committees

We continued this year with the 3 committees we have maintained for over 2 years now; **Workshops and Events**, **Partnerships** and **Outreach/Socials**. Where our Workshop committee has continued its trend of delivering workshops and informing various different audiences about our activities in the lab (as shown in section 6), this year served as a preparation year for the other two committees. With big changes coming up in the upcoming year, we want to make maximum use of the new marketability to create interesting content for our social media channels, and attract companies that might be interested in the newfound exposure. Our plans for 2026 are to scale up our activities in these committees, by encouraging our team members to actively participate in these committees.

3 Vision

To perceive its surroundings, the NAO robot relies primarily on its two high-definition (HD) cameras, which serve as its eyes. Both cameras are located in the head of the robot: one faces forward to detect distant objects, while the other is angled downward to capture nearby objects. This year, we focused on several projects aimed at extracting and utilizing the visual information from these cameras. In this section, we describe our recent work on chromaticity, which has been newly added, and on improvements in line detection along with improvements in vision-based localization.

3.1 Chromaticity

To robustly filter the green field from other objects in different lighting conditions, our vision pipeline uses chromaticity. For each pixel we calculate their green chromaticity value as shown in Equation 1, normalizing the green channel with respect to their total intensity. This helps us look at how green each pixel is, independent of how bright or dark the pixel appears.

$$g = \frac{G}{R + G + B} \tag{1}$$

To classify the pixels, we set a green chromaticity threshold. Field pixels typically have a consistently high green ratio, even when lighting conditions change. In contrast, white field lines, or most other elements on the field, have a more balanced or reduced green ratio, despite possibly having high absolute green values in RGB space. With this approach, pixels exceeding the green chromaticity threshold are reliably classified as field regions.

3.2 Line Detection Improvements

In our line detection algorithm, we consider the center of each white scan region as a sample called a line spot. We take the set of vertical and horizontal line spots and project them to the field. We then perform several iterations of RANSAC to fit lines between these points, which will serve as field line candidates. If adjacent line spots within a fitted line are too far apart, we split the line candidate into two.

A common occurrence is that multiple candidates lie on the same actual field line, so once we have all the line candidates, we perform an additional candidate merging step similar to the approach used by B-Human [6]. That is, if the candidates are part of the same line, the following assumptions should hold:

1. The points on the line segment connecting two candidates should be colored white.
2. Points one line thickness away in the directions along the normal of the line segment (i.e., points slightly outside of the candidate line) should be colored green.

Using these assumptions, the merging step works by taking all candidates that are close in angle and drawing a set of samples in between them. We test the ratio at which samples in a connecting line segment are brighter and less saturated than the samples plus a small offset along the normal. If this ratio is high enough, we merge the lines.

Finally, we perform an extra rejection step for lines that do not fit our quality criteria. The current criteria are:

- A line segment must contain a certain minimum of inlier line spots.
- A line segment must be longer than a minimum length.
- A line segment must be shorter than a maximum length.

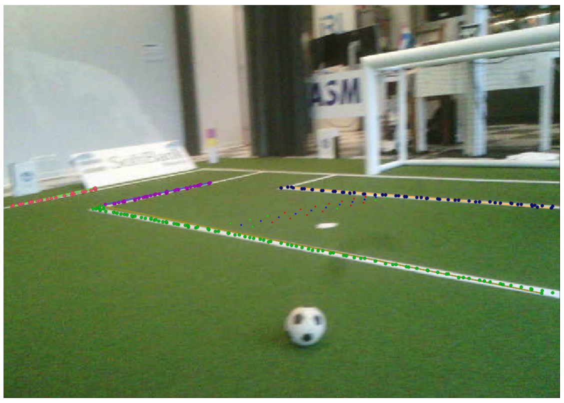


Figure 2: An example of lines and inlier line spots found by the line detector. A merge test is also performed between the outer lines of the goal area and the penalty area.

Throughout the year, we further tweaked the parameters for each field we played at.

3.2.1 Vision Based Localization

We improved our self-localization module significantly this year. While we still make use of only lines and no higher-level line crossings, the robustness of detected lines have been improved with local color comparison along the line normals, and we changed our line fitting over to a gradient-based approach.

3.3 Ball tracking

Ball position tracking now uses a ball tracker that estimates the ball state over time. The tracker maintains a best current estimate that includes position and, when supported by the observations, velocity. Behaviors and the game controller no longer use the latest projected ball detection, and instead query the tracker for the current ball estimate.

The tracker runs two estimations that reflect different motion regimes. For a moving ball, it maintains a motion model and estimates both position and velocity, propagating the state between observations. For a stationary ball, it maintains a state where the velocity is treated as zero and the estimate is kept stable. The tracker selects the estimate that is most consistent with the incoming measurements, see Figure 3.

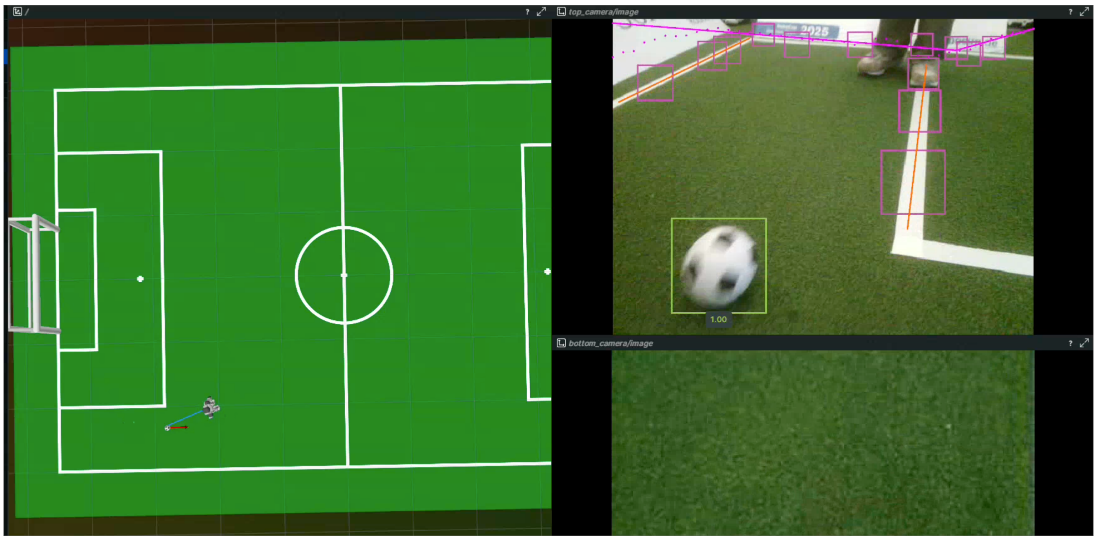


Figure 3: Estimated ball state for a moving ball. The tracked ball position is shown together with the estimated velocity vector, indicated by the red arrow.

