Survey response 2

Software

Team Name
NUbots

Is your software fully or partially OpenSource. If so, where can it be found:
Fully OpenSource

Do you have a kinematic or dynamic model of your robot(s)? If so, how did you create it (e.g. measure physical robot, export from CAD model)?
Yes, a URDF file is generated from an Onshape CAD assembly (using https://onshape-to-robot.readthedocs.io/en/latest/) which is parsed using tinyrobotics C++ library for both kinematics and dynamics.

Are you using Inverse Kinematics? If so what solution (analytic, (pseudo)inverse jabcobian, etc...) are you using?
Yes, Inverse Kinematics is achieved using 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 [9] library.

Are you simulating your robot? If so what are you using simulation for?
Yes, we have a simulation environment in Webots from the virtual league. It is used extensively for rapid development of all our modules.

What approach are you using to generate the robot walking motion?
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 (vx, vy , vyaw). 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

What approach are you using to generate motions for standing up?
Key frame animations

What approach are you using to generate kicking motions?
Key frame animations

Do you use any other motions than the previously mentioned? If so, what approaches are you using to generate them?

Which datasets are you using in your research? If you are using your own datasets, are they public?
We use our own datasets which are being prepared for publication

What approaches are you using in your robot's visual perception?
The Visual Mesh underpins the vision system and is used for sparse detection of balls, points on the field, field lines, goal posts, and other robots. The visual mesh is a convolutional neural network 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. 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. It can detect soccer balls, robots, field lines, and goal posts, however, it is built assuming the object to be detected is spherical. The visual mesh is created using the following geometric assumptions for each image: The camera lens type, height, and orientation with respect to the ground are known. The object is spherical with a known radius and remains on the ground. The height, orientation, and radius are used to create a set of unit vectors based at the camera position. Any vector with origin at the camera can be thought of as a ray of light traveling towards the camera. If the vector is within the field of view of the camera it will be collected by the lens. The vector is mapped to a point on the image using the lens projection equation, provided the point is within the bounds of the image sensor of the camera. The visual mesh is formed by taking an array of vectors and associating each of them with a point and pixel in the image. The array of vectors is specially constructed using two equations to efficiently sample the space around the camera for the object. This means the points (sample points) on the image will efficiently sample the image for the object. Because the vectors are generated in the space around the camera, the sample is consistent despite changes in lens type, image resolution, and the apparent size of an object due to distance. Each sample point is connected to its six closest neighbors and these connections become the edges of the mesh. A fully convolutional neural network is used on the mesh where each sample point and its neighbors become the input to a convolution. All layers in the network use seven points convolutions since there are six neighbors for each sample point. A parameter controls the number of sample points on the object which determines the level of detail available to the network. The height and orientation of the camera are tracked using the kinematics and inertial measurement unit of the robot. The radius of the soccer ball is given. The unit vectors only need to be calculated once for each height and radius pair. As the robot walks its height varies. A series of visual meshes for different heights are calculated on startup. For each image, this information is recorded and the appropriate visual mesh is chosen. A binary search pattern is used to select the vectors that will become points on the image based on the camera's orientation. The partitions of the binary search pattern are built on startup and the search is run in real-time. Every training and run-time image is converted into the mesh before the network is run. The network labels each point of the mesh with the probability it belongs to an object class. A post-processing algorithm based on the probabilities of the mesh can cluster the sample points to determine an object's position. From the visual mesh, a series of specialized detectors are employed to detect field edges, balls, and goal posts. All calculations in all detectors are done in three-dimensional world coordinates. Firstly, all points that the Visual Mesh has identified as either field points or field line points are clustered into connected regions and each cluster is then either merged or discarded using some heuristics until a single cluster remains. Finally, a convex hull algorithm is applied to the final cluster determining the edge of the field. The ball detector forms clusters out of all the points that the Visual Mesh has identified as ball points. Specifically, the clusters are formed from Visual Mesh points that are identified as being a ball point, but have at least 1 neighbor which is not a ball point. This allows us to form clusters of ball edge points. Any clusters which are not below the field edge are discarded. A circular cone is then fitted to each cluster. The cone axis is determined from the line segment between the center on the camera and the average of all ball edge points. The radius of the cone is determined by the maximum distance between the ball edge points and the average of all of the ball edge points. Different heuristics, such as degree of circle fit and different distance metrics, are then used to discard cones. The goal post detector follows a similar structure to the ball detector. Clusters are formed from goal post edge points and any clusters that do not intersect the field edge are discarded. The bottom center point of the goal post is then found by averaging the edge points. The distance to the goal posts is determined and if there are multiple goal posts detected an attempt is made to assign leftness and rightness to each post. The field line points from the visual mesh are collected and fed into the localization system, which will determine how those points are used. While the vision system does have the capability to classify robot pixels, a future goal is to use this information to know the location of individual robots. This information could then be used in the behavior system for obstacle avoidance. The Visual Mesh performs well given it has adequate data. Its use of constant sampling density makes it ideal for seeing objects

