The 2004 NUbots Team Report

Michael J. Quinlan, Craig L. Murch, Timothy G. Moore,
Richard H. Middleton, Lee Andy Yung Li,
Robert King, and Stephan K. Chalup

School of Electrical Engineering & Computer Science
The University of Newcastle, Callaghan 2308, Australia
{mquinlan,rick,chalup}@eecs.newcastle.edu.au
WWW home page: http://robots.newcastle.edu.au

1 Introduction

At RoboCup 2004 in Lisbon the NUbots successfully defended their 3rd placing for the second year running, having first entered the competition in 2002. Once again, the NUbots only lost one game in the entire tournament: this time to the German Team who went on to win the competition.

The introduction of the Sony AIBO ERS-7 robot led to considerable attention being paid to the porting of code. As a result we were unable to spend as much time implementing new ideas and approaches. The NUbots also had a smaller development team than in previous years which further hampered the design and implementation of new methods. It should also be noted that the NUbots are intending to perform complete rewrite for 2005. For this reason, this team report will focus on the new features of the system and the changes made to accommodate the new robot. Further information on our current/past system can be found in our 2003 [1] and 2002 [2] reports.

2 The System Architecture

The high-level architecture of the NUbots has not changed from 2003. In essence the control system conforms to the classical sense-think-act-cycle architecture which processes sensory data into information that can be used by the actuators. The main relationships between the modules are illustrated in Figure 1.

Each module and the interactions between them will be discussed in their individual sections as follows. As with many teams, *vision processing* is a large portion of the software, which we discuss in Section 3. We follow this by a description of new features in *localisation and world modelling* in Section 4. Then we describe the *locomotion* and *behaviour* modules in sections 5 and 6 respectively.

2.1 Inter-robot communication

2004 introduced the possibility of using UDP for robot-to-robot communication. This is a considerable change, since previous years had required the use of a

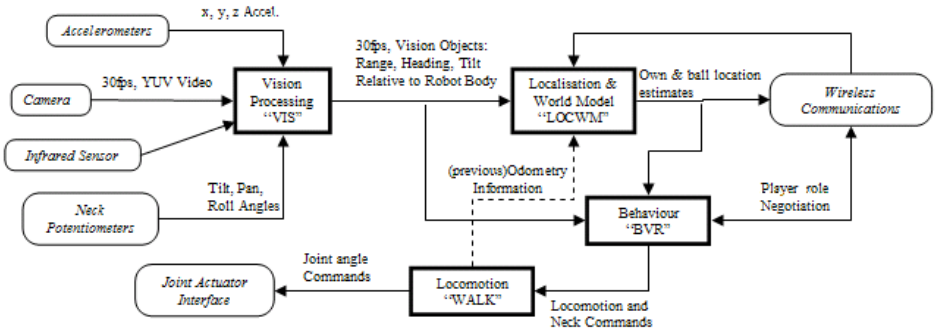

Fig. 1. Main Software Structure Block Diagram

host PC for relaying messages between the robots. We note that the NUbots wrote code to perform UDP communications in 2003, but were hampered by an apparent bug in the system software on the robots which under certain conditions would cause packet sending to stop after sometime. New versions of OPEN-R and rewritten UDP code solved this problem.

UDP brings with it several benefits. The latency for inter-robot communications is much reduced and bandwidth usage can be reduced considerably by using UDP broadcast packets. Using UDP also reduces the number of layers involved in communication, meaning it is easier to debug problems.

The use of UDP was extremely beneficial in testing team behaviour. For example, ad-hoc tests involving two strikers became much more convenient and so were performed much more frequently. Additionally, the use of UDP eliminated possible problems with the host PCs provided at the RoboCup competition. There is one caveat, however: Occasionally, a UDP endpoint stops working and so team communication ceases. This seemed to occur more often with an unstable wireless network, and did happen several times at the 2004 competition. The NUbots' system is able to recreate UDP endpoints when such a problem is encountered, maintaining communications even on an unstable wireless network.

3 Vision System

3.1 Robot recognition

For this year's competition, the *robot recognition* code was completely rewritten to make it more reliable under game conditions and to deal better with the new robot. The new version consists of two parts, the robot noise filter and robot recognition proper.

The robot noise filter's purpose is to eliminate 'false positives', in particular, red objects that are not other robots. First, we check each object to ensure the lowest point in the image is below the horizon line. We then remove any objects

smaller that 4 pixels across or high. Additionally, if the entire object is more than 50 pixels from the centre of the image we also remove it (due to the chromatic shifts in the camera at the edges of the image).

Prior to actual recognition being performed, all red and blue objects are transformed to account for the effective roll of the robot's head so that they are oriented in line with the true centre line. These objects are then sorted (by colour) from left to right in the image. This allows us to combine objects that are close together horizontally (fewer than ten pixels apart).

Once we have distinguished a robot from all other objects near it of the same colour we determine distance to it by looking down in the (rotated) image until we find green (i.e. the ground), and using the DistanceToPoint method that uses three-dimensional trigonometry to generate a distance to the ground at a point in the image return an approximate distance to the robot. More precisely, DistanceToPoint uses the projection from the focal point through the image to the field. This method requires knowledge of the object's height above the field, and the robot's height and pose. The pose is inferred using averaged accelerometer measurements and therefore does suffer from some error and noise sources. Therefore, distance to point isn't highly reliable, particularly at long distances, but it does give an idea of the scale of distance to the robot. At short distances the technique is much more reliable, and is accurate enough to allow the robot to veer around other robots that are directly in its path.

