RoboCup Standard Platform League - rUNSWift 2010

Jayen Ashar, David Claridge, Brad Hall, Bernhard Hengst, Hung Nguyen, Maurice Pagnucco, Adrian Ratter, Stuart Robinson, Claude Sammut, Benjamin Vance, Brock White, Yanjin Zhu
University of New South Wales, Australia
{jayen,davidc,bradh,bernhardh,dhung,morri,adrianr, stuartr,claude,bvance,brockw,yanjinz}@cse.unsw.edu.au

Abstract

Multi-agent robotic competitions such as RoboCup provide the motivation for a developmental research agenda – one that focuses on the evolution of complete working systems and their cognitive architectures. In this paper, we describe the components and integration of one such system – the 2010 RoboCup Standard Platform League entry *rUNSWift*. The real-time control architecture employed consists of a hierarchy of modules that implement functions of perception, world-modelling and behaviour generation. The system was successfully deployed in both soccer and challenge competitions.

1 INTRODUCTION

At the time of the 50th anniversary of the 1956 Dartmouth summer research project on artificial intelligence (AI), concern was raised by several researchers that AI's shared vision of creating intelligent machines may be threatened by the fragmentation of the field into a myriad of speciality areas with little communication between them [Cohen, 2005; Langley, 2006; Brachman, 2006]. In response they called for what Cohen termed a *developmental* research strategy, one that demands complete, integrated systems to solve whole problems.

Robotic competitions can provide challenging problems that foster a developmental research agenda and motivate re-integration of AI. RoboCup™ is such an international research and education initiative providing standard problems where technologies can be integrated and examined[1].

In this paper, rather than focus on one component in great detail, we describe a complete and integrated robotic project – that of the rUNSWift multi-agent system comprising three bipedal Nao robots[2] competing in the 2010 RoboCup Standard Platform League (SPL) in soccer (Figure 1). Our aim is to provide an overview of all the key components of the system and their interrelationship through the framework of a cognitive robotic agent architecture.

Figure 1: rUNSWift team of Nao robots playing soccer in the 2010 RoboCup SPL league in Singapore.

It would not have been possible for a new multi-person team to undertake such a complex project in the short time available without instituting adequate project management systems and practices. We employed a suite of products[3] to facilitate collaboration and knowledge management, track issues, and keep tabs on the team members and the state of repair of our robots. Further elaboration on project management is outside the scope of this paper.

Innovative developments ranged across all components and systems, and while incremental, it is a case of the whole being more that the sum of its parts. Consistent with a developmental research approach that starts with a poor, but complete working system, innovation followed the *Kaizen* practice of continually improving all functions by involving team members, to ratchet up the

[1] RoboCup http://www.robocup.org/

[2] Aldebaran http://www.aldebaran-robotics.com/

[3] from Atlassian Pty Ltd http://www.atlassian.com/

competence of the system. The project involved the total rewrite of the real-time robot control system by largely undergraduate students supported by research students and faculty staff over a period of 8 months.

In the rest of the paper we start by discussing intelligent agent architectures and the specific architecture employed by rUNSWift. This is followed by a systems level description of the main components of the system: perception, the world-model, and behaviour generation. Finally we describe the real-time monitoring tools designed to support the development effort and comment on results achieved with the final system tested under conditions of international competition.

2 ROBOTIC ARCHITECTURE

The literature abounds with cognitive agent architectures. Examples include: ICARUS, a cognitive architecture that incorporates ideas including work on production systems, hierarchical task networks, and logic programming [Langley *et al.*, 1991]; SOAR, a symbolic cognitive architecture based on a production system [Lehman *et al.*, 1996]. A key element of SOAR is a *chunking* mechanism that transforms a course of action into a new rule; BDI (Belief-Desire-Intention), a software model for programming intelligent agents that has led to several agent implementations [Rao and Georgeff, 1995]; and Adaptive Control of Thought – Rational (ACT-R), a cognitive architecture that aims to define the basic and irreducible cognitive and perceptual operations that enable the human mind [Anderson, 2005]. A recent survey [Langley *et al.*, 2009] explores the motivations for research on cognitive architectures and reviews several other architectures that have been explored in the literature.

While our robotic architecture was independently developed, it is similar to Albus's Real-time Control System (RCS) [Albus, 1991], a multi-layer control system composed of a hierarchy of modules whose functional elements include sensor processing, world-modelling, behaviour generation and value judgement.

2.1 rUNSWift Robotic Architecture

The rUNSWift robotic architecture is best envisaged as a *task-hierarchy* [Dietterich, 1998] that consists of a set of finite state machines [Hartmanis and Stearns, 1966] linked in a lattice. Tasks are modules of the architecture that accept input, perform a computation that can change the state of the module and produce output as a function of the input and state. The output may be input to another module or invoke an action.

If an action invokes and executes another module it is called an *abstract* or *temporally extended* action. An action that has a direct effect on the agent's environment it is called a *primitive* action. We call actions *multi-tasking* when they are required to complete multiple subtasks, but can do so only one at a time [Hengst, 2008]. They are *concurrent* when they execute multiple tasks at the same time. When subtasks terminate, they return control to their parent task. Generally the states of the system represent more abstract concepts as we ascend the control hierarchy and the reaction time decreases.