3.4 Referee Pose Detection

We added a new module for referee pose detection. The goal of this module is to detect and classify the pose of referee at specific times during the game.

Our approach is a two-stage pipeline. First, we apply a pretrained YOLOv11 [7] pose detector to a center-cropped camera image to obtain candidate human poses. For each detected pose, the detector provides a set of keypoints together with a confidence score. We filter detections using a configurable threshold on pose quality to reject partial or unreliable poses, and only keep full poses that are sufficiently confident.

Second, each remaining pose is classified using a lightweight fully-connected network (FCN). The FCN takes the pose representation as input and outputs a gesture class. This decoupling of pose estimation and gesture classification lets us train a robust pose classifier with a relatively small amount of data. Since only limited labeled referee data was available, we trained the classifier on a small dataset recorded in our lab and at the RoboCup German Open 2025.

To improve generalization across lighting conditions, camera viewpoints, and referee body proportions, we apply heavy data augmentation during training. We publish the dataset on HuggingFace[1].

4 Motion

Stable and reliable motion is fundamental for the ability of the robot to perform well on the football field. Whether walking, maintaining an upright stance, or getting up from the ground after a fall, every action depends on precise coordination and control. This section describes developments in kinematics for precise motion control, key-framed motion creation in Blender, visualization tools for debugging robot poses, and body contour filtering to ensure the body is not accidentally classified as a line.

4.1 Kinematics

To control and optimize the robot's motion, it is necessary to know its precise position and orientation on the field. This is achieved by relating the robot's local reference frame, defined by its body and joint positions, to the global reference frame of the playing field. Kinematics uses this relationship to calculate the necessary joint angles and sequences of movements to achieve desired actions, such as walking, turning, or kicking.

Implementing these calculations in software, however, introduces its own challenges. Linear algebra, geometry, and related domains are usually agnostic to the reference frame they are expressed in, as long as consistency is maintained. In software implementations, such as the `nalgebra` crate for Rust, this means that a type implementing such spatial transformations is devoid from any information containing which reference frame any given coordinates are expressed in. This leads to the possibility that different frames (e.g., local and global space) are misused together, or that a given vector is expressed in different bases than it is interpreted as. Such errors are often hard to debug, since they require specific domain knowledge to uncover and typically do not trigger any exceptions or compilation errors.

In order to avoid such bugs, we leverage the Rust type system to keep track

[1]`https://huggingface.co/datasets/dutchnaoteam/pose-classification/`

of what reference frame a given spatial object is expressed in. Because of the information being recorded in the type system, this means we can catch the most common errors at compile-time, allowing for more thorough checks at run-time. Additionally, we can use generic parameters and type inference to provide generic methods that can transform an object in any given frame to its equivalent in the inferred frame, eliminating a lot of the opportunities for bugs to arise when the programmer is unsure about the spatial transformations required.

4.2 Blender Animation

Many of the NAO robot's movements, such as getting up, kicking, or taking a specific stance, are created using keyframed motions. In this approach, a sequence of joint positions for the required movement are defined at several moments in time. The robot then calculates and performs the intermediate joint movements needed to transition from one defined position to the next. This results in a complete, continuous motion.

A JSON-based file format was used to store keyframed motions. Although the format supports all features we wish to support, it is a cumbersome format to create or edit manually. Previous methods relied on manually adjusting joint angles and recording the joint positions needed to build reliable keyframes. In order to enhance productivity, a tool was developed to visually edit and animate the NAO's joint positions over time. This provided a far more intuitive and efficient way to create keyframed motions than the original methods

Specifically, a Blender add-on was implemented to import and export animations between Blender's native format and the JSON-based file. This way, the experience of team members in Blender and its animation tools can be exploited without needing to learn another tool. Although Blender lacks some features present in the custom file format, export parameters can be used to provide the necessary additional information.

4.3 NAO Visualization

Being able to review the robot's movements over time is essential for understanding issues in the code, especially when bugs are difficult to reproduce outside the specific conditions in which they occur. Because a lot of our debug tooling has been focused on Rerun and its ability to play back multi-modal recordings, we decided to expand on the debug information by logging the entire NAO pose based on the transformations exposed by our kinematics system.

Doing so means we are able to clearly visualize the pose expressed by the NAO.

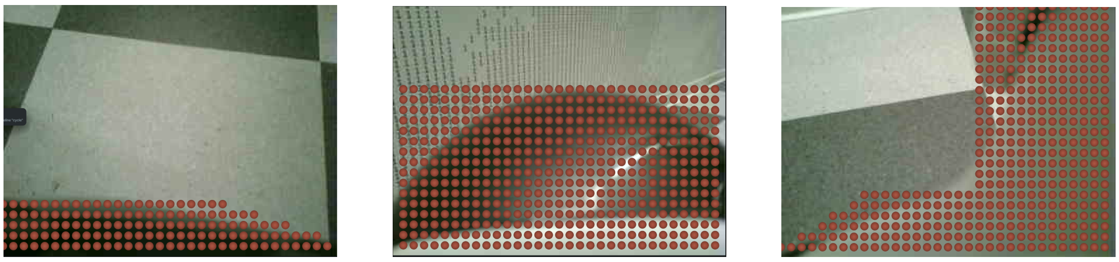
This allows our developers to intuitively grasp what the robot is doing in an instant, whereas logging the same information to print statement or a less intuitive data modality would require significant mental labor to understand what is happening, even if the logged data is theoretically equivalent in terms of information conveyed. This allows for rapid development cycles, accelerating every facet of debugging pertaining to locomotion, localization, orientation, or any other domain in proximity to the joint states.

4.4 Body Contour

The bottom camera of the NAO is able to see its own body, which causes problems for the line detection and the ball proposals. Because the body of the NAO is mostly white, the line detection detects many false positives within the body of the robot. The ball proposal model also makes a lot of proposals on the body, especially around the black shoulders. The ball detection model correctly avoids classifying these proposals as balls. However, due to the large number of ball proposals that are generated, the ball detection model needs to run more often than necessary, which increases the cycle time. This increase in cycle time means that other systems get starved of execution time, which is especially a problem for the walking engine.

To circumvent these issues, a method to filter the body from the image has been implemented called body contour [6]. Using forward kinematics, the robot is aware of the position of its body relative to its camera. This information can be used to determine whether a body part is present in the camera image. For example, to filter the chest from the camera image, three points on the chest, one on the left, one in the middle, and one on the right, are projected onto the camera image. These points can be used to create a boundary, where everything below the boundary is part of the body, see Figure 4. Therefore, the line detection and ball proposal systems should ignore everything below this boundary. This process is repeated for both feet and shoulders, after which the most important body parts are filtered out of the camera image.

While the body contour is working as intended, we found that it does not always project the body points onto the correct locations on the camera frame. It turns out that a good camera calibration is paramount for an accurate body contour. Even if the camera calibration was slightly off, the body points would be significantly misprojected onto the camera image, preventing the body contour from working correctly.