Once the distance calculation has been performed, the distance and heading information is passed to localisation and behaviour. These methods are repeated after red robot detection is complete for all blue objects, with the additional constraint that any blue objects within red robots are since this is a common false positive caused by misclassification of dark parts of an ERS-210 with a red jersey.

3.2 Multiple lookup tables

This year we implemented a dual look-up table system, wherein a different colour table was used for classification when the robot's head was looking almost directly downwards and thereby shadowing the image and darkening colours. This made us able to better separate ball orange and robot red, as the darker oranges were no longer needed to ensure finding the ball in an image, and shadowed red becomes very much darker. This of course requires us to build two look-up tables, but the downwards-table is very simple as only a few colours need to be thoroughly classified and as such not as many images need to be taken.

Although we used multiple look up tables in the competition, it is not clear whether this was an improvement or not. Dark orange tends to merge with red, and adding an extra colour table does not change this fact. In particular, if a robot is using its downwards table and it sees part of a red robot it is very often identified as the orange ball. This was further complicated by the fact that we had not completed implementation of compensation for chromatic aberration in the camera.

3.3 Improved field point detection

The NUbots' vision system in 2003 featured the ability to identify extra landmarks on the field, rather than just the coloured beacons and goals used by many teams. By locating field and sidelines in the image, it is possible to locate points such as the corners of the penalty box, the centre circle and the intersection of the centre line with the sideline (the "centre corner"). After the 2003 competition, these methods were substantially rewritten and improved. This work is described in [6]. The particular points recognised are labeled in Figure 2.

These methods were used only by the NUbots' goalkeeper during the 2004 competition, and were restricted to only search for the two penalty box corner points (at the relevant end of the field). This is largely due to problems caused by the introduction of white robots, which caused a number of false positive problems for the point recognition algorithms.

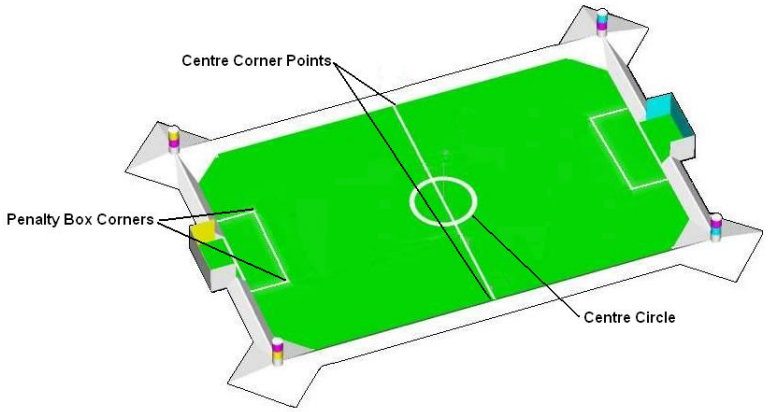

Fig. 2. Landmarks identified by field point detection.

3.4 Improved circle fitting

In 2003, the NUbots applied a circle fitting algorithm to the ball based on optimising a sum of squared error distance criterion [6]. This allowed the accurate determination of the radius of the ball (and therefore the distance to it). The original circle fitting code would only search for points on the ball from left to right (then right to left) in the image, which worked very well when both sides of the ball were in the image. When this is not the case the fitted circle can be quite poor. This is mainly due to poor classification of orange near the edges of the image; a problem made worse by the lower quality camera in the ERS-7. As

a result searching left to right often led to finding false orange edges which led to a poor circle fit (Figure 3).

The solution was to select the search direction based on the position of the ball blob in the image. We used the following rules to decide the search direction - (verticalProblem means a blob comes very close to the top or bottom of the image, horizontalProblem means a blob comes very close to the left or right of the image).

- (!verticalProblemTop AND using low table) = top-bottom (Fig. 3)
- (!verticalProblemTop AND horizontalProblemRight) = top-bottom (Fig. 4)
- (!verticalProblemTop AND horizontalProblemLeft) = top-bottom
- else left-right

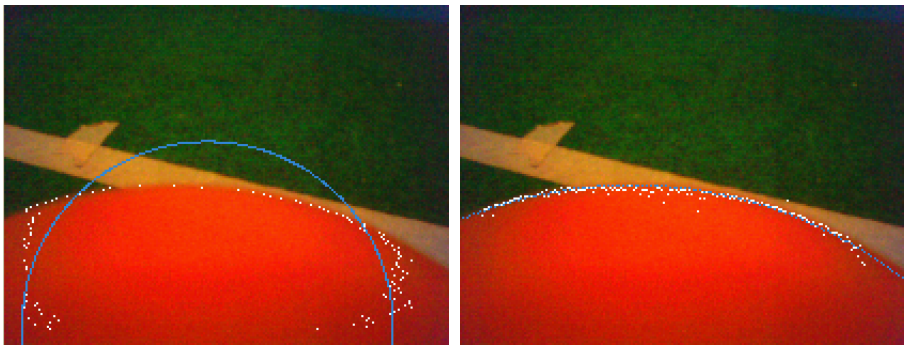

Fig. 3. Ball off bottom of image. Left image is the circle fitted using the old left-right search. Right image is the circle fitted using the new top-down search. The points returned by the search are shown in white with the result of the circle fit displayed in blue.