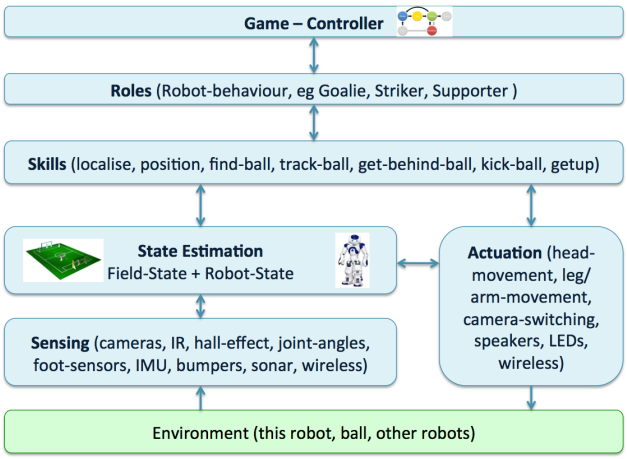

Figure 2: The 2010 rUNSWift robotic architecture.

Task-hierarchies have been formalised in approaches to hierarchical reinforcement learning and open the door to learning behaviour at multiple levels [Andre, 2003; Ferrein *et al.*, 2003].

The rUNSWift robotic architecture is shown in outline in Figure 2. As the team of three Naos have no central controller, control is distributed by implementing the architecture on each robot. This restriction is imposed by the event organisers and has the advantage of providing some redundancy in case individual robots are disqualified or stop working. The robots may have slightly different beliefs about their world formed from partially observable and noisy inputs.

At the root-level, the game-controller changes soccer play states (Initial, Ready, Set, Playing, Penalized, Finished) and player robots determine their roles (Striker or Supporter) by sharing world-models. At these higher levels the control response is measured in seconds or even minutes.

At lower levels, the walk-generators execute temporally extended walk-phases that invoke primitive state changes in joint angles at 100 Hz. The omni-directional walk and kick generators are themselves task-hierarchies. In some cases the the rUNSWift architecture may execute through as many as nine-levels of task-hierarchy.

3 PERCEPTION

3.1 Vision

Identification of objects in a video feed is a significant research area in robotics and forms the major component of the robot's sensory perception of the world. While the structured area of a soccer field permits the use of algorithms tailored for the identification of specific objects, such as the orange ball used in the competition, a game of soccer played using the Nao robots presents specific and complex challenges in the field of computer vision, including:

- Vision has to run fast enough to provide up-to-date information to other components. 640×480 pixel video frames should be processed at 30 frames per second on the Nao's 500MHz processor and still leave capacity for all other functions.
- Objects must be identified accurately enough to allow kinematics to provide reasonable estimates of their distance from the robot.
- The vision system must be robust enough to perform with a high level of image blurring due to the bipedal locomotion, varying lighting conditions over the field and a significant amount of object occlusion from referees and other robots.

Our new approach to object identification combines cues from both colour classification and edge detection. Colour classification allows fast identification of approximate areas of objects, while edge detection allows the positions of these objects to be found very accurately in the image, and provides a substantial amount of robustness to uneven and changing lighting conditions.

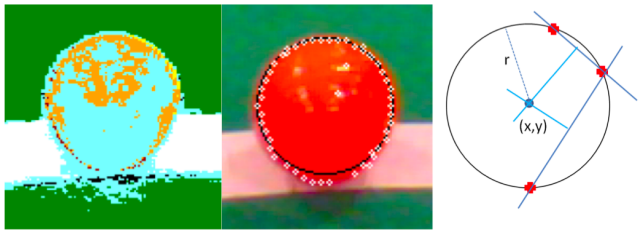

Figure 3: Using the combined cues of colour and edge features, the orange ball is able to be located and segmented out to accurately determine its image location and radius. This information together with the pose of the robot and camera allows the robot relative ball distance and direction to be determined. Left – Colour classified image used for segmentation. Centre – Ball edge points overlaid on the original image. Right – Geometry used to find candidate circles. We take the median of several random samples.

We start with a *saliency-scan* by subsampling each visual frame to produce a much smaller 160×120 pixel colour classified image to efficiently identify the probable locations of the various objects. By focussing attention on high probability locations within the full 640×480 resolution, we effectively implement a virtual saccade in software to accurately determine the location of each object in the frame. Figure 3 shows colour and edge cues used to identify a ball.

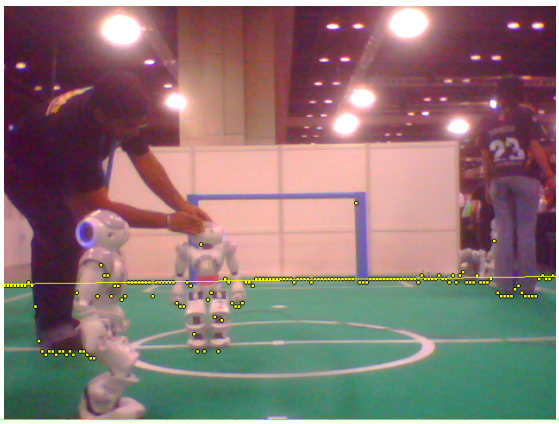

Figure 4: Edge detection using RANSAC.

The saliency-scan was also used to identify field-edges using a *random sample consensus* RANSAC algorithm [Fischler and Bolles, 1981] as shown in Figure 4. The horizon is calculated from the forward kinematic chain from the foot to the camera. The field-edges and the horizon allow us to reason about the objects in the environment. For example, soccer balls cannot appear above the field-edge and the recognition system rules these hypotheses out.