(a) chest (b) shoulder (c) chest and shoulder

Figure 4: Visualization of the body contour. The detected body parts of the NAO robot are projected in orange on the camera image.

4.5 Improved Walking Engine

The walking engine was updated to address limitations in the previous iteration. The previous monolithic engine is replaced by a modular pipeline structured around explicit gait states such as sitting, standing, starting, walking, and stopping (See Figure 5). Execution is organized into ordered phases covering preparation, step planning, gait generation, balancing, and finalization. This structure makes transitions between behaviors explicit and makes it easier to extend the walking engine further.

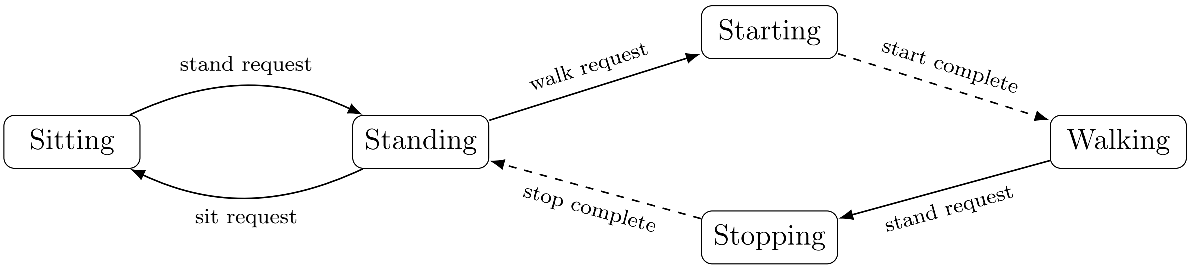


Figure 5: Gait state transitions in the walking engine.

Step planning is now centralized and operates on full six degree of freedom foot poses in ground coordinates rather than scalar offsets. The planner enforces acceleration limits, kinematic constraints, and bounded turning (See Algorithm 1 for more). Step timing and foot lift are scaled based on the commanded motion. Start and stop are handled as dedicated gaits, and support switching is restricted to a minimum fraction of the nominal step duration to avoid premature transitions.

Algorithm 1 Pseudocode overview of the step planning algorithm. Desired motion commands are constrained, converted into a target swing foot pose in the ground frame, and scheduled with appropriate timing and foot lift for gait generation.

```
Require: Desired command $u = (v_x, v_y, \omega)$, previous step $s_{k-1}$, support side $S$
Ensure: Planned step $s_k$
1: $u \leftarrow \text{clampAcceleration}(u, s_{k-1})$
2: $S_{\text{swing}} \leftarrow \text{opposite}(S)$
3: $T \leftarrow \text{targetPose}(u, S_{\text{swing}})$
4: $T \leftarrow \text{applyConstraints}(T)$
5: $(\text{duration}, \text{foot lift}) \leftarrow \text{schedule}(u, T)$
6: $s_k \leftarrow \langle u, S_{\text{swing}}, T, \text{duration}, \text{foot lift} \rangle$
```

Contact state estimation and balance control have been improved. Support foot detection is now based on a calibrated, weighted pressure model with prediction of support changes and explicit handling of unreliable measurements. The resulting support state is also used by downstream components such as odometry and ground alignment. Balance control uses filtered gyroscope feedback and applies ankle compensation only to the support foot.

5 Behavior

A behavior defines how the NAO robot acts in response to its environment and game situation. While a motion is a single series of joint angles that produces a specific movement, such as kicking a ball or standing up, a behavior combines multiple motions and chooses which to execute based on certain conditions. For example, the 'walk to ball' behavior involves detecting the ball and performing a sequence of walking motions to reach it. In this section, we describe our recent work on improving the robot's behaviors, which includes our `DNT-RL` framework, behavior simulation, team communication, obstacle detection using foot bumpers, path planner, and referee pose detection.

5.1 RL Behaviors

Most behavior engines in the SPL are hand-crafted, where every behavior is planned out in detail [8], [9]. However, developing behaviors like this is time-consuming since the behaviors have to account for many different situations. Therefore, using RL to replace the traditional behavior engine could be time-saving and allow for more flexibility and dynamic behaviors, as shown by WisTex

United (an SPL team from UT Austin and U. Wisconsin) [10], [11].

Last year, we developed a basic framework for training RL behaviors. This year, the framework was further developed and extended with a realistic 2D simulation that includes support for parallelized GPU training, resulting in `DNT-RL`. In subsection 5.1.1, we elaborate on the framework architecture, and in subsection 5.1.2, we give examples of RL behaviors we developed in the past year.

5.1.1 DNT-RL

We developed `DNT-RL`, a framework based on `skrl`, an open-source reinforcement learning library for Python [12]. We selected `skrl` for its modular architecture that enables extensive customization. Components such as model architecture, number of agents, and trained networks per agent can be customized independently. This flexibility makes `DNT-RL` suitable for specific use cases while still leveraging the library's robust training and evaluation methods.

The `DNT-RL` framework consists of three main components: a simulator to execute output actions, environments for specific behaviors, and policy training modules.

Simulation To simulate SPL games at a massive parallel scale, we developed a dedicated 2D physics engine designed to run efficiently on GPUs. The simulator is implemented entirely in Taichi [13], a high-performance parallel programming language embedded directly in Python. Its architecture is inspired by Genesis [14], but targets 2D environments instead. Each environment is represented by a `Scene` containing entities defined by their position, orientation, velocity, and mass. Therefore, the environment can be easily adapted to accommodate various situations for different behaviors, including the usage of multiple robots for multi-agent training. The simulator compiles optimized GPU kernels to execute many environments in parallel within a single batch, enabling millions of physics steps per second. Running these environments in parallel directly supports RL training. Specifically, it enables simultaneous policy rollouts across thousands of independent instances. This parallel data generation significantly accelerates the experience gathering required for policy optimization, reducing the overall training time.

Environments The `DNT-RL` framework requires specific environments to train RL behaviors. All environments inherit from a base environment that defines general steps applicable across behaviors. The observation space for each environment consists of the relative position of the robot, while the action space parameterizes a step of the walking engine. Since our action space is step-based, it is easily transferable across different robotic frameworks. To develop specialized behaviors, we create environments that inherit from the default environment. These specialized environments require two key functions: one to compute rewards and another to check episode termination. Episodes can end when either the goal is achieved or when specific termination events occur. For example, in goal-scoring behaviors, episodes end when goals are scored, while in goalkeeper behaviors, episodes terminate if opponents score.

Training The `DNT-RL` framework uses the `ParallelTrainer` from `skrl` for training, enabling both single-agent and multi-agent training as all agents train in parallel. The value and policy models share an initial linear layer, then branch into separate output layers to accommodate their different functions. For the policy model, a Gaussian distribution fits over the action space to enable continuous actions. The value model uses a deterministic final layer for precise output control. We implement PPO to train both models. After training completes, we export the policy model to ONNX format for seamless integration with the behavior framework [15]. The system achieves parallel training across environments by batching all environment components, including rewards, observations, and termination conditions. This approach synchronizes rollouts and updates across all training environments.