4 Localisation and World Modelling

4.1 Multiple models

The Kalman filter employed by the NUbots performs well in most cases. However, there is no built-in mechanism for dealing with ambiguity in the provided information. Various situations in RoboCup cause such ambiguities to arise:

- Inherent ambiguities in field data: e.g., it is often not possible for vision alone to distinguish between the two penalty box corners.
- The reliability of odometry data provided by locomotion is dependent on whether the robot is experiencing a collision or not.

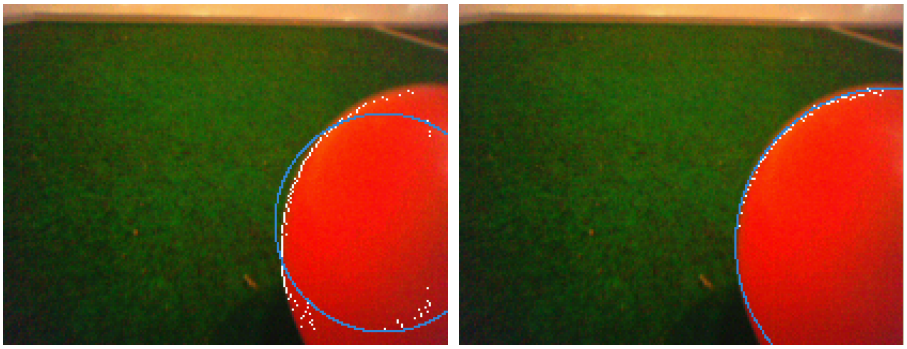

Fig. 4. Ball off right side of image. Left image is the circle fitted using the old left-right search. Right image is the circle fitted using the new top-down search. The points returned by the search are shown in white with the result of the circle fit displayed in blue.

– The ball may be moving rapidly or not at all. There is no odometry information provided, and so it is difficult to tune the filter correctly.

One of the additions made to localisation and world modelling after the 2003 competition is the possibility of executing multiple models of the current world state simultaneously. This necessitated a more modular design in which several Kalman filters, each with different parameter selections, can be maintained at once.

At this point, only two models have been implemented: One assuming that the ball is close to stationary, and the other assuming that the ball is moving quite rapidly. The two models are compared approximately every second, and the model with an estimate closest to that returned directly from vision module is used. If the difference between the best model and the other model is sufficiently large, the poorly performing model is overwritten with the better one and then continues as before.

It is difficult to evaluate the performance of the robot's Kalman filter, and so it is not clear whether the multiple model approach is resulting in more accurate estimation of the ball position at this time. As a consequence, multiple models was not used during the 2004 competition.

4.2 Clipping

Due to noise in the measurements, the Kalman Filter may not always return values of (x,y) that are valid in the robot soccer application, that is, within the soccer field. The exact method by which the robot is adjusted back to within the field of play is well known given a convex boundary region. Unfortunately, the field of play used on the Four-Legged League is a non-convex shape when the goal areas are included. In 2002, the NUbots assumed that the inside of the goal

did not exist as far as localisation was concerned. This caused various strange anomalies in goalkeeper behaviour. In particular, if the robot did indeed find itself inside the goal area, then the clipping caused the estimated position to be in the field of play, and therefore, often behaviour would not know that the keeper was mis-positioned. This was further exacerbated by the fact that once inside the goal, it is difficult to see beacons; 'goal post' objects may be misplaced due to vision difficulties close to a goal (in particular the yellow goal); and, collisions with the goal area are common which makes odometry information inaccurate.

To combat this, a new solution was proposed in 2003: to include the goals in the field region while maintaining a convex shape. This had the unfortunate side effect that certain extra regions (not part of the field of play) were now included in localisation as well. This is illustrated in Figure 5(a). However, this gave rise to a different problem, wherein localisation could now give estimates of position that were behind the goal line. This appeared to give poor kick directions and other behaviour problems in our previous code.

An observation this year is that the robot is only very rarely right at the back of the goal area. It is therefore reasonable to clip the robot's position so that it is never at the back of the goal (and thus reduce the area included in localisation which is not part of the soccer field). The new clipping boundaries are shown in Figure 5(b).

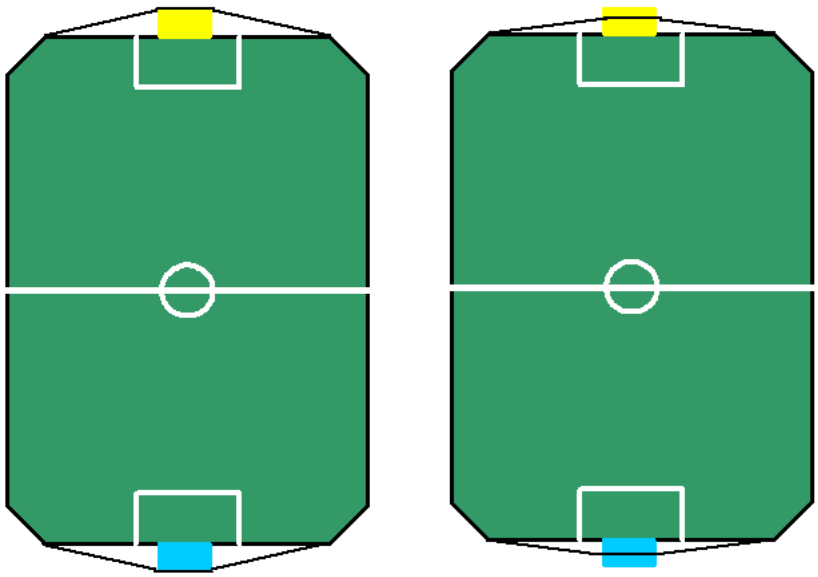

Fig. 5. Field shapes used for Extended Kalman Filter estimates projection

It is difficult to be certain that the new clipping boundaries perform better than the old since various other bugs causing difficulties with keeper localisation were fixed at around the same time. Our observation of the combined effect of these multiple changes was clearly positive however.