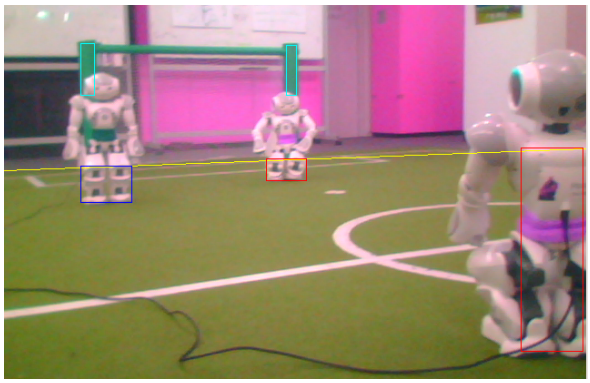

Figure 5: Detection of goal-posts, field-edge, and robots.

The part of the image that contains the field is scanned to identify and grow regions [Röfer *et al.*, 2009] that could possibly contain balls, robots, or field-lines. The shapes in these regions and colours of the pixels is used to identify the most probable objects they contain, and specific algorithms for ball, field-line, and robot detec-

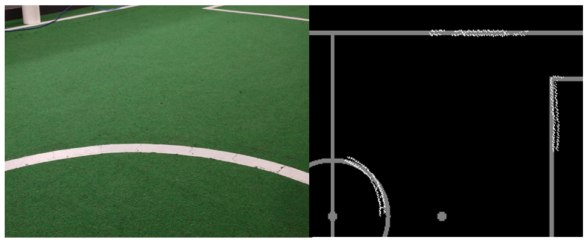

Figure 6: Image showing field-lines and corresponding matches.

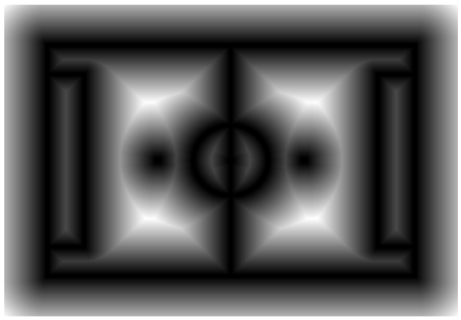

Figure 7: Shortest distance of all locations to nearest field-edge point.

tion can be used on the appropriate regions to reliably and accurately identify the objects. Figure 5 shows the detected field-edge, two goal-posts, and three robots.

Figure 6 shows field-line detection matching field-edge points locally using a distance metric from a precomputed nearest-line distance map of the field shown in Figure 7, along similar lines to [Sheh and Hengst, 2004].

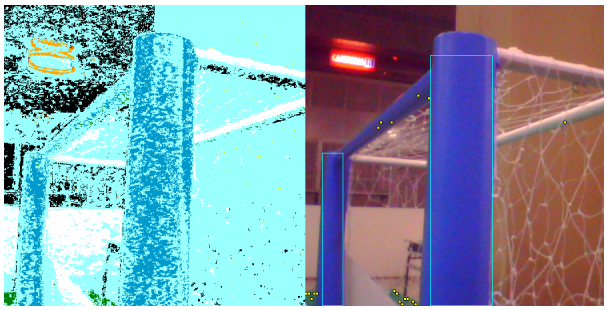

Figure 8: Accurate detection of goal-posts despite considerable noise in the colour-classified image.

Using histogram information generated with the saliency-scan, goal detection is performed separately as the goals are mostly above the field edge. Figure 8 is an example of object detection where the lighting conditions have considerably degraded the goal-post colour as previously classified. Nevertheless, this information is sufficient to help demarcate the goal-post edges.

Once the position of an object in the image has been identified, we use a kinematics chain to estimate the distance and heading the object is away from the robot. In some circumstances where kinematics cannot provide an accurate distance estimation, other methods are used, such as the width of goal-posts and the radius of the ball, to calculate the distance.

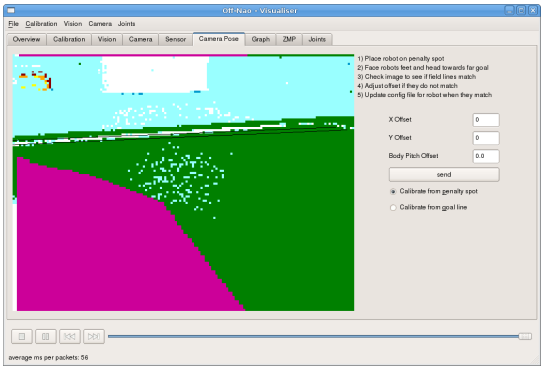

Figure 9: The pink convex shaped area at the bottom left of the image is excluded from processing due to the robot's own shoulder.

To ensure that vision does not detect objects such as field-lines or balls within the robot's image of its own body, forward-kinematics is used to determine where parts of the body would appear in the image, and these areas are excluded from vision processing as shown in Figure 9.

3.2 Sonar

Left and right sonar sensors are available on the Nao. We augmented the robot's belief about the location of other robots with appropriately filtered sonar observations. The filter retained the history of the last 10 sonar values. At each iteration it looped over these 10 values and if 8 of them were below a threshold an object was detected occasioning avoidance behaviour.

4 WORLD-MODEL

Sensors only provide partial and noisy information about the environment. Robots need a component that allows them to integrate observations over time and filter out noise to build up a relevant and complete picture of their surroundings and themselves. This process is also called *situation awareness*, a term rooted in military history and now more widely adopted. In control engineering, statistics and signal processing we refer to this component as *state-estimation*, usually by some form of Bayesian filtering.

