The NUbots Software Overview for RoboCup 2023

Joe Bailey, Darcy Byrne, Clayton Carlon, Stephan Chalup, Lachlan Court, Liam Craft, Jason Disher, Joel Ferguson, Luke Haigh, Utkrisht Jain, Sam McFarlane, Alexandre Mendes, Cameron Murtagh, Alana Noonan, Thomas O'Brien, Jesse Perrin, Ysobel Sims, and Jesse Williamson

Newcastle Robotics Laboratory
College of Engineering, Science and Environment
The University of Newcastle, Callaghan 2308, Australia
Contact: nubots@newcastle.edu.au
Homepage: http://robots.newcastle.edu.au

1 Walking

NUbots currently use Bit-Bots' Quintic Walk [3] based off Rhoban's Quintic Walk [7] and Rhoban's IK Walk [7]. This is an open loop walk engine that creates quintic splines representing the three-dimensional trajectory of the feet and torso both in rotation and translation. 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 an approximate analytical solution for the inverse kinematics of the NUgus robot based on geometry.

Updates to the walk engine computations in late 2022 have removed the lean in the walk on the NUgus robots and resulted in a better walk overall. A new walk tune results in a faster walk that responds well to changes in the velocity command, with better mobility which will enable behaviour improvements.

A current mechatronics final year undergraduate project has shown success with a new quasi-static walk engine and a dynamic zero moment point based walk engine in simulation. Another project involves developing a modular walk engine system, initially comprising of dynamic motion planners and a static walk engine.

2 Vision

The Visual Mesh [4] 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 [4]. 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 [4]. 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:

1. The camera lens type, height and orientation with respect to the ground are known.
2. The object is spherical with 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 travelling 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 neighbours and these connections become the edges of the mesh. A fully convolutional neural network [8] is used on the mesh where each sample point and it's neighbours become the input to a convolution. All layers in the network use seven points convolutions since there are six neighbours 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 is 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 specialised 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, an upper convex hull algorithm is applied to the final cluster determine 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 neighbour 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 centre 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 to 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 centre point of the goal post is then found by averaging the edge points. The distance to the goal posts are 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 localisation 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 behaviour system for obstacle avoidance. Implementation is schedule in the NUbots roadmap in time for RoboCup 2023.

3 Localisation

The localisation on-board the robot is performed using a Particle Filter. This allows us to maintain multiple hypotheses about the current pose of the robot, providing robustness against the mirrored field problem and having multiple potential initial positions when entering the field. The filter relies on measurement updates from the vision module, and on IMU data for time updates. The measurement update previously only tracked the four goal post locations; however, the team is working on a grid fitting method using field lines to enhance the robot's ability to localise.

4 Behaviour

The robot's behaviour is a basic state machine where, during the playing state, the robot will look for the ball and goals using head movements and the vision

system, position itself to kick the ball toward the goals using a simple path planning algorithm and the walk engine, and then kick when it determines it is in the right position to kick. The vision system alongside localisation determines which foot the robot will kick with. The robot will only move its head during the initial state and look for features on the field and localise itself. In the ready state it will walk to a specific position on the field. In the set state, it will stop moving.

The team is working on implementing a new behaviour framework called the Director. Within the Director, there are tasks and providers. Examples of tasks are '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 take a particular task and execute code that will perform that task. For example, the walk engine will be a 'walk' provider that contains a velocity command. The system aims to be highly modular, with small providers that provide the functionality for specific tasks such as 'look at ball' or 'walk to ball'. A group of providers provide the functionality for the same task, e.g. there may be providers that implement a 'stable static walk' and a 'fast dynamic walk' which both provide for the task 'walk', but in different contexts. Only one provider in a group can have access to the task and run.

Providers will often emit tasks of their own. In some cases, if they cannot emit these tasks then there is little point in running. For example, the walk engine needs access to the servos in all limbs. Another provider for 'getting up' may also want access to all servos in the limbs. These will have different priorities in the system. The Director algorithm will create trees of potential solutions and determine which solution should run.

Some providers may need the system to be in a particular state to run. For example, the 'kick' provider may require that the robot is in a standing position before running. This is specified in the kick provider, and it will not be a valid solution for the Director algorithm unless that state is satisfied. Providers can also declare that when they run, the system will achieve a particular state. A special 'walk' provider may cause the robot to end in a standing position. This provider would not usually run, as it does not perform the usual activity of the 'walk' task. However, if the 'kick' has higher priority than the 'walk', it will force the 'walk' task to be done by the special walk provider that guarantees a standing state, so that the kick will be a valid solution to the Director algorithm. This aims to improve transitions, so that before a new task runs it ensures it is in the right state.

With the transition of the codebase to the Director framework in 2023, the behaviour system will undergo an overhaul. This will include the additions of specific play styles for striker, defender and goalie with better decisions.

It is intended that by the 2023 competition, the robots will be able to identify other robots on the field. If this is achieved, the behaviour overhaul will include the addition of robot avoidance. In addition, robot-to-robot communication using the official RoboCup communication protocol will be implemented.

5 Contributions 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), 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 [5], hardware [9], and debugging tools [2] 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 [1]. 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 [6], 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.

References

1. M. Amos and A. Biddulph. NUbots PBR pipeline repository. `https://github.com/nubots/NUpbr`.
2. B. Annable, T. Houliston, M. Olejniczak, J. Paye, A. Biddulph, and L. Court. NUsight2 real-time web-based debugging utility code repository. `https://github.com/NUbots/NUbots/tree/main/nusight2`.
3. Team BitBots. Bitbot's quintic walk. `https://github.com/bit-bots/bitbots\_motion/`.
4. Trent Houliston and Stephan K. Chalup. Visual mesh: Real-time object detection using constant sample density. *CoRR*, abs/1807.08405, 2018.
5. M. Metcalfe, J. Fountain, A. Sugo, T. Houliston, A. Biddulph, A. Dabson, T. Johnson, J. Johnson, B. Annable, S. Nicklin, S. Fenn, D. Budden, J. Walker, J. Reitveld, Y. Sims, T. Young, M. Amos, D. Ginn, K. Hamiltons, J. Paye II, C. Murtagh, Y. de Koeyer, L. Court, L. Craft, P. Carlyle, A. Hall, T. O'Brien, and S. McFarlane. NUbots robocup code repository. `https://github.com/nubots/NUbots`.
6. J. Paye and Y. Sims. NUbots handbook. `https://github.com/NUbots/NUbook`.
7. Team Rhoban. Rhoban's quintic walk. `https://github.com/Rhoban/model/`.
8. E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, April 2017.
9. T. Young and A. Biddulph. NUbots hardware repository. `https://github.com/nubots/NUware`.