4.3 Dead zone

One of the problems noticed in debugging localisation was the tendency of the algorithm to slowly "drift" during extended periods where only limited vision information is available. For example, if over a period of 30 seconds, only one beacon is ever seen, there is insufficient information available to uniquely determine the robot's coordinates and orientation. In this situation, there is a tendency for the estimates from the Extended Kalman Filter (EKF) to slowly drift.

This kind of problem (slow drift in estimation during periods of limited information) has been observed in other contexts including adaptive control systems (e.g. [4, 5]). The source of the problem can be traced to the presence of small correlated (i.e., non-white noise) error sources in the measurements available to the estimator (i.e., in the sense of the NUbots' code, the EKF).

In this case, a useful heuristic, the "deadzone", is very valuable in eliminating the drift. The EKF includes a variable, sometimes referred to as the *innovation*, which describes the deviation of a measurement (e.g., the heading to a beacon) from the prediction of that variable based on prior data. The deadone works by adding extra calculations if the innovations are small (which is usually the case if we repeatedly see the same object).

For each measurement type we define: the deadzone size (ϵ), the innovation (η), the measurement variance (R) and the prediction variance (CPC). The extra code implemented (just prior to doing a measurement update) is then:

$$\begin{aligned}\{\text{if} \quad & 4\epsilon^2 < \text{CPC} \\ & \text{then} \quad R = +\infty \\ & \text{else} \quad R = \max\left(R, \frac{1}{\frac{1}{4\epsilon^2} - \frac{1}{\text{CPC}}}\right) \\ \eta = 0\}\end{aligned}$$

Note that this has no effect for large innovations. However, for the case of small innovations, the effect of making the a-posteriori variance of the predicted measurement at least $4\epsilon^2$, and also, not performing any update on the estimates.

4.4 Improved world model debug application

Like most teams, the NUbots have a (wireless) debugging tool that gives a visual representation of the robot's estimated position and heading on the soccer field. The version used at the RoboCup 2003 competition performed very poorly, only receiving approximately one out of every five packets sent from the robot.

This resulted in the visual representation of the robot's position estimate being very jumpy and difficult to read. Additionally, it was difficult to ascertain what information had caused the robot's position estimate to change, since the debug application would only be notified of one out of every five objects that the robot had observed.

Although the team was well aware of the problem in 2003, it was not clear how to fix it since similar debug applications (e.g., for vision) were able to stream much larger amounts of data from the robot. For reasons that remain unclear, increasing the TCP buffer size to 8192 (rather than the calculated upper bound for the data being sent) for the appropriate TCP endpoint on the robot fixed the problem and so the debugging tool is now able to receive virtually all information sent by the robot.

With post processing via MATLAB® and using estimates based on video-taping simultaneously with streaming data, we are able to get graphical data such as that shown below:

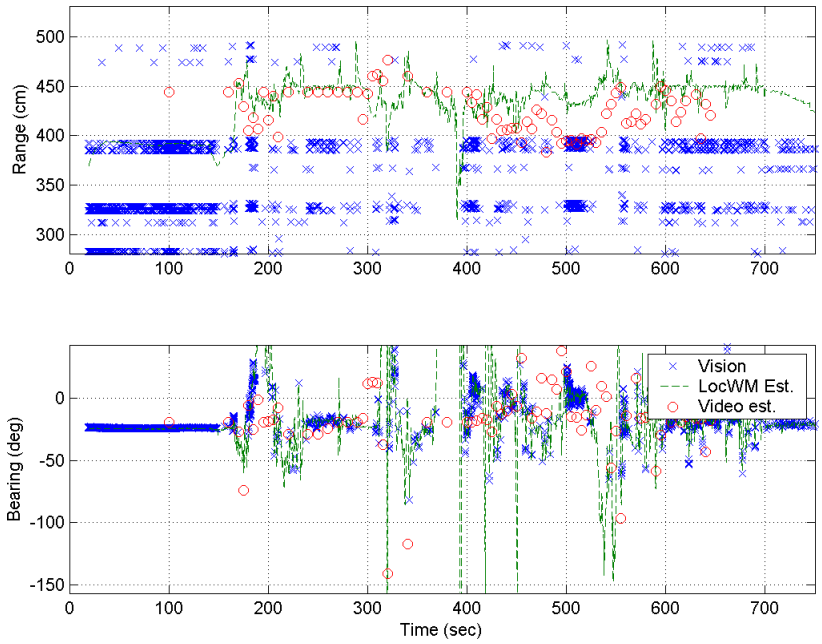

Fig. 6. Robot vision data (blue crosses), robot localisation module estimates (green dashes) and video-tape estimates (red circles) for the robot's range and bearing to a particular beacon.