As shown in Figure 2, the rUNSWift architecture divides the environmental state into two sets of variables,

field-state and robot-state. Field-state defines the location and orientation of all the robots and the ball required to reason about actions at a more strategic level. Robot-state includes the pose and dynamics of the robot, and whether it can see the ball. It is required to predict the field-of-view of the camera and the closed-loop control of bipedal walking and kicking. The total state is kept on a *Blackboard* (shared memory).

Each robot's world-model is broadcast at 5 Hz to each of its team-mates, informing them of its belief about its position and that of the ball. This information is incorporated in the world-model of each robot.

4.1 Bayesian Filtering

For efficient state-estimation, we chose to use a combination of particle (PF) and Kalman filters (KF) [Thrun *et al.*, 2005]. The particle filter is used without resampling to quickly integrate temporal information. It terminates when it narrows the robot position down to one likely hypothesis. Its invocation is usually required when the robot is placed anew on the field – solving the so-called "kidnapped robot" problem. The uni-modal Kalman filter uses two types of observation updates. When observations uniquely measure the position of the robot we use standard KF updates with time-varying gains. When the observations are ambiguous and are consistent with multiple hypotheses, we update based on the hypothesis that is closest to the current estimated position of the robot. The dissonance between observations and current belief is indicated by a high innovation variance and determines when to switch in the PF again.

The robot-relative ball position is independently filtered using a KF, however we have previously used a multi-modal KF to track the combined state of all our robots and the ball [Sushkov, 2006].

The robot state is determined from unfiltered joint-sensors, constant gain filtered accelerometers and foot-sensors. The field-state and robot-state is fed back to vision to provide context for interpreting the scene.

4.2 Position Observation Geometry

Observations from vision can provide redundant noisy information, for example, the distance to two goal-posts plus the subtended angle between them. While in principle we can filter this information with appropriate sensor models, for efficiency reasons we use closed form geometric calculations. The accuracy of determining the position can depend on the estimated location on the field. Figure 10 and 11 show two ways to determine the position of the robot given the observation of two goal-posts. The orange circle shows equi-subtended angle positions. The yellow circle shows equidistant positions to the nearest goal. The yellow dot is the reported robot position. Further down-field, the method in Figure 10 leads to multiple hypotheses (blue dots) when measurements are noisy. Instead we use the distance to both goal-post and project the point (intersection of yellow circles) from the goal-centre to be consistent with the subtended angle.

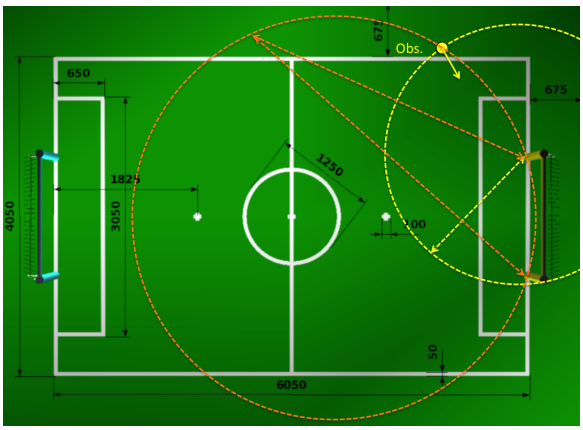

Figure 10: In corner field positions the distance to the nearest goal-post and the subtended angle between the posts is more accurate.

Figure 12 shows the geometric reasoning employed when the robot sees a field-edge and a single goal-post. The distances to these objects alone will generate up to 12 position hypotheses, 6 of which are on one side of the goal-posts. We test consistency with the direction of both the field-line and the goal-posts, and only accept an observation when we have exactly one match.

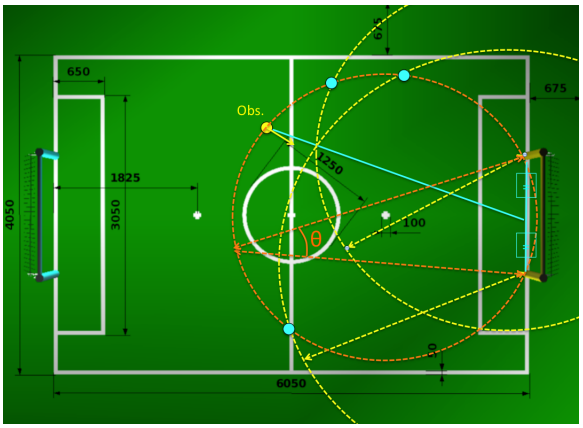

Figure 11: Robot position observation when further down-field.

The observations used to localise the robot include goal-post distances and directions, field-edge inclinations and distances, and local field-line matching. One feature of the system is the close integration of vision and field-state estimation, each providing context to the other with innovation variance breaking the loop when the sys-

tem traps itself in a discordant state.

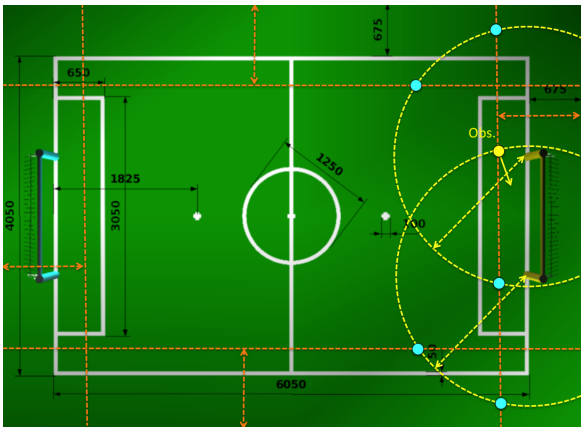

Figure 12: Multiple position hypotheses are generated from distance observations to a goal-post and a field-line (blue dots). They may be narrowed down to one (yellow dot) when directional data is consistent.