5.1.2 Developed behaviors

On top of the common NaoEnv infrastructure, the project defines several behavior classes that capture qualitatively different interaction patterns between the robot and its environment. Each behavior is framed as an RL task with its own objective, but all share the same underlying simulation, action space and training pipeline.

The **navigation behavior** trains the robot to move purposefully toward a spatial goal on the field. It exposes the agent to a variety of start/goal configurations and, optionally, obstacles. The emphasis is on learning smooth, efficient locomotion: approaching the target, orienting the body correctly, and stabilising near the goal without overshooting. When obstacles are present, the same behavior generalises toward basic obstacle-aware path selection, since the policy must balance progress with collision avoidance. This behavior has been successfully tested on the robot.

The **ball-handling** behaviors (dribbling and goalkeeping) extend this idea from pure self-motion to coordinated control of another dynamic object. In dribbling, the robot must position itself relative to the ball and push it toward a desired region, maintaining contact while respecting field boundaries. In goalkeeping, the robot instead protects a critical area, tracking the ball and positioning itself to block or intercept it. Both behaviors require the agent to reason about relative geometry between robot, ball and goal, and to use its limited action space to influence the coupled robot–ball dynamics.

The **search behavior** addresses a different aspect of autonomy: active perception. Here the robot's task is not to reach a predefined goal but to locate the ball using a limited field of view. The environment models a vision cone and a discretised field, encouraging systematic exploration and strategic scanning rather than random wandering. Success is defined in perceptual terms (bringing the ball into view), so the learned policy must coordinate motion and orientation to probe unseen regions of the field efficiently while staying within safe bounds. This behavior has been successfully tested and deployed on the robot, and was an important part of our behavior engine during RoboCup 2025.

5.2 Behavior Simulation

As had become evident in previous years, developing behaviors within a constantly changing and improving framework can be extremely difficult. Strategies that are effective in one version may become ineffective or even problematic as the framework evolves. With last year's move to Bevy, we built the first version of a behavior simulator. This version was limited to using only the abstracted output of the behavior module. While this simplified the implementation, it also introduced limitations, as the effects of other modules, such as odometry, localization, and pose estimation, were not included.

In a recent Bevy release, a new feature called *SubApps* was introduced, allowing the creation of multiple secondary applications with their own independent state. Using this functionality, we developed a new simulator capable of running five independent instances of the framework, using a simulated backend for low-level modules and with the vision modules removed. This new approach enables simulation of almost the entire robot behavior, including reinforcement learning behaviors. An additional advantage is a reduced sensitivity to behavior-specific changes in the framework.

The new version of the simulator was used to further improve the robot's behavior and resolve existing issues, without requiring physical robots or a field.

5.3 Obstacle Detection Using Foot Bumpers

Part of the robot's behavior involves responding to obstacles. This requires obstacles being detected. One way of achieving this is using bumper information. Each foot of the NAO robot contains an inner and outer bumper at its tip, as shown in Figure 6. Each cycle, we check whether the bumper detects pressure for

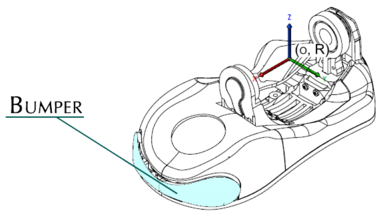


Figure 6: Foot bumpers of the NAO.

each sensor. Every detection is added to a counter that tracks the total number of pressure detections per foot. The individual sensor counts on each foot are summed, since the outer bumpers are triggered most often, and the sensitivity of the bumper sensors can vary a bit per robot. After a period of inactivity (no pressure detections), which is configured at 1.5 seconds, the counters will be reset.

When this count exceeds a threshold, which is empirically configured at 55 detections, we consider it an obstacle. The obstacle's location is estimated based on which foot passes the threshold. If only the left foot exceeds the threshold, the obstacle is assumed to be on the left side of the robot (and similarly for the right), with an estimated angle (θ) of $\pm$ 0.4 radians. If both the left and right foot exceed the threshold, the obstacle is estimated to be in front of the robot ($\theta = 0$). The distance d from the robot to the center of the obstacle is fixed at 0.22 meters. With this information, the relative coordinates can be calculated with

$$x_{\text{relative}} = d \cdot \cos(\theta)$$
$$y_{\text{relative}} = d \cdot \sin(\theta)$$

The relative coordinates are then converted to world coordinates, after which a dynamic obstacle is created at that location. The obstacle is given a radius of 0.15 meters and a time-to-live of five seconds. This obstacle can then be incorporated into the robot's path planning (see Section 5.5). All configurable values are determined empirically and logged to Rerun. Figure 7 shows an example of how a detected obstacle looks in Rerun.

5.4 Team Communication

Because our framework is designed in a modular fashion, we require coordination between different modules to ensure we uphold the constraints placed on robot-to-robot communication (i.e., the message budget). Because the priority and precedence between different modules wishing to communicate has to be a negotiation between all relevant modules, we split the responsibility into two parts. First, every module that seeks to communicate information with its peers can push its

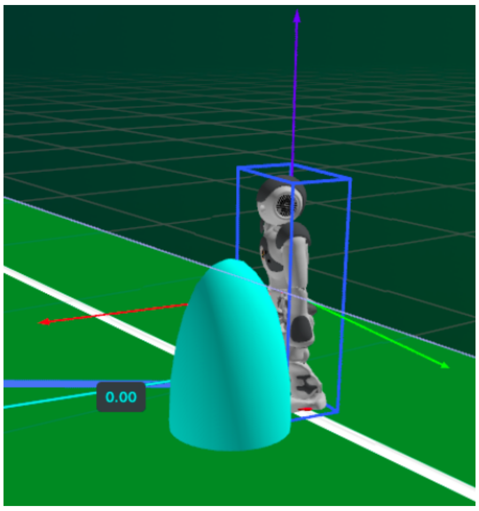


Figure 7: An example of a visualized obstacle that appears on the robot's left side.

message onto a priority queue based on a deadline for the message to be sent out by. This way, no module needs to know what other modules are running, or how they compare with regards to their priority directly. Instead, we base the priority on an interpretable time scale, and bundle and schedule the packets to be sent out on a best effort basis.

Under typical load conditions, the scheduler is able to send out every message within the time frame specified. Only under circumstances in which more packets are requested within a given interval than is possible to send out, do we delay messages to distribute the message budget allocation evenly across the match. By keeping the message rate close to the optimal for a given budget and mean rate, we also allow for flexibility at any point in the match by tweaking the various parameters associated with the packet scheduler. This means that higher-level decisions, such as game state or player roles, influence the rate of communication.