We are also able to execute our localisation system entirely offline (one frame at a time), based on vision and odometry data streamed from the robot. This also allows algorithm testing and debugging. Future developments in this area will

focus on means for generating 'ground truth' data - i.e., accurate independent measures of the robot's location and orientation. In Figure 6, this data (the circles) is generated by manual interpretation of video tapes of a match.

5 Locomotion

5.1 Arbitrary locus

Various locus shapes have been tested, including rectangular, elliptical and trapezoidal. Although the trapezoidal locus has provided us with the best results in the past, it is not clear that this shape is optimal. With this in mind, a system to easily allow arbitrarily complex locus shapes has been developed.

For reasons of efficiency and flexibility, the algorithm for defining and following an arbitrary locus is quite involved. More precisely, the arbitrary locus must be deformable to allow the walk parameters to easily control the robot's motion and maintain the advantages of a highly parameterised motion engine. Additionally, there is only a very limited amount of processor time available for locomotion, meaning that time consuming operations can only be performed occasionally.

An arbitrary locus is defined by an arbitrary number of 'critical points' that, when joined together by straight lines, represent the desired locus shape. The critical points need not be a fixed distance apart. For a rectangular locus, the critical points would be the four corners. Only the shape of the locus is relevant, as its size is scaled later. Although straight lines may seem unnatural, we would argue that using straight lines is much less of a limitation than it seems, as an arc can easily be approximated by using several points. Furthermore, there is little advantage in the extra complexity of more complex curves (e.g. splines) as the robot's joint motors and servo controls smooth out the leg motion anyway.

When a new set of walk parameters is received by the locomotion engine, a number of steps must be performed in sequence to prepare an arbitrary locus for use. Firstly, the critical points must be deformed to account for walk parameters. For example, the critical points may have to be stretched for a large stride length, or compressed for a small stride height. This is a simple matter of scaling the coordinates of the critical points so the shape they define is of the appropriate dimensions.

The next - and final - step before walking can commence is interpolation. This task is performed prior to walking primarily for reasons of efficiency. The algorithm linearly traverses the deformed critical points and defines new points at (small) fixed intervals. These evenly spaced points allow the walk engine to efficiently control the timing of the locus traversal when actually walking. Simply put, the walk engine must be able to very efficiently calculate the exact position of each leg in 3-dimensional space at any given time during a step.

5.2 Walk learning

A version of Hill Climbing [3] was chosen to allow automated learning of 'optimal' walk parameters because it can provide good results in parameter optimisation

problems. We have chosen to use an Evolutionary Hill Climbing with Line Search (EHCLS) algorithm, which should provide a reasonably quick learning time.

The walk engine is determined by a vector θ consisting of 11 walk parameters (turn and strafe are excluded from the learning) and the critical points defining an arbitrary locus shape. Each parameter is randomly set to an initial value; for our task we make sure this initial vector is feasible (i.e. it is one that will cause the robot to move in the required direction).

In our experiment one episode is to be defined as two runs of approximately $190cm$ each with $time$ being the number of vision frames processed during the episode. Both the number of runs and distance can be modified to suit the environment.

We ran the learning algorithm on the ERS-7 robot from a set of parameters that were developed on the ERS-210 robot. In one evening we managed to learn a walk with an approximate speed of 41cm/s.

Since we used parameters associated with an ERS-210 as the starting point, the NUbots ended up with a walk that involved long smooth steps (very similar to the optimised ERS-210 walks developed by various teams). This seems to be in contrast to many teams that developed short and fast stepping walks. In terms of straight line walking speed it is unclear what approach is better, but it does appear that the short stepping method greatly increases agility and reduces the robot's apparent reaction time.

5.3 Odometry calibration

The locomotion engine uses inverse kinematics to calculate the desired joint angles required to perform a step of a particular size. Unfortunately the distance actually travelled rarely equals the expected distance, resulting in both the behaviour and localisation systems experiencing problems. To overcome this problem, the desired forwards, strafe and turn components of the step are multiplied in such a manner that the resulting step moves the robot the desired distance. In the past this was a completely manual job and for that reason some of the odometry calibration was done poorly or not at all. This year the task was semi-automated.

Firstly the multipliers are set to 1, and the robot is commanded to walk to a home position on the field. At this stage the robot is instructed to walk 10 steps of a particular size (i.e. forward steps of size 20mm). After panning for beacons, the robot can then use localisation information to determine how far it travelled and thus calculate the multiplier required for this walk command. The effect of the multiplier on the walk is non-linear so we can not simply take this calculation as accurate, instead we create the new multiplier to be the average of the old multiplier and the calculated multiplier. The robot will then proceed back to the home position and the experiment is then run until the multiplier converges. After converging for a step of a particular size the robot will then vary size of the step (i.e forwards 40mm) and calculate the multiplier required for this step. It will continue to do this until a predefined maximum step size is reached. At

this stage it will then follow a similar procedure for walking backwards, strafing and turning.

The result is an array of multipliers for steps of various sizes which is then used to estimate a function for that multiplier. Currently the function estimation is done off-line but future versions may perform this step automatically on the robot.

5.4 Kicks

The arrival of the ERS-7 meant that *all* kicks had to be redeveloped.