5 BEHAVIOUR GENERATION

The task-hierarchy in Figure 2 starts from the soccer game-controller that manages the two 10 minute game halves and cascades down to actions setting joint angles at 100Hz. The behaviour generation along this path is best divided into robot roles and skills implemented in Python (described in the appendix), and actuation or motion generation that executes omni-directional locomotion and kicking.

5.1 Motion Generation

The Nao is a flat-footed bipedal robot required to play a competitive game of soccer. We therefore need to concern ourselves with the opposing requirements of speed and agility for this 21 DoF machine – rapid stop/start, acceleration/deceleration, turns, and accurate kicks. Other than special canned routines for getting up, the main motion generators are based on bipedal locomotion routines. We used a combination of three gaits; one supplied by the manufacturer, and two of our own *SlowWalk* and *FastWalk*.

SlowWalk has the desirable property that the robot's weight can be shifted totally to either leg and in this statically balanced state the other leg is free to move to execute various omni-directional kicks; forward, 45deg to either side of forward, sideways and backward.

FastWalk is a dynamic gait designed for speed with a duty factor[4] just over 50%. The walk is designed by treating the sagittal (forward) and coronal (sideways) dynamics as orthogonal and then synchronously recombining the two motions to produce bipedal locomotion.

The coronal rocking frequency (2.6Hz) is induced by lifting each leg alternately. It is tuned to the natural frequency of the robot and the cycle synchronised using the zero-crossing point of the filtered centre-of-pressure (CoP) measured across both feet. The dynamics of the system are best understood in terms of an inverted pendulum model. Figure 13 shows a diagram of the model, and the simulated and actual CoP and leg-lift motions.

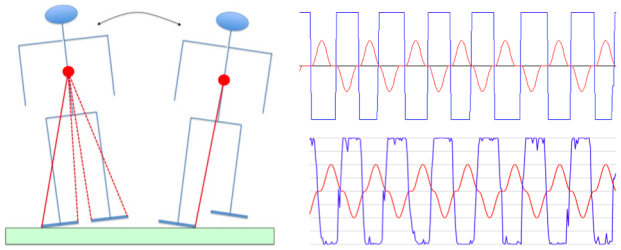

Figure 13: Inverted pendulum model for the FastWalk coronal rocking motion. The red dot is the centre-of-mass of the system. The pivot of the pendulum changes depending on which foot-edge makes contact with the ground as the robot rocks from side to side. The top time-series plot shows the centre-of-pressure over both feet (blue squarish wave) and the sinusoidal leg-lift for the simulated gait for each foot (+ve one foot, -ve the other). The bottom time-series shows the same plots measured on the actual Nao robot.

The sagittal motion keeps the upper body moving at constant speed determined by the three walk parameters – forward, sideways, turn. The legs alternate between stance and swing phases keeping the double support phase close to zero seconds. Foot placement is determined by the three walk parameters to achieve omni-directional locomotion. The swing foot stride is maximised by a constant acceleration of the foot to the halfway point, followed by a constant deceleration to a complete stop just before replacement. This is designed to minimise slippage with the other foot which we assume is governed by dry friction forces. Sagittal motion is stabilised with feedback from the sagittal centre-of-pressure by leaning the upper body forward or backward.

Figure 14 shows an example plot of various dynamic variables characterising the omni-directional bipedal gait as the robot accelerates from a standing start to full-pace and is directed to make a sharp turn to the left. The plot shows cycle-time, coronal and sagittal centre-of-pressure, left and right stride lengths, and the amount of turn. To avoid falling over, the robot first decelerates before executing the turn.

[4]Duty factor is the percent of the total cycle which a given foot is on the ground. Duty factors over 50% are considered a "walk", while those less than 50% are considered a run.

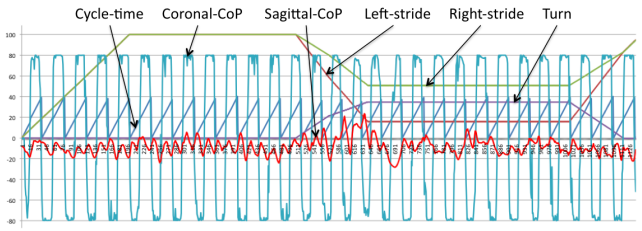

Figure 14: Various time-series data showing FastWalk accelerate from a standing start to a speed of 24cm/sec and then executing a hard turn. The stride length plots show automatic deceleration as the turn rate is increased. During this left turn the right leg stride length is longer than that of the left leg. The sagittal CoP shows some destabilisation during the onset of the turn, but is feedback corrected.

The algorithm driving the leg joint angles relies on inverse kinematic calculations in closed-form and on iterative methods [Buss, 2004] when turing due to the complexity of the hip-joint with Nao motors inclined at 45 degrees to the body.

5.2 Behaviour Roles and Skills

The top-level behaviour roles are *striker*, *supporter*, and *goalie*. Rules required the goalie to be a dedicated robot. The striker and supporter role is switched dynamically depending on which robot is closest to the ball. To reduce decision vacillation and conflict, hysteresis is used to threshold these behaviours.

The state of each role is determined by the robot and field states of the world-model. The behaviour modules function as finite state machines, with the state succinctly determined by a decision tree. The leaves of the tree are the states that invoke concurrent head and body skills. Skills include parameterised GotoPoint, FindBall, TrackBall, Localise, ApproachBall, and Kick routines.