5.5 Path Planner

Path planning is essential for avoiding obstacles in advance. Our previous path planner emitted paths that were known to theoretically intersect with the objects it nominally avoided, requiring an exaggerated radius for the obstacles to be sufficiently avoided during game play.

Our new implementation sought to eliminate these problems by properly accounting for the geometries of both the obstacles and the robot, which were both modeled using circles. We kept the A* algorithm as the backbone to our path planning, but solved for the angles and winding directions around which obstacles were navigated analytically. This means splitting our A* states into qualitatively different states, e.g., whether we are following an arc around an obstacle or freely walking between two obstacles. The result of this new path planner is visualized in Figure 8.

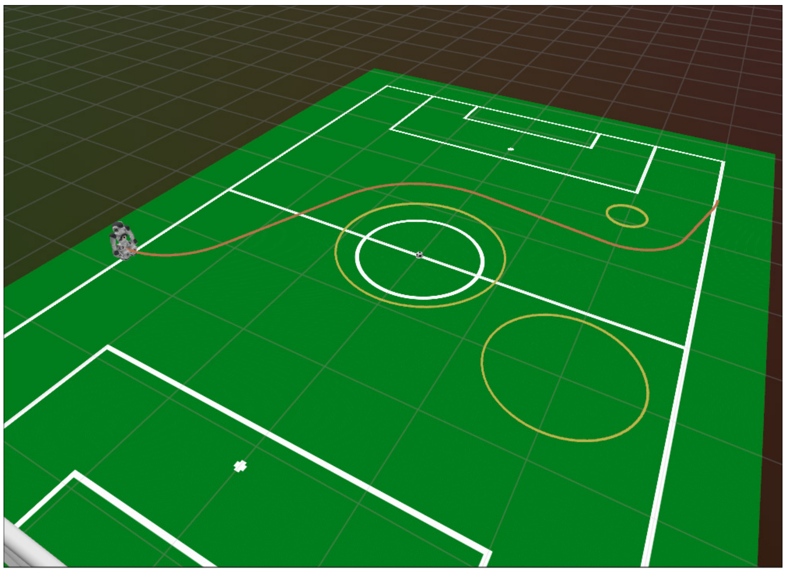


Figure 8: A visualization in Rerun of the new path planner. The chosen path is shown in orange and avoids the yellow obstacles.

5.6 Referee Pose Detection

Due to the integration of the referee poses into the games, we developed a deep learning based, pose recognition pipeline. The referee gives a visual indication by raising both hands over the head for 2 seconds. This indicates transition from the standby phase to the ready phase. Only after 35 or 45 seconds (Challenge Shield or Champions Cup rules respectively), a signal is sent over the network. The ready state is a maximum of 45 seconds, so the punishment of not recognizing the referee's pose, means that there is little or no time to walk from the initial position to the ready position. After a false positive recognition the robot will walk in the field, which will result in a "motion in standby" and penalizes all robots in place of the same team until the ready signal is received from the game-controller [16].

The referee is assumed to be standing in the center of the field. Therefore, the top camera of the NAO is pointed towards the center T-junction on the opposite of the robot's start position. The pre-processing consists of cropping and resizing the input image. The resizing is required to comply with the input size of the pose recognition model (See subsection 3.4). In a lab setting, the referee is often the only person in the image frame. However, during the public events such as the German Open, there can be an audience surrounding the field, which leads to multiple people in the frame. To reduce detections on other humans that are not the referee, the left and right side of the image are removed to focus on the center of the image frame.

False positives have big consequences due to the *motion-in-standby* penalty. To reduce the false positives, a visual signal is only marked as the ready pose if n consecutive ready poses are detected. The value of $n = 2$ consecutive detections worked best.

6 Workshops and Events

Next to working on our framework, DNT has attended multiple events and organized demonstrations and workshops with our Workshops and Events committee. Throughout the year, we were part of university open days, Science Park tours, and learning programs, where we arranged robot soccer demonstrations and programming workshops. The Workshops and Events committee primarily has an educational purpose, however, some demonstrations are given to gather sponsors and create publicity for our team.

The demonstrations include a short presentation about our team, the NAO robots we use, the SPL rules, and what is necessary to build an autonomous soccer framework. The level on technical details depends on the visitors' knowledge. After the presentation, we demonstrate a soccer match with two teams of either 1 or 2 robots, depending on how many are available. The demonstration takes 10 minutes and shows the visitors the abilities of the robots and our framework, as well as the rules and workings of the SPL matches.

Next to demonstrations, we give programming workshops. For high school students (ages 15-18), we have a Python workshop that teaches them about variables, operators, if-statements, and loops. For younger children (ages 10-12), we have a workshop in Scratch, in which they learn the basic concepts of programming in a kid-friendly way using blocks. For both workshops, we aim to create fun exercises that will enable them to make small programs on their own and understand the workings behind it. They are soccer-themed to relate them to DNT, and the goal is to teach them how we program the robots in a very simplified way.

6.1 Workshops and Demonstrations

The following subsections provide a list and descriptions of the bigger events DNT attended and participated in this year. Besides this, we also provided demonstrations for several Science Park tour groups, and a few for individual people curious about our team.

6.1.1 Careerday - March

The career day is an event to help highschoolers choose a study and career path, organized by JetNet. There are several companies that teach them about their career field and provide a short workshop. The day is divided into rounds of 30 minutes, in which a group of students visits our stand. We have 5 minutes to introduce our team and 20 minutes for an interactive activity. In our case, the activity was a short introduction to programming. The focus lies on the studies the team members do, and what kind of career opportunities the highschoolers can expect if they follow our path.

6.1.2 Demo Stichting Gelijke Kansen - April

In April, we received a visit from Stichting Gelijke Kansen, an organization dedicated to promoting equal opportunities in education. They were accompanied by several high school students from diverse backgrounds who may face fewer educational opportunities. The purpose is to show them different study options and help them find something they want to do. As Dutch Nao Team, we told them about the robotics field, which studies we did, and what we learn in our team. Of course, they also received a match demonstration with the robots.

6.1.3 Girls Day - April

The Girls Day is meant for girls to gain more interest in scientific topics. Forty two girls from high school VWO 1 come in groups of three after lunch to join a carousel of activities at Science Park, with our robot lab being one of these activities. Each group gets a demo and a presentation of 25 minutes in total to learn about robotics and AI.

6.1.4 Weekend van de Wetenschap - October

The "Weekend van de Wetenschap", or Weekend of Science, is a day when Amsterdam Science Park is open for everyone to visit and learn about the research that is done and what the students are learning. There are multiple activities, mainly focused at younger kids to let them learn about scientific topics in a fun way. On this day, DNT showed them a demonstration by doing a 2v2 soccer match. Every hour, there was a scheduled match that people could join to watch. Between the scheduled matches people were free to visit the lab and ask us questions.