The shape of the new robot meant that some of our favorite kicks were basically impossible to rebuild. For example the forwards kick that we used so successfully at RoboCup 2003 was very unreliable even after hours of tweaking. This kick appeared to fail because the robot is slightly wider at the front (i.e., it has a slightly greater separation between its two front legs).

The new robot did allow kicks that were previously impossible. For example, many teams had very effective head kicks and we (along with CMU) employed a backwards kick. We had two versions of this kick, one would kick at approximately 110^o while the other would kick at 150^o. Both of these were fast to execute and were relatively reliable. It should be noted that we never tried to accurately direct these kicks, they were only used if the robot found itself facing in the opposite direction and had to clear the ball immediately.

We also made two versions of the slap kick used by UPenn at RoboCup 2003. The 'high' version was used when slapping at a distance, it was capable of sending a ball the length of a field. The 'low' version was used when the ball was close (i.e. after a grab), it kept the height of the robot low enough that the ball could not pop back under the slapping leg (this could occur with the original 'high' version).

Along with these kicks we had a forwards head kick that could be aimed at an arbitrary angle. This was done by moving one of the from legs to an angle and propelling the ball along this leg by moving the head from side-to-side. This kick allowed for soft and precise kicking around the goal, although it was rather time consuming to execute.

Our robots also had a variety of other kicks that were actually combinations of steps, these acted like small turn kicks and were designed to move balls down the wall.

6 Behaviour

On the whole our high level behaviour remained relatively unchanged from 2003. Some bugs were corrected and some relatively minor logic changes were implemented.

The only major change was the amount of time we would take with the ball. In 2003 we would often take the entire 3 seconds to release the ball. This resulted in extremely good accuracy against slower teams (hence our large number of

goals that year). However, this strategy is not as useful against teams with good reaction times and speed, since during the time taken to line up a shot, frequently an opponent has tackled the ball. Therefore, for Robocup 2004 we decided that we would take less time with the ball and accept that this would lead to less accurate kicks, though the improved timing would help against faster teams.

6.1 Player positioning

The increased speed of the robots allowed us to spread the robots out a little more and still completely cover the field. The positioning of the goal keeper was also changed. In 2003 when facing the side of the field, the keeper was often positioned slightly too far from the goal line. This left a small gap that the ball could slip through along the back wall and into the goal. This problem was addressed by shifting the semi-circle the keeper would occupy backwards when the ball was near the defending corners.

6.2 Wall behaviour

In previous years, "scrums" often occurred along the side walls. These "scrums" can easily be caused by a combination of the difficulties in walking, grabbing and kicking when close to a wall. A team that can quickly and easily move the ball along the field walls is able to gain a considerably advantage in this situation. We implemented specific wall code that involved dribbling the ball with the paw (known as 'ram') along the length of the wall (as opposed to attempting to grab or kick it). This meant we could propel the ball effectively up the wall while still walking at top speed. This strategy of maintaining high walk speed whilst dribbling down the side walls proved effective against most other teams. In particular, we believe this contributed significantly to our success, particularly in the closer games against UT-Austin and UPennalizers.

Because the 'ram' of the ball worked so well we actually applied this technique when not on the walls. For example when we are close to the goal and lined up directly we simply 'ram' the ball in, rather than trying to grab, turn and kick.

Detection of the wall was done in a similar method to that described in the 2003 report [1]. It was found that we had to tune this method slightly due to the white robots on the field.

6.3 Kick-off code

For 2004 the robots would automatically set up for kick-offs. The current version of the rules doesn't give many options for a defensive kick-off. Our robots would space themselves across the field on a line 107cm from halfway. When the game was started they would all chase for one second after which the normal chase and positioning decisions were made.

For an offensive kick-off the robots had specific tasks, one robot would be the defender and return a position 107 cm from halfway. A second robot would be a

winger (in the second half of this semi-final against the German Team this robot was changed to be more defensive). The third robot would loop around the ball and then approach in such a manner that his paw would be lined up with the middle of the ball. The robot would stop exactly 15cm from the ball, when the game started he would then continue to line the ball up with his paw and 'ram' it at angle up the field. Because the robot did not need to stop to grab or kick we were able to clear the ball quickly from the centre circle. The direction of the 'ram' was decided at run-time based on the location of the robot when the first ready message was sent.

6.4 ERS-7 head movement

The ERS-7's head features a pair of tilt motors and a single pan motor. Correct head control is critically important: the robot must line up the ball so it is in the centre of its camera image or risk losing track of it too easily if the ball is kicked or discarded by an overzealous vision sanity check. However, the presence of two tilt motors introduces a certain ambiguity into head control, in that there can be more than one way for the head to look at a particular point on the field. If the tilt motors are used incorrectly, the robot's head may end up in a position where it will tap the ball as the robot approaches it.