6 OFF-BOARD MONITORING

Off-Nao is a desktop off-board monitor application to help debug the autonomous robotic system online and in real-time. The application can also separately run the vision module on a sequence of recorded frames, allowing for reproduction of errors and regression testing of vision enhancements. Off-Nao has several tabs to switch-in debugging modules.

The overview tab (Figure 15) displays the colour classified image overlaid with detected objects. It also displays a 2D view of the world-model of a robot including the belief of the location of all robots and the ball.

The colour calibration tab (Figure 16) is used to train the robots vision system with a nearest-neighbour classifier to recognise the primary colours in the environment.

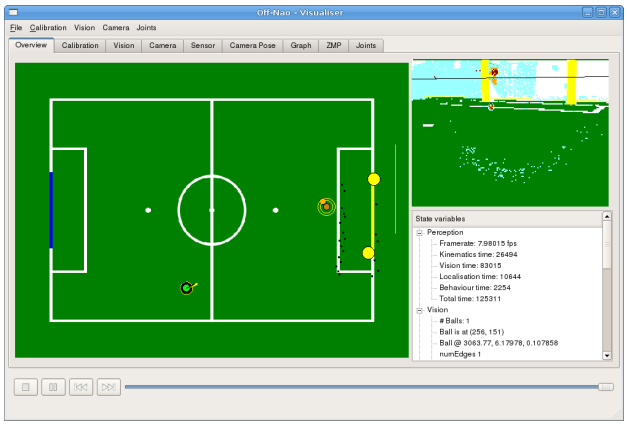

Figure 15: Off-Nao Overview Tab.

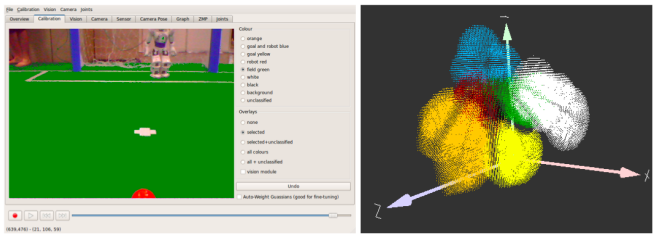

Figure 16: Colour calibration tab (right) and the resultant nearest neighbour classified YUV 3D colour-space (left).

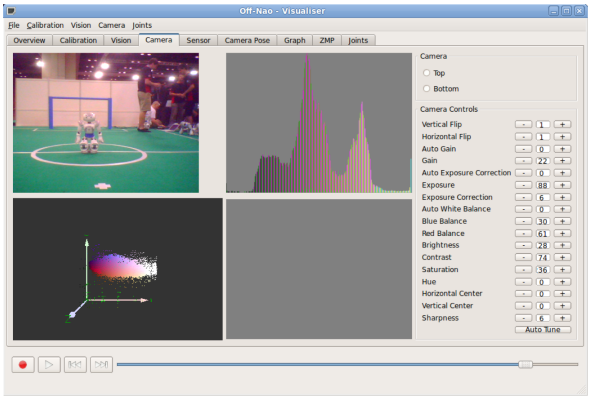

Figure 17: The Camera Calibration tab in Off-Nao.

The vision tab allows closer analysis of the object recognition system. The calibration tab adjusts parameters in the kinematic chain to offset the camera pose for each robot. Figure 18 shows an Off-Nao screen-shot calibrating camera offsets.

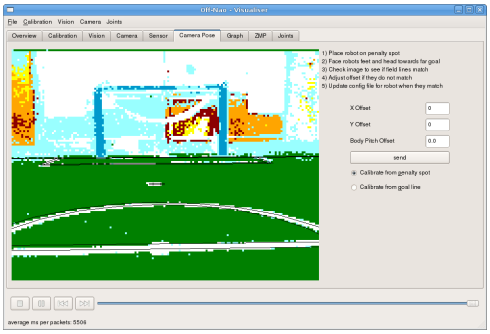

Figure 18: Off-Nao showing kinematic calibration of camera offsets.

A camera calibration tab is used to tune 16 camera parameters to achieve a good compromise between image clarity and colour separation given venue lighting conditions. These setting were determined manually this year, but we are experimenting with hill-climbing algorithms to automate this process using a maximum entropy fitness criterium.

The graph tab plots the time-series for accelerometers, foot-sensors, torso-tilt, sonar, battery charge and current.

7 CHALLENGE COMPETITION

In addition to the soccer competition, team innovation is stretched with three challenges. In 2010 these were ball passing, dribbling and a open challenge [RoboCup SPL Technical Committee, 2010].

Accurate ball passing and dribbling around obstacles requirers a more controlled kick. The specially developed *passing kick* had the right reliability characteristics. For the passing kick the robot lines up the ball with one foot, takes a forward step with the non-kicking foot placing it slightly in front of the ball, and then swings the kicking foot through at controlled speeds to kick the ball. The rational behind this action is to guide the direction of the kick and to control the amount of kinetic energy imparted to the ball. Over the desired shorter kick distances, the variance in both direction and distance was reduced from that of the more powerful forward impact kick.

For the open challenge we chose to find, pick-up, and throw-in a more regular black and white ball. The amount of non-ball black and white in the scene is significant and our standard orange-ball edge detector does not suffice. Instead we use a circular Hough transform on all transitions to green pixels. By processing only the subsampled saliency-image, reducing the dimensionality, range, and discretisation of the Hough transform, and caching trigonometric functions, the transform performed at an acceptable 25 frames per second. Figure 19 shows the edge pixels, the 2D Hough accumulator array, and the final ball overlaid on the original image.