6.1.5 UvA Open Dag - October

The Open Campus Day is an event where senior highschoolers can learn about the studies of the UvA. They visit the UvA to look around the campus, talk to students and visit presentations about different study directions. DNT gave a scheduled robot soccer demonstration every 45 minutes, and in the meantime, answered questions about DNT, our studies, and the UvA.

6.1.6 Scratch Workshop - November

In November, DNT was part of a series of after school lessons about computer science and robotics. The lessons are for children of around 10 to 11 years old who are interested in these topics. We organized their first lesson where we gave them a Scratch workshop. Scratch is a free coding platform meant for children to gain their first programming experience. The coding works with intuitive blocks, which are easy to use but often have the same logic as actual programming. The workshop has a fun robotic soccer theme to connect it to DNT. We start with a demonstration and a talk about what our team does, after which they learn about programming and robotics in the workshop.

6.1.7 Python Workshops - throughout the year

Throughout the year, we provided several Python workshops, aimed at senior highschoolers. The workshop is meant for beginners and starts out very simple with basic programming concepts, such as variables, operations, if-else statements and for-loops. Once they understand these concepts, they use them to create a small soccer simulation within a Pygame setup we made. Through the workshop, we give them the opportunity to learn about coding, something that is often not provided by the school itself.

6.1.8 StartUp Village visits - throughout the year

DNT has a partnership deal with StartUp Village (SUV) at Science Park. This deal entails that they sponsor us, and in turn they can schedule visits to our lab. When they have a group or company visiting them, they can come to the Intelligent Robotics Lab and we give them a talk about DNT, what we do, and show them a demo with our robots. These are mostly short 10-15 minute visits with a small group of around 20 people. They happen more often throughout the year. Examples of groups we hosted are a Swedish delegation, a delegation from Krakow, master students from the NTNU Trondheim, and groups from small companies.

6.2 German Open

This year's German Open took place in Nuremberg. Since this competition is played with the same rules and under similar conditions as the RoboCup later in the year, it serves as a good benchmark to check our progress.

During the competition, the team encountered networking-related issues that did not arise during testing in the lab environment. The robots had issues with connecting to the Wi-Fi at the venue, and it seemed like the new team communication module introduced in subsection 5.4, could cause intermittent crashes, so these issues needed to be addressed.

A major milestone for the German Open was the use of the new team communication module to send simple events between the robots, such as a detected whistle and a detected referee pose. Detecting the referee pose can only be done by one or two robots, by sending these events to all players in the team, the robots can start playing immediately once the game starts instead of waiting 35 seconds for the game controller signal.

Another significant milestone was the first integration of an RL behavior into the framework. The first RL behavior was a *ball-search* behavior for the striker. This simple behavior makes the striker walk around in the opponents half of the field, until a ball has been found by any of the team's players.

Finally, the spectators at the matches turned out to be problematic. Because the referee stood in front of the crowd, it was difficult for the robots to detect the referee's hand signals. The children in the crowd also caused trouble, as their high-pitched voices caused false whistle detections whenever they were cheering.

6.3 RoboCup Brazil

The RoboCup 2025 took place in Salvador, Brazil. This was the first competition with our `yggdrasil` framework where all key elements to obey the rules and play well were included. During the matches, we indeed saw that various components that we struggled with during earlier competitions, such as networking, communication and pose recognition, were working as expected. This resulted in a functioning framework that we scored multiple goals with. Although only one of the goals was valid because the other goals were made without passing the ball, this was considered a final test and approval of `yggdrasil`.

During the competition, we further improved the framework by adding multiple new features. Examples of the added features are a head motion manager, improved state transitions and a close-range ball search behavior. In addition to this, we further improved `DNT-RL`, resulting in a successfully trained goalkeeper in simulation and a successfully deployed step planner.

A big challenge during the RoboCup was the robot hardware. Since the NAO is discontinued, there was very limited robot repair available. Since we used the few months before RoboCup testing match play, we used the robots a lot for walking around the field. Because of this, lots of ankle gears were broken, which disables the robot's locomotion. Therefore, we made the decision to learn and start repairing the ankle gears ourselves. After members of the HULKs team explained how to replace the ankle gears at the start of the competition, we replaced all broken ankle gears for the remainder of the competition ourselves. This is a skill that will continue to be useful when robots break in our lab.

6.4 Beijing Masters

This year, we had the opportunity to participate in the RoboCup Asia-Pacific (RCAP) Beijing Masters, held in the Beijing Olympic Village. The event brought together teams from different parts of the world to compete in a setting where many teams were working with the Booster T1 platform for the first time.

For the Dutch Nao Team, the competition was a new experience, as the Booster T1 was used for the first time instead of the NAO robots from the SPL. The Booster T1 differed significantly from the NAO in hardware, sensors and motion capabilities. Therefore, `yggdrasil`, which was developed for the NAO, could not be used. The team had to adapt to the new platform within the limited available time.

Since the team stayed in Beijing for around three weeks, the focus was placed on becoming familiar with the open-source Booster T1 codebase from GitHub, rather than building a new one. During this period, incremental improvements were made to aspects such as locomotion stability, reinforcement learning-based walking, perception, passing, robot alignment, Voronoi-based strategies, role switching between attacker and defender, and overall team behavior. Extensive on-field tests were performed, with parameters adjusted and issues resolved throughout rehearsals and matches. Teammate detection was implemented but not yet tested in the competition.

During our time in Beijing, the first two weeks were spent preparing the robots for the competition, which lasted four days. Matches were played in a 5 vs 5 setting, and the team's performance included a win against a Portuguese team in the placement rounds. The Dutch Nao Team finished in 8th place overall.

In summary, the Beijing Masters provided a good introduction to the Booster T1 robots and the technical demands of the Humanoid League, offering hands-on experience that will be valuable as the team transitions away from using NAO robots in future competitions.

6.5 RoHOW

The Robotics Hamburg Open Workshop (RoHOW) 2025 is hosted by SPL team HULKs at the Hamburg University of Technology. The RoHOW aims to provide a platform for SPL teams to exchange knowledge with other teams, attend seminars and workshops, and engage in discussions on robot football and the latest advancements in robotics research. Teams also have the opportunity to request specific seminars or to offer one of their own. The members of the Dutch Nao Team participated in workshops and discussions, shared insights with other teams on their recent developments, and gathered new ideas for future improvements. As winners of last year's SPL quiz, the team was responsible for preparing this year's edition.

7 Plans for 2026

For the past 15 years, the Dutch Nao Team has been playing in the Standard Platform League (SPL) with the NAO robots. However, this year marks the final year of the Dutch Nao Team, as the team will switch to a new league and a new robot. Because of this, the team will be renamed to `whIRLwind Amsterdam`[2]. Therefore, the focus of the coming year will first be on developing a new framework for the new hardware, after which we will shift the focus towards more advanced behaviors.

7.1 SPL merging with the Humanoid League

The RoboCup Federation has announced that starting from 2026, the Standard Platform League and the Humanoid League will be merged into a single Humanoid Soccer League. There were already discussions about the SPL moving to another robot platform in 2027; however, following the bankruptcy of United Robotics Group, the manufacturer of the NAO robot used in the Standard Platform League, these decisions were further accelerated.