Controlling the head to focus on the ball or another object is a visual servoing problem. It is therefore natural that we turn to this field in looking for the most efficient way to control the robot's head. We define the angle of the pan motor by θ_{pan}, and the two tilt motors by θ_{big} and θ_{small}. The "small" tilt motor is the one that pivots inside the head of the robot, while the "big" tilt motor is the one that pivots inside the body of the robot. In general, the true head angles, (x_{pan}, y_{tilt}) will be related to the joint angles by a nonlinear equation encapsulating the forward kinematics of the robot neck:

$$(x_{\text{pan}}, y_{\text{tilt}}) = \underline{f}\,(\theta_{\text{small}}, \theta_{\text{big}}, \theta_{\text{pan}}) \tag{1}$$

Note that to avoid doing on-line inverse kinematics, and also since we need to solve the problem of actuator redundancy, we convert to a small deviation model of (1). The change in the three joint angles is denoted by $\delta\theta_{\text{pan}}$, $\delta\theta_{\text{big}}$ and $\delta\theta_{\text{small}}$. We model the change in the effective angle of the head - in terms of pan (δx) and tilt (δy) - using the Jacobian of $\underline{f}$ as follows:

$$\begin{pmatrix} \delta x \\ \delta y \end{pmatrix} = \begin{pmatrix} 0 & \sin(\theta_{\text{pan}}) & 1 \\ 1 & \cos(\theta_{\text{pan}}) & 0 \end{pmatrix} \begin{pmatrix} \delta\theta_{\text{small}} \\ \delta\theta_{\text{big}} \\ \delta\theta_{\text{pan}} \end{pmatrix} \tag{2}$$

For brevity, we define the Jacobian matrix $\mathbf{X}$ as:

$$\mathbf{X} = \begin{pmatrix} 0 & \sin(\theta_{\text{pan}}) & 1 \\ 1 & \cos(\theta_{\text{pan}}) & 0 \end{pmatrix} \tag{3}$$

In order to aim the head at a particular point, we determine the distance in degrees that the head must move in each axis in order to aim at the point -

that is, we determine a δx^* and δy^*. Given these target angle changes, we must determine how to drive the head motors to aim at the desired location (i.e., obtain $\delta\theta_{\text{pan}}$, $\delta\theta_{\text{big}}$ and $\delta\theta_{\text{small}}$). Clearly, we must solve (2) for $\delta\theta_{\text{pan}}$, $\delta\theta_{\text{big}}$ and $\delta\theta_{\text{small}}$, and where we set $(\delta x, \delta y) = \alpha(\delta x^*, \delta y^*)$ with α a positive real constant determining how rapidly we try to correct visual servoing errors.

Note that the matrix $\mathbf{X}$ is not square so we cannot determine its inverse. Instead, we use the pseudo-inverse (denoted $\mathbf{X}^+$), which can be calculated using:

$$\mathbf{X}^+ = \left(\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{X} \tag{4}$$

Then assuming $\alpha = 1$ (which our observations show to be both fast and stable), we have:

$$\begin{pmatrix}\delta\theta_{\text{small}}\\ \delta\theta_{\text{big}}\\ \delta\theta_{\text{pan}}\end{pmatrix} = \left(\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{X}\begin{pmatrix}\delta x^*\\ \delta y^*\end{pmatrix} \tag{5}$$

This solution results in the head aiming in the correct direction using the smallest possible change in joint angles from the current location. However, direct implementation of (5) is limited since it does not limit the range of movement of the motors. This can give rise to problems such as the robot accidentally hitting the ball with its head. Fortunately, it is possible to modify the formulation above to lock certain joint angles at particular times.

7 Challenges

7.1 Open challenge

The NUbots' managed to place third in the RoboCup 2004 Open Challenge.

The setup was quite simple: The orange plastic ball used in competition was placed among a number of other bright orange objects (two orange hats, a large yellow/orange/red beach ball, an orange cup (provided by UTS Unleashed) and an orange shopping bag). The objects were all placed in a line with some random ordering determined by a member of the audience. The robot's task was to walk up to the orange ball only. This is by no means a simple task: Most teams will have experienced the difficulty of having robots wrongly identify anything and everything that is orange as being the ball!

The robot performed this particular task perfectly, although it failed to perform in a more challenging experimental setup. This success is made all the more impressive by the fact that only one line of code was written specifically for the challenge - and it was written early in the morning on the day of the challenge.

Since early 2003, the NUbots have performed circle fitting on the ball in order to obtain better distance estimates. The mechanisms for doing this are explained in [6], and a minor improvement is described in Section 3.4. The circle fitting method used returns a standard deviation indicating the quality of the circle fit provided. If this standard deviation is very high, then the orange blob being considered is unlikely to be the ball. It is precisely this feature that is

used to prevent the robot chasing orange objects that are not the ball: If the standard deviation is too high, the orange blob under consideration to be the ball is rejected. There are issues with using this feature in games, however - it tends to reject balls that are very close to the robot and may also reject balls that are partially occluded (for example, when grabbed by an opponent robot). It is currently thought that a more sophisticated method of combining different sanity checks will solve this problem.

7.2 Variable lighting challenge

Our implementation of the variable lighting challenge was based on a multiple-look-up-table system wherein up to five tables were created under varying lighting conditions, from the brightest (the standard look-up table) to the dimmest (no overhead lighting and all strobes covered). Prior to the commencement of the challenge the images from which these tables were built were scanned and the average Y-value (i.e. intensity) for field green was calculated for each and kept in respect to the table to which is is related. When the challenge is running, once every five frames a competition is run to see which table has a Y-value for field green closest to that of the green in the current image (based on the current classification table - it is assumed that the lighting changes would not be immense in the sixth of a second that this test is run), and the default look-up table is swapped to the winner.

The system worked well under test conditions in our lab, but there were a few minor modifications made prior to the competition that we did not have time to test on the day and one of these caused our robot to crash on start-up, and hence we could not compete in the challenge proper.

