Software Survey 2026

Team name

NUbots

Which division(s) are you applying for? If your used software differs between divisions, please fill out the survey once per division.

Small Size (height < 110 cm, weight < 15 kg)

Is your software fully or partially OpenSource? If so, where can it be found?

Fully; it can be found at https://github.com/NUbots.

Are you using any software developed by other teams? If so, list every component that you are reusing and the team that originally developed it.

No. But we do use Bit Bot's dataset to train YOLO.
https://github.com/bit-bots/TORSO_21_dataset

Are you using any datasets in your research? If you are using your own datasets, are they public?

Yes. We have one to train YOLO and two others for segmentation; they can be found here https://huggingface.co/NUbots.

Please list the scientific publications your team has made since the last application to RoboCup (or if not applicable in the last 2 years).

- [Awaiting Proceedings Release] Sims, Y., Mendes, A., Chalup, S.: Zero-Shot Environmental Sound Classification Onboard a Resource-Constrained Robot. In: RoboCup 2025 Symposium. Salvador (2025)
- O'Brien, T.: Sampling-based predictive control for non-prehensile manipulation. In: Australasian Conference on Robotics and Automation (ACRA). Perth (2025)
- Burgin, W., O'Brien, T.: Real-time humanoid state estimation with imu, kinematic odometry, and visual slam. In: Australasian Conference on Robotics and Automation (ACRA). Perth (2025)

Are there any other contributions you would like to share with the RoboCup community?

We would like to save our debugging tool NUsight, and our automatic synthetic data annotator NUpbr. Both are found here https://github.com/NUbots.

Which approach are you using to generate the robot walking motion?

At the moment, NUbots currently use an open-loop walk engine which creates polynomial splines representing three-dimensional trajectories of the feet and torso pose between steps. The trajectories are generated in the planted foot frame as a function of the desired walk command (v_x, v_y , v_yaw). The engine interpolates over these splines to find the next target position for the feet, which is then converted into servo joint angles using inverse kinematics. NUbots use a combination of an approximate analytical and optimisation based solution for the inverse kinematics of the NUgus. The approximate analytical solution is used to warm-start a levenberg marquardt style algorithm from the tinyrobotics [?] library.
This walk engine is inspired by Bit-Bots' Quintic Walk, based off Rhoban's
Quintic Walk and Rhoban's IK Walk.
However, the team intends to implement a new walk using reinforcement learning (RL). Preliminary steps have been taken to test policies in the MuJoCo simulation environment.

Which approach are you using to generate other motions of the robot (e.g. kicking, standing up)?

Kicking is done with a larger step in the walk engine which is based on polynomial splines. Standing up from both prone and supine is currently done with a scripted animation using timed keyframes. However, the team intends to add dynamic feedback using sensors such as the IMU in a hybrid scripted animation.

Do you have a kinematic or dynamic model of your robot? If so, how did you create it (e.g. measure physical robot, export from CAD model)?

The kinematic model of the robot was derived in uniform robot description format (URDF) from a CAD model.

What approaches are you using in your robot's visual perception?

The team uses a hybrid system of two architectures, namely the Visual Mesh which outputs

field-lines themselves, goals, balls, and other robots, and YOLOv11 which outputs field-line intersections, goals, balls, and other robots.
The Visual Mesh is a convolutional neural network (CNN) for object detection based on a mesh which samples pixels from an image. The mesh is depth independent and the number of mesh points is small enough to allow a CNN to run in real-time on a humanoid robot. Tests on the CPU from the Intel® NUC7i7BNH show that a nine layer visual mesh had an execution time of 2.44ms.(Houliston & Chalup, 2018) Furthermore, the visual mesh does not degrade in accuracy when objects occur at different distances, due to the depth independence introduced by using the mesh. The visual mesh was used to detect a soccer ball on a field up to 10m away from the camera.(Houliston & Chalup, 2018) It can detect soccer balls, robots, field lines and goal posts, however it is built assuming the object to be detected is spherical.
A YOLOv11 neural network is used for redundancy to classify field-line intersections, goal posts, robots, and balls for redundancy. In particular, field-line intersections such as L-, T- and X-intersections are used in the localisation algorithm. The world position derived from the image coordinate of the bounding box output is used to determine where the robot is based on multiple hypotheses and is refined over time.

Are you planning with objects in Cartesian or image space? If you are using Cartesian space, how do you transform between the image space and cartesian space?

We use three-dimensional Cartesian space. The Visual Mesh already transforms from image space. Whereas, we use a spherical to Cartesian function on the output after Yolov8.
https://github.com/NUbots/NUbots/blob/36586a572afa5967802677c9a48ecff3941e8f82/shared/utility/vision/projection.hpp#L206-L252

Do you have some form of active vision (i.e. moving the robots camera based on information known about the world)?