[2]whirlwind.team

This new Humanoid Soccer League has no standard robot platform, which means that teams are allowed, to a certain extent, to design their own robots or modify existing ones. This is a practice that has been standard in the Humanoid League for years; thus, they will have an advantage compared to the SPL teams in that regard. Due to our team not being from a technical university, and historically more software oriented, we do not have this experience in designing and building our own robots. Since we as a team want to play to our strengths, this being software and AI-engineering, we joined the majority of the other SPL teams in opting for off-the-shelf hardware, in the form of the Booster K1.

This K1 robot has numerous advantages compared to the NAO robots we have been using the past couple of years. The main advantage is that the K1 has between 100 and 200 times as much compute power as the NAO [3]. This will allow us to run more and larger AI models on our robots, such as RL behaviors, a fully RL walking engine, and object detection models, without being limited by their impact on cycle time. Our team has been longing for more possibilities to explore AI within robotics, this is an exciting development for us, which we are delighted to explore.

7.2 Organizational changes

This past year, we opted for a clear separation between the AI and software teams. However, this led to some occasional communication difficulties between the two teams, especially when collaboration between teams was necessary on projects such as integrating AI behaviors into the framework. For this reason, starting from next year, the AI and software teams will be disbanded. Instead, smaller teams will be formed, consisting of AI and software specialists, who will work together on specific projects.

7.3 Projects

With the team switching to a different robot, this means that the current framework cannot be used directly anymore. Instead of porting our current `yggdrasil` framework to the new robot, the team has decided to start developing a new framework from scratch once again. This new framework will employ a pub-sub architecture like ROS 2. This architecture should reduce some of the dependencies between the modules of the framework, which resulted in long build times and slowed down the development in `yggdrasil`.

While the team will be working on a new framework from scratch, it will use a

[3] www.generationrobots.com

ROS 2 based framework until at least the German Open 2026, to quickly get a framework capable of playing soccer matches. Some projects until the German Open include: basic behaviors such as walk to ball, a game-controller connection, object detection, localization and a kicking motion. After the German Open 2026, the team will be focusing more on long-term goals, such as implementing soccer strategy, the new framework, and a custom walking engine.

8 Contributions

The following list summarizes the contributions of everyone that worked on the tech report, in no particular order:

- **Vivienne Jansen** and **Amanda Jansen** were the main coordinators of this year's team report.
- **Mark Honkoop** worked on the body contour, damage prevention, and the RL integration in the framework.
- **Julia Blaauboer** worked on the improved path planning, and added a progress bar to `Sindri`.
- **Macha Meijer** acted as AI tech lead. Furthermore, she developed DNT-RL together with Gijs, collaborated on various detection models, and developed behaviors during the competitions.
- **Marilène Oud** worked on various behavior fixes during the Beijing Masters.
- **Rick van der Veen** worked on the referee-pose detection, the head-motion manager, as well as the rerun-control module.
- **Fiona Nagelhout** acted as lead of the workshop and events team. She was responsible for organizing all workshops given by the Dutch Nao Team.
- **Harold Ruiter** acted as the team lead. Furthermore, he implemented the vision-based localization, the improved line detection module, and the moving ball position predictor.
- **Fyor Klein Gunnewiek** implemented the behavior simulation, and worked on many behavior fixes.
- **Kim Verkuijl** acted as the treasurer since September. Furthermore, she worked on the foot bumper obstacle detection, the whistle communication, and the low battery notification.
- **Gijs de Jong** acted as software tech lead. Furthermore, he worked on locomotion, planning behavior, developed DNT-RL together with Macha primarily focusing on the GPU-accelerated physics simulator, collaborated on various detection models, and debugging/visualization capabilities.

9 Conclusion

This year, the Dutch Nao Team worked on developing the more advanced modules of its custom framework `yggdrasil`, and has achieved the goals set out in the previous year: being able to play matches and score goals. This report discussed details of all projects that were completed over the past year to achieve those goals. Additionally, it outlined future plans for the new team `whIRLwind Amsterdam`, which will be participating in the new Humanoid Soccer League.

10 Final Chapter

Over the years, the Dutch Nao Team has built upon the contributions of many dedicated students and researchers. As this is the final report under the Dutch Nao Team name, we would like to offer a special acknowledgment to the works and individuals listed below, whose efforts have laid the foundation for whIRLwind Amsterdam, and upon which we will continue to build.

2025 Dutch Nao Team – Technical Report, 2025
Vivienne Jansen, Amanda Jansen, Mark Honkoop, Julia Blaauboer, Macha Meijer, Marilene Oud, Rick van der Veen, Fiona Nagelhout, Harold Ruiter, Fyor Klein Gunnewiek, Kim Verkuijl, Gijs de Jong

2024 Dutch Nao Team – Technical Report, 2024
Harold Ruiter, Gijs de Jong, Macha Meijer, Marina Orozco González, Mark Honkoop, Julia Blaauboer, Rick van der Veen, Fyor Klein Gunnewiek, Morris de Haan, Fiona Nagelhout, Joost Weerheim, Stephan Visser, Juell Sprott and John Yao [1]

2023 Dutch Nao Team - Technical Report, 2023
Gijs de Jong, Harold Ruiter, Derck W.E. Prinzhorn, Jakob Kaiser, Mark Honkoop, Joost Weerheim, Fyor Klein Gunnewiek, Ross Geurts, Dario Xavier Catarrinho, Stephan Visser, David Werkhoven, Rick van der Veen and Madelon Bernardy [17]

2022 Dutch Nao Team - Technical Report, 2022
Lex Bolt, Fyor Klein Gunnewiek, Hidde Lekanne gezegd Deprez, Lasse van Iterson, Derck Prinzhorn [18]

2021 Dutch Nao Team - Technical Report 2021
Jakob Kaiser, Hidde Lekanne gezegd Deprez, Wike Duivenvoorden, Pim Heeman, Rogier van der Weerd and Thomas Wiggers [19]

2020 Dutch Nao Team – Technical Report, 2020
Pim Heeman, Thomas Wiggers, Hidde Lekanne gezegd Deprez and Wouter Zwerink [20]

2019 Dutch Nao Team – Technical Report, 2019
Thomas Wiggers, Douwe van der Wal, Hidde Lekanne gezegd Deprez, Wouter Zwerink and Pieter Kronemeijer [21]

2018 Dutch Nao Team – Technical Report, 2018
Hidde Lekanne gezegd Deprez, Douwe van der Wal, Pieter Kronemeijer, Michiel van der Meer, Linda Petrini, Thomas Groot, Jier Nzuanzu and Caitlin Lagrand [22]