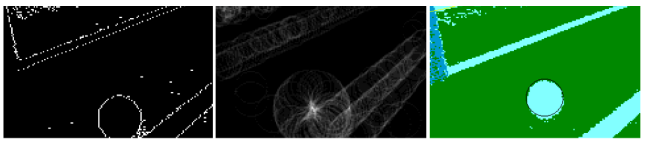

Figure 19: Black and white ball detection. Left – edge pixels. Centre – Grey scale 2D Hough accumulator array. Right – detected ball location.

8 RESULTS

One way to evaluate results for system developmental research is to test the final system in the environment for which it is designed, to see whether it achieves its objectives, and to compare it to competitors. The SPL league in RoboCup differs from other leagues in that the robotic hardware is identical for all competitors and may not be modified, with results reflecting only the suitability of robotic architectures and the performance of software components.

rUNSWift took out first place in the Challenge competition and was a finalist in the SPL soccer competition in Robocup 2010, Singapore. The results are shown in Figure 20. The top four teams in the 2010 competitions, out of a field of 24, are the University of New South Wales (rUNSWift), the University of Bremen (B-Human), the University of Texas at Austin (Austin Villa), and Carnegie Mellon University (CMurfs). A video is available on this website `http://www.cse.unsw.edu.au/~robocup`.

9 CONCLUSIONS

This paper presented some aspects of the design of rUNSWift's 2010 robotic software for the Nao, and the contributions we made to the state of the art. This year saw innovations in all major components: Vision, Localisation, Behaviour, and Motion; and importantly, we developed a robust and modular framework that can be used to facilitate research going forward. The full 150 page report is available from the University of New South Wales [Ratter *et al.*, 2010].

A key to rUNSWift's performance in the 2010 RoboCup Competition was the effective allocation of human resources, through making design trade-offs, such as implementing a less sophisticated localisation system that was relatively simple to develop and accurate

Challenges	Passing	Open	Dribbling	Sum
rUNSWift	24	22	23	**69**
Austin Villa	25	19	24	**68**
CMurfs	22	13	21	**56**
B-Human	23	24	0	**47**
Nao Team Humboldt	19	23	-	**42**
Austrian Kangaroos	0	16	25	**41**
UPennalizers	19	-	22	**41**
Robo Eireann	19	21	-	**40**

Quarter Finals	Q1	**B-Human**	UPennalizers	8:1
	Q2	**Austin Villa**	Nao-Team HTWK	3:1
	Q3	**rUNSWift**	Northern Bites	3:0
	Q4	NimbRo	**CMurfs**	1:2
Semi finals	S1	**B-Human**	Austin Villa	8:0
	S2	**rUNSWift**	CMurfs	6:0
3rd Place Final	3rd Place	**Austin Villa**	CMurfs	5:1
	Final	**B-Human**	rUNSWift	6:1

Figure 20: Results for the three challenges (top) and the soccer competition (bottom).

enough for our needs, or using a slower language for behaviour, that allows for rapid development. This allowed us to focus our efforts on the systems that needed major overhaul to be effective on the Nao platform, particularly Vision and Motion. All these systems have proved themselves worthy in the face of competition, and provide a strong platform for future teams to work from.

Of particular significance is the adoption of a Paul Cohen's developmental approach, that gradually improves the competence of the whole system over time. Instead ...

> If the organisers had followed the traditional divide and conquer strategy, then the first few annual competitions would have tested bits and pieces – vision, navigation, communication, and control – and we would probably still be waiting to see a complete robotic team play an entire game. – [Cohen, 2005].

Appendix – Systems Implementation

A major task to facilitate integration of the system was a total rewrite of software. We provide here a brief outline of the scope of the software engineering involved.

The Nao is provided with a fully-functional Linux operating system, as well as custom software. Standard practice is to program the robot by producing a dynamic library that is loaded when the robot boots to allow for interaction with third party code. The disadvantage of this approach is that it can create difficulty in debugging and recovery from crashes. To avoid damage to the robot, and to isolate potentially dangerous and complex code in the robot's control system from low-level hardware interaction, we have developed two separate binary packages that we deploy on the robot: *libagent* and *runswift*, which communicate using shared memory.

The primary purpose of libagent is to provide an abstraction layer over the Device Communications Manager (DCM) that has the simple task of reading sensors and writing actuation requests. Due to its simplicity, is is less likely to contain errors, and thus less likely to crash and cause the robot to fall in a harmful way. The DCM provides two functions, `atPreProcess` and `atPostProcess`, to register callback functions which are called before and after the 10ms DCM cycle, respectively. In the pre-DCM-cycle callback function, we read the desired joint and LED commands from the memory shared with 'runswift', and use the DCM's `setAlias` commands to actuate the robot. In the post-DCM-cycle callback function, all sensor values are read into shared memory.

The *runswift* executable is a multi-threaded process, featuring 6 threads: Perception, Motion, Off-Nao Transmitter, Nao Transmitter, Nao Receiver and GameController Receiver. All of these except for Perception are IO bound, providing reasonable multithreading performance on a uniprocessor, such as the AMD Geode found in the Nao.

Off-Nao is a desktop application written with the Qt4 widget toolkit, which streams data from the Nao using a TCP/IP connection established using the boost::asio C++ library. Recordings can be reviewed in Off-Nao to help debug vision processing, world-model state-estimation, and monitor other sensory variables. Figure 18 shows an Off-Nao screen-shot calibrating camera offsets.