Yes. The robot moves its head to track a ball if the vision-system has detected one. If the robot loses sight of the ball, then it will begin moving its head around to widen its field of view. Once it sees the ball, it will then centre its head on the ball.

What approach are you using to localize your robot?

Localisation is achieved through a combination of our Odometry system and an optimisation routine which is solved using NLopt's Constrained Optimisation BY Linear Approximations (COBYLA) algorithm.(Johnson, 2007)
The loss for the non-linear optimisation algorithm is split into three parts.
- Field Line Alignment Cost: This component measures the alignment of predicted field lines with observed ones. It is calculated as the squared Euclidean distance between field line points and the nearest point on any observed line, scaled by a predefined weight.
- Field Line Intersection Cost: This assesses the accuracy of predicted field line intersections against observed intersections. It is computed similarly through the squared distances between predicted and observed intersections.
- Field Line Intersection Cost: This assesses the accuracy of predicted field line intersections against observed intersections. It is computed similarly through the squared distances between predicted and observed intersections.
The whole cost-function is thus the sum of these above subject to the following constraints.
- State Bounds: Limits the allowable state changes between optimization steps to ensure the solution does not jump an unrealistic amount between updates.
- Minimum Field Line Points: The algorithm requires a minimum number of field line points to run the optimization to ensure sufficient data for accurate estimation.
- Robot Stability: Optimization will not proceed if the robot is in an unstable state (e.g., falling.
In addition, odometry estimates the pose over time from an initial starting position using a Mahony Filter(Mahony et al, 2008) for roll and pitch and an anchor point method (dead-reckoning of kinematics) for translation and yaw, see the blog by Stéphane Caron for a detailed overview of this approach. https://scaron.info/robotics/floating-base-estimation.html

Is your team performing team communication? Which communication protocol are you using?

Yes. We are using the standard Humanoid League protocol.

What approach are you using for navigation? Are you avoiding obstacles?

The robot's path-planning is done using the robot's walk engine which converts desired

trajectories (v_x, v_y , v_yaw) into foot trajectories. The path-planning logic avoids an obstacle by adding waypoints around it and dividing the overal trajectory into smaller trajectories. The waypoints are calculated as coordinates offset from that of the obstacle and at whicever side depending on the position in the field relative to the goals, etc.

How is the behavior of your robots structured? (e.g. Behaviour Trees)

The behaviour system is driven by the Director, a framework and algorithm for a reactive tree-style system that emphasises modularity and transitions.(Sims & Houliston, 2023)
The Director uses the concepts of tasks and providers. Tasks are requests for an action to happen, such as `walk to ball', `walk', `left hip yaw servo', each defined as a Protobuf message. Each task may contain some information, i.e. the `walk' task may have velocity information, and `left hip yaw servo' task may have joint angle, gain and torque.
Providers provide the functionality for a particular task by either achieving the task directly or emitting subtasks. For example, the walk engine will be a Provider for the `walk' Task. It will then call subtasks itself to move the arms and legs. A Provider can only provide for one Task at a time. If both the Walk and Kick are requesting subtasks for the legs, only one will take control of the legs. Tasks have an associated priority that determines who takes control.
The Director aims to be highly modular, with small Providers that provide the functionality for specific tasks such as `look at ball' or `walk to ball'. There are Provider groups that provide the functionality for the same task under different conditions. There may be a group of Providers that all provide for the `Striker' Task, with each only applicable for a particular game state.
Some Providers need the system to be in a particular state to run. For example, a `Kick' Provider may require that the robot is in a standing position before running. The Director algorithm will not consider it a valid solution unless the robot is standing. Providers can also declare that when they run, the system will achieve a particular state. If the `kick' has a high enough priority then the Director will make the Provider run that will result in a standing state so that the kick can then run.
Both the referenced pre-print on arXiv and our NUbook Director page give an extended description of the Director. https://nubook.nubots.net/

Are you simulating your robot? If so, which simulator are you using and for what purpose do you use simulations?

We simulate our robots in Webots. https://cyberbotics.com/
We simulate in order to test the robot's behaviour and planning. This tool allows members to test at home and to focus solely on the behaviour in a vacuum without any inconveniences or hardware.

What operating system is running on your robot and which middleware are you using (for example Ubuntu 22.04 and ROS2 Galactic)?

The operating system is Linux, the distribution is Arch 64-bit. The middleware is NUClear, a real-time message-passing framework for C++. https://github.com/Fastcode/NUClear

Is there anything else you would like to share that did not fit any previous question?

We intend to form a secondary joint team with rUNSWift and RMIT Redback Bots in the middle division.
We have developed our own web-based debugging tool, found here.
https://github.com/NUbots/NUsight2
We also develop our own firmware for the subcontroller and other electronic devices.
https://github.com/NUbots/NUcontroller
https://github.com/NUbots/NUware