2017 Dutch Nao Team – Technical Report, 2017
Douwe van der Wal, Pieter Kronemeijer and Caitlin Lagrand [23]

2016 Dutch Nao Team – Technical Report, 2016
Caitlin Lagrand, Michiel van der Meer, Jonathan Gerbscheid, Thomas Groot, Sebastien Negrijn and Patrick de Kok [24]

2015 Team Qualification Document for RoboCup, 2016
Patrick de Kok, Sébastien Negrijn, Mustafa Karaalioğlu, Caitlin Lagrand, Michiel van der Meer, Jonathan Gerbscheid, Thomas Groot and Arnoud Visser [25]

2014 Team Description for RoboCup 2014 – João Pessoa, Brasil
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 [26]

2013 Team Description for RoboCup 2013 in Eindhoven, the Netherlands
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 [27]

2012 Dutch Nao Team – Team Description for RoboCup 2012
Camiel Verschoor, Duncan ten Velthuis, Auke Wiggers, Michael Cabot, Anna Keune, Sander Nugteren, Hendrik van Egmond, Hessel van der Molen, Richard Rozeboom, Inge Becht, Maarten de Jonge, Richard Pronk, Chiel Kooijman and Arnoud Visser [28]

2011 Dutch Nao Team – Team Description for RoboCup 2011
Duncan ten Velthuis, Camiel Verschoor, Auke Wiggers, Sharon Gieske, Anna Keune, Sander Nugteren, Michael Cabot, Eszter Fodor, Maurits van Bellen, Timothy Dingeman, Tim van Rossum, Steven Laan, Arnoud Visser [29]

2010 Dutch Nao Team – Team Description Paper – Standard Platform League – German Open 2010
Arnoud Visser, Robert Iepsma, Maurits van Bellen, Ravi Kumar Gupta and Bardia Khalesi [4]

References

[1] H. Ruiter et al., “Dutch nao team - technical report,” Tech. Rep., Dec. 30, 2024.

[2] R. Federation, *Objective*. Accessed: Dec. 8, 2025. [Online]. Available: `https://www.robocup.org/objective/`.

[3] R. Federation, *RoboCup Standard Platform League*. Accessed: Dec. 8, 2025. [Online]. Available: `https://www.robocup.org/leagues/5`.

[4] 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*, Jan. 30, 2010.

[5] S. Oomes, P. Jonker, M. Poel, A. Visser, and M. Wiering, “The dutch aibo team 2004,” Jul. 1, 2004.

[6] T. Röfer et al., *B-Human team report and code release 2019*, Only available online: `http://www.b-human.de/downloads/publications/2019/CodeRelease2019.pdf`, 2019.

[7] G. Jocher, J. Qiu, and A. Chaurasia, *Ultralytics YOLO*, version 8.0.0, Jan. 2023. [Online]. Available: `https://github.com/ultralytics/ultralytics`.

[8] A. Essig, P. Gleske, K. Nölle, M. Schmidt, H. Sieck, and B. Wetters, “Hulks team research report 2021,” HULKs e. V., Hamburg University of Technology, Am Schwarzenberg-Campus 3, 21073 Hamburg, Germany, Tech. Rep., 2021. [Online]. Available: `https://hulks.de/_files/TRR_2021.pdf`.

[9] T. Röfer et al., *B-Human team report and code release 2022*, Only available online: `https://raw.githubusercontent.com/bhuman/BHumanCodeRelease/coderelease2022/CodeRelease2022.pdf`, 2022.

[10] A. Labiosa et al., *Reinforcement learning within the classical robotics stack: A case study in robot soccer*, 2025. arXiv: `2412.09417 [cs.RO]`. [Online]. Available: `https://arxiv.org/abs/2412.09417`.

[11] A. Labiosa et al., “Reinforcement learning within the classical robotics stack: A case study in robot soccer,” *arXiv preprint arXiv:2412.09417*, 2024.

[12] A. Serrano-Muñoz, D. Chrysostomou, S. Bøgh, and N. Arana-Arexolaleiba, “Skrl: Modular and flexible library for reinforcement learning,” *Journal of Machine Learning Research*, vol. 24, no. 254, pp. 1–9, 2023. [Online]. Available: `http://jmlr.org/papers/v24/23-0112.html`.

[13] Y. Hu, T.-M. Li, L. Anderson, J. Ragan-Kelley, and F. Durand, “Taichi: A language for high-performance computation on spatially sparse data structures,” *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, p. 201, 2019.

[14] Genesis, *Genesis: A universal and generative physics engine for robotics and beyond*, Dec. 2024. [Online]. Available: `https://github.com/Genesis-Embodied-AI/Genesis`.

[15] ONNX Community, *ONNX: Open neural network exchange*, `https://onnx.ai`, Accessed: 2025-04-21, 2019.

[16] RoboCup, *Robocup standard platform league (nao) rule book*, `https://spl.robocup.org/wp-content/uploads/SPL-Rules-2025.pdf`, Accessed: 2025-11-30, 2025.

[17] G. de Jong et al., "Dutch nao team - technical report," Tech. Rep., Dec. 31, 2023.

[18] L. Bolt, F. K. Gunnewiek, H. L. gezegd Deprez, L. van Iterson, and D. Prinzhorn, "Dutch nao team - technical report," Tech. Rep., Dec. 26, 2022.

[19] J. Kaiser, H. L. gezegd Deprez, W. Duivenvoorden, P. Heeman, R. van der Weerd, and T. Wiggers, "Dutch nao team - technical report," Tech. Rep., Jan. 16, 2022.

[20] P. Heeman, T. Wiggers, H. L. gezegd Deprez, and W. Zwerink, "Dutch nao team - technical report," Tech. Rep., Jan. 1, 2021.

[21] T. Wiggers, D. van der Wal, H. L. gezegd Deprez, W. Zwerink, and P. Kronemeijer, "Dutch nao team - technical report," Tech. Rep., Jan. 15, 2020.

[22] H. L. gezegd Deprez et al., "Dutch nao team - technical report," Tech. Rep., Dec. 31, 2018.

[23] D. van der Wal, P. Kronemeijer, and C. Lagrand, "Dutch nao team - technical report," Tech. Rep., Dec. 31, 2017.

[24] C. Lagrand, M. van der Meer, J. Gerbscheid, T. Groot, S. Negrijn, and P. de Kok, "Dutch nao team - technical report," Tech. Rep., Oct. 14, 2016.

[25] P. de Kok et al., "Team qualification document for robocup 2016 – leipzig, germany," Tech. Rep., Jan. 1, 2015.

[26] P. de Kok et al., "Team description for robocup 2014 – joão pessoa, brasil," Tech. Rep., Jun. 6, 2014.

[27] P. de Kok et al., "Team description for robocup 2013 in eindhoven, the netherlands," Tech. Rep., May 25, 2013.

[28] C. Verschoor et al., "Dutch nao team – team description for robocup 2012," Jun. 1, 2012.

[29] D. ten Velthuis et al., "Dutch nao team – team description for robocup 2011," Jul. 1, 2011.