far away, such as balls. The Mesh runs very quickly, and can reach over 100fps.

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?

Cartesian space. The output of the visual mesh directly provides unit vectors in Cartesian space (using the lens projection equation) which are used in all planning, estimation and behaviour modules.

The vectors are projected onto field plane exploiting known camera pose (IPM) or object characteristics (eg ball radius).

How is your robot localizing?

Localisation is achieved through a combination of our Odometry system and a Particle filter.

Odometry:
Odometry estimates the pose over time from an initial starting position using a Mahony Filter for roll and pitch and an anchor point method (dead-reckoning of kinematics) for translation and yaw, see blog https://scaron.info/robotics/floating-base-estimation.html for a detailed overview of this approach.

Particle Filter:

At initialisation, n particles are sampled from a multivariate normal distributions on either side of the field facing towards the centre, to produce initial hypothesis' of the robots starting position. Additionally, a discretized field line distance map is pre-computed at startup which encodes the minimum distance to an occupied cell (field line).
Measurement update:
The measurement update involves weighting each particle. The general idea is to weight particles higher if more field line point observations are closer to field lines. For each particle, the field line point observations are projected onto the field plane (using Odometry) and mapped into an index in the field line distance map. The weight of the particle is then calculated using simple inverse function

$$
W_n = 1/(\delta + \epsilon)
$$

where W_n is the weight of the nth particle, $\delta = \sum_{i=1}^{k} (\delta_i)^2$ is the total sum of squared distance values obtained from the map for all field line point observations, and ϵ is machine epsilon to prevent numerical issues.
Time update:
The time update does not directly involve propagating particles forward in time using a motion model since the latest Odometry information is used within the measurement update. Instead, the time update involves simply adding noise to the particles to simulate the uncertainty in the robot's motion.
Resampling:
Resampling refines the set of particles in Particle Filter localisation based on their calculated weights as follows

1. Normalize Weights: Particle weights are adjusted so their sum is 1, ensuring each represents its proportional likelihood.

2. Particle Selection: Using the normalized weights, particles are probabilistically chosen. The original set of particles is replaced with the newly resampled set, now skewed towards more likely positions and orientations of the robot.

Is your robot planning a path for navigation? Is it avoiding obstacles? How is the plan executed by the robot (e.g. dynamic window approach)?

This is currently not completed, however, is under development.

How is the behavior of your robot's structured (e.g. Behavior Trees)? What additional approaches are you using?

The behaviour system is driven by the Director, a framework and algorithm for a reactive tree-style system that emphasises modularity and transitions.

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 paper and our Director page on our NUbook documentation give an extended description of the Director.

Logic Implementation:

There are five distinct layers to our system.
Actuation:Providers that directly control the servos. Includes groups of servos, sequences of servos, and kinematics calculations.
Skill: Physical motions that the robot can perform such as kicking, walking and getting up.
Planning: Higher-level calculations that take a task and mathematically compute how to utilise skills to execute that task. Includes path planning, where the Provider receives a location to walk to and calculates a velocity for the walk subtask.
Strategy: Utilises environment information to determine what to do. Includes walking to the ball, finding the ball, aligning with the goals, and more.
Purpose: This layer represents the robot's overall goal. In the context of soccer, this involves using GameController information to determine what game position to play in and calling the relevant strategy subtasks to play in that position.