8 Game Reports

This section outlines our results at Robocup 2004 along with our comments and opinions on each game.

Round Game 1 vs SPQR (8-0)

This was a replay of our quarter-final match from Robocup2003. SPQR were still using ERS-210s so our ERS-7 robots had a significant speed and power advantage. The margin of victory was less than the corresponding QF match from last year. While happy with this result it was clear we still had some calibration issues, in particular the number of missed grabs.

Round Game 2 vs Metrobots (10-0)

The Metrobots were also an ERS-210 team so our robots had a physical advantage. This game was marred by constant stoppages so it was hard to diagnose any problems. Again it was evident that we had calibration problems not only in grabbing but also in vision. At this stage it was clear that the state of the wireless network would make debugging extremely difficult.

Round Game 3 vs UW Huskies (8-1)

Despite the score line the Huskies were a good ERS-210 team and provided a good fun match. As a team this was our best performance so far as some of the bugs were starting to get ironed out.

Round Game 4 vs Hamburg DogBots (4-1)

This was by far the toughest match of the rounds. The DogBots were an evolution of the 2003 German Team code and were arguably the best full ERS-210 team at Robocup2004. Luckily we managed to escape this match with a victory since the game was almost lost due to the introduction of a serious bug. The change was designed to fix an ongoing problem where the robot would take its eye of the ball, but it resulted in us having no elevation check on the ball for the duration of this game! At various stages in the game our robots can be seen chasing a member of the Dutch AIBO Team who was standing on the practice field (wearing a bright orange shirt).

Round Game 5 - Jollie Pochie (9-0)

This was our first game against an ERS-7 team so we were extremely interested in how we would go. We felt out robots had a speed advantage and a team co-operation advantage, and these added up to good result for us. Although the result was 9-0 we felt that Jollie Pochie were the second best team we had played so far. They had a very powerful kick and although it required a long setup time it was probably the most powerful kick possessed by any team in Lisbon.

Quarter-Final - UT-Austin Villa (6-5)

This was a very exciting game, mainly due to the high scoring back and forth nature of the match. It was also interesting to watch because of the contrasting walking style taken by the two teams with UT-Austin using a very short and fast step. We also noticed that our dribbling down the wall code gave us a distinct advantage and probably won us the game.

For this game, we were required to switch fields. Unfortunately, our robots suffered from vision problems during the match resulting in a large number of poor kick decisions. After the game we realised that our robots were seeing yellow goals inside the white walls (a problem we later discovered had been seen and addressed earlier in the week by several other teams). Even so, the game was very exciting and it is not clear how much the vision problems affected the robots' performance.

Semi-Final - German Team (2-9)

The German Team performed very well in the competition, and we were not able to match them. Their goal keeper localisation seemed to be very good and their chasing behaviour excellent. It also became clear that our kick-off strategy of rushing forward had been effectively countered by the German Team.

In particular, when it was our kick-off in the first half, GT rapidly gained control of the ball, and got into an attacking position, whilst the NUbots had advanced too far to be most effective in defense. The kick-off positions were altered during the game (using a timeout), which seemed to help slightly, but this was not enough to alter the outcome of the game.

3rd Place Playoff - UPennalizers (5-4)

As with the Quarter final match against UT, this was a close exciting match, with the lead changing several times during the match.

Final Results:

1. German Team (Germany)
2. UTS Unleashed (Australia)
3. NUbots (Australia)

9 Conclusion

Acknowledgements The NUbots are grateful to all colleagues, friends, previous members, and other supporters of the team including Peter Turner and other technical support and administrative staff in the School of Electrical Engineering & Computer Science and the Faculty of Engineering & Built Environment. Financial support for the NUbots' participation in RoboCup 2004 was provided by the ARC Centre for Complex Dynamic Systems and Control (CDSC) and a Research Infrastructure Block Grant 2004 from the University of Newcastle in Australia.

Links to the NUbots' publications can be found at the NUbots' webpage

`http://robots.newcastle.edu.au/`

References

1. J. Bunting, S. Chalup, M. Freeston, W. McMahan, R. Middleton, C. Murch, M. Quinlan, C. Seysener, and G. Shanks. Return of the nubots ! the 2003 nubots team report. Eecs tech report, University of Newcastle, 2003.
2. S. Chalup, N. Creek, L. Freeston, N. Lovell, J Marshall, R. Middleton, C. Murch, M. Quinlan, G. Shanks, C. Stanton, and M.-A. Williams. When nubots attack ! the 2002 nubots team report. Eecs tech report, University of Newcastle, 2002.
3. Stephan Chalup and Frederic. Maire. A study on hill climbing algorithms for neural network training. In *Proceeeding of the 1999 Congress on Evolutionary Computation (CEC'99)*, pages 2014–2021, 1999.
4. R. H. Middleton, G. C. Goodwin, D. J. Hill, and D. Q. Mayne. Design issues in adaptive control. *IEEE Transactions on Automatic Control*, 33(1):50–58, 1988.
5. C. Samson. Stability analysis of adaptively controlled systems subject to bounded disturbances. *Automatica*, 19:81–86, 1983.
6. C. J. Seysener, C. L. Murch, and R. H. Middleton. Extensions to object recognition in the four-legged league. In D. Nardi, M. Riedmiller, and C. Sammut, editors, *Proceedings of the RoboCup 2004 Symposium*, LNCS. Springer, 2004.