To facilitate the rapid development of behaviours, we chose to use a dynamic language, Python. The Python interpreter was embedded into the *runswift* C++ executable using libpython. The inotify library for Linux was used to monitor a directory on the robot containing Python code, and reload the interpreter whenever Python code changes. This architecture allows us to make small modifications to behaviour and upload them to the robot whilst runswift is still running. The robot's motion thread is uninterrupted, so it will continue walking, standing, or kicking as per the action request on the Blackboard. When the replacement Python code is uploaded, the Perception thread continues with the behaviours reset.

The *boost::asio* C++ library is used to broadcast information from each robot at 5 Hz to each of its teammates, informing them of its position and the position of the ball, this information is incorporated at the Localisation level.

10 ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of the ARC Centre of Excellence for Autonomous Systems and contributors: Oleg Sushkov, Will Uther, Carl Chatfield, Nathan Kirchner, Jannik Jakobson, Jesper Sorensen, Sowmya Arcot, Ammar Alanazi, Franco Caramia, Tenindra Abeywickrama, Brenda Ford. Aldebaran Robotics supplied and repaired the robots. Atlassian Pty Ltd provided the JIRA suite for project management.

References

[Albus, 1991] James Albus. Outline for a theory of intelligence. In *IEEE Transactions on Systems, Man and Cybernetics*, 1991.

[Anderson, 2005] John R. Anderson. Human symbol manipulation within an integrated cognitive architecture. *Cognitive Science*, 29:313–341, 2005.

[Andre, 2003] David Andre. *Programmable reinforcement learning agents*. PhD thesis, 2003. Chair-Russell, Stuart.

[Brachman, 2006] Ronald J Brachman. More than the sum of its parts. *AI Magzine*, 27(4):19–34, Winter 2006.

[Buss, 2004] Samuel R. Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. Technical report, IEEE Journal of Robotics and Automation, 2004.

[Cohen, 2005] Paul R Cohen. If not Turings's test, then what? *AI Magzine*, 26(4):61–67, Winter 2005.

[Dietterich, 1998] Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 1998.

[Ferrein *et al.*, 2003] Alexander Ferrein, Christian Fritz, and Gerhard Lakemeyer. Extending dtgolog with options. In Georg Gottlob and Toby Walsh, editors, *IJCAI*, pages 1394–1395. Morgan Kaufmann, 2003.

[Fischler and Bolles, 1981] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[Hartmanis and Stearns, 1966] J. Hartmanis and R.E. Stearns. *Algegraic Structure Theory of Sequential Machines*. Prentic-Hall, 1966.

[Hengst, 2008] Bernhard Hengst. Partial order hierarchical reinforcement learning. In *Australasian Conference on Artificial Intelligence*, pages 138–149, 2008.

[Langley *et al.*, 1991] Pat Langley, Kathleen B. McKusick, John A. Allen, Wayne F. Iba, and Kevin Thompson. A design for the icarus architecture. *SIGART Bull.*, 2(4):104–109, 1991.

[Langley *et al.*, 2009] Pat Langley, John E. Laird, and Seth Rogers. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, 10(2):141–160, June 2009.

[Langley, 2006] Pat Langley. Cognitive architectures and general intelligent systems. *AI Magzine Magazine*, 27(2):33–44, Summer 2006.

[Lehman *et al.*, 1996] Jill Fain Lehman, John Laird, and Paul Rosenbloom. A gentle introduction to soar, an architecture for human cognition. In *In S. Sternberg and D. Scarborough (Eds), Invitation to Cognitive Science*. MIT Press, 1996.

[Rao and Georgeff, 1995] Anand S. Rao and Michael P. Georgeff. Bdi agents: From theory to practice. *Proceedings of the first international conference on multiagent systems ICMAS95*, 1995.

[Ratter *et al.*, 2010] Adrian Ratter, Bernhard Hengst, Brad Hall, Brock White, Benjamin Vance, David Claridge, Hung Nguyen, Jayen Ashar, Stuart Robinson, and Yanjin Zhu. runswift team report 2010 robocup standard platform league. Technical report, School of Computer Science and Engineering, University of New South Wales, 2010.

[RoboCup SPL Technical Committee, 2010] RoboCup SPL Technical Committee. Technical challenges for the robocup 2010 standard platform league competition. Website: http://www.tzi.de/spl/bin/view/Website/WebHome, 2010.

[Röfer *et al.*, 2009] Thomas Röfer, Tim Laue, Judith Müller, Oliver Bösche, Armin Burchardt, Erik Damrose, Katharina Gillmann, Colin Graf, Thijs Je ry de Haas, Alexander Härtl, Andrik Rieskamp, André Schreck, Ingo Sieverdingbeck, and Jan-Hendrik Worch. B-human team report and code release 2009. http://www.b-human.de/index.php?s=publications, 2009.

[Sheh and Hengst, 2004] Raymond Sheh and Bernhard Hengst. A fast vision sensor model: Matching edges with nightowl. In *Australian Conference on Robotics and Automation*, 2004. rUNSWift 2004.

[Sushkov, 2006] Oleg Sushkov. *Robot Localisation Using a Distributed Multi-Modal Kalman Filter, and Friends*. Honours thesis, The University of New South Wales, 2006.

[Thrun *et al.*, 2005] S Thrun, W Burgard, and D Fox. *Probabilistic Robotics*. MIT Press, 2005.