The Director tree starts with the Soccer Provider that determines what position to play in. If it's penalised, it will stand still. Otherwise it will use configuration information to play as either a striker, defender or goalie. Before RoboCup 2024 we plan to integrate robot-to-robot communication with this system to dynamically choose the position. The official RoboCup Protocol is currently used. At the root level, the fall management system is running at a higher priority than the Soccer subtree.

All robots will walk to a designated point on the field in the ready state.

In the playing state, the Striker Provider will emit Tasks for finding the ball, walking to the ball, looking at the ball, aligning to the goal, and kicking the ball. The finding the ball Task will run at the lowest priority, so if the walk to ball and look at ball are not running as there are no balls, then find the ball will kick in. Find the ball moves the head in a search pattern and turns on the spot. The kick planner will not kick the ball unless the ball is in front of the robot and the robot is facing the goals.

The Defender Provider will emit a task to patrol a search area. If the ball enters the search area, the robot will approach the ball and kick it away from the goal.

The Goalie Provider will dive in the appropriate direction when a ball is close to the robot. Before the 2024 competition this will be improved to also kick the ball away from the goal.

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

Yes, our robot has behaviour to look at objects of interest (eg ball, goal) and "look around" to find objects.

Do you apply some form of filtering on the detected objects (e. g. Kalman filter for ball position)?

All detected objects are filtered.
- Field: Particle filter
- Ball: Unscented Kalman Filter for position and velocity

Is your team performing team communication? Are you using the standard RoboCup Humanoid League protocol? If not, why (e.g. it is missing something you need)?
Yes, we are performing team communication using the standard RoboCup Humanoid League protocol.

Please list contributions your team has made to RoboCup
The NUbots team participated in the 2021 Humanoid Kid-Size League and finished as semi-finalists. The NUbots have participated in the Four Legged League (2002-2007), the Standard Platform League (2008-2011), the Kid-Size Humanoid League (2012-2017, 2022-2023), and the Teen-Size Humanoid league (2018-2019). NUbots were the Four Legged League world champions in 2006. The team won the first Standard Platform League in 2008 as team NUManoid in collaboration with the National University of Maynooth, Ireland. The team's RoboCup robot code, hardware, and debugging tools are open source on GitHub. The NUbots team have developed a Blender plugin to generate semi-synthetic images with fully-annotated ground truth segmentation maps. The images contain random ball positions, robot positions and kinematic poses, obstacles, and viewer orientations. This tool is public on GitHub for anyone in the League to use. The NUbots team maintains a comprehensive documentation resource in the form of a public website, providing detailed information about the hardware and software systems, as well as guides on various aspects of our systems. This resource aims to be useful to other RoboCup teams, as well as the wider robotics and AI community.

Please list the scientific publications your team has made since the last application to RoboCup (or if not applicable in the last 2 years).
Exploring GPT-4 for Robotic Agent Strategy with Real-Time State Feedback and a Reactive Behaviour Framework, Thomas O'Brien and Ysobel Sims, 2023 Stereo Visual Mesh for Generating Sparse Semantic Maps at High Frame Rates, Alexander Biddulph, 2023 The Director: A Composable Behaviour System with Soft Transitions, Ysobel Sims and Trent Houliston, 2023

Please list the approaches, hardware designs, or code your team is using which were developed by other teams.
Hardware platform is heavily based on the igus Humanoid Open Platform from University of Bonn and igus. Knee spring design is based on Bit-bots CAD model

What operating system is running on your robot and which middleware are you using (for example Ubuntu 22.04 and ROS2 Galactic)?
Operating system: Customized Arch image Middleware: NUclear https://github.com/Fastcode/NUClear

Is there anything else you would like to share that did not fit to the previous questions?