The 2006 NUbots Team Report

Michael J. Quinlan Naomi Henderson Richard H. Middleton
Steven P. Nicklin Robin Fisher Florian Knorn
Stephan K. Chalup Robert King

February 22, 2007

School of Electrical Engineering & Computer Science
The University of Newcastle, Callaghan 2308, Australia
http://www.robots.newcastle.edu.au

Contents

1 Introduction

1.1 Overview

RoboCup 2006 in Bremen saw the NUbots win the four-legged league at RoboCup for the first time. This success comes on the back of a 2nd place finish in Osaka (2005) and three 3rd place finishes, Lisbon (2004), Padua (2003) and Fukuoka (2002).

The strength of our performance in 2006 can strongly be accredited to our 2005 development [Quinlan *et al.*, 2005; Henderson, 2005; Hong, 2005; Nicklin, 2005]. 2005 saw a complete redevelopment of our RoboCup code, the result being a base that was easy to extend and debug in 2006 (and hopefully 2007). One of the main causes for success this year was special attention being shown to particular elements in localisation, mainly ball velocity and close-to-goal localisation. This, in combination with a few (key) fixes from 2005, led to a substantial increase in our overall game performance.

1.2 Summary of important changes

Below is a list of the *important* changes,upgrades and fixes that we performed in 2006. Where possible we have referenced the relevant section of the report.

Vision

- Fixed goal gap detection (Section 2.7.5)
- Relaxed 'narrow' goal check (Section 2.7.2)
- Updated goal post recognition (Section 2.7.4)
- Adapted to the 'pink' beacons at RoboCup (Section 2.5.2)

Localisation & World Model

- Alternate Ball Measurement Coordinates (Section 3.1)
- Incorporation and Tuning of Deadzones (Section 3.2)

Behaviour

- Increased reliance of velocity when grabbing (Section 5.2)
- Catching (Section 5.2.3)

- Auto-switching between high-level strategies (Section 4.4)
- Entirely new goal keeper (Section 4.6)
- Fixed a *massive* bug regarding search direction

1.3 Software Layout

Our 2006 system was built upon our 2005 architecture and has remained almost unchanged. Our system consists of one control module *NUbot* and four functional modules - *Vision, Localisation & World Modeling, Behaviour and Locomotion.*

1.4 2005 & 2006 Design

The flow of information in our system can be seen in Figure 1. It has been observed that in every year since 2002 the code has been migrating towards a single global store of variables (approaching a whiteboard architecture), these variables can then be written to and read by each module as required. In the past we have run into problems where new ideas require variables that were private in other modules, often it is not trivial to retrospectively make these variables public or to pass them around. The complete rewrite in 2005 enabled us to setup the principle of the global store, in addition we have also forgone the use of private variables in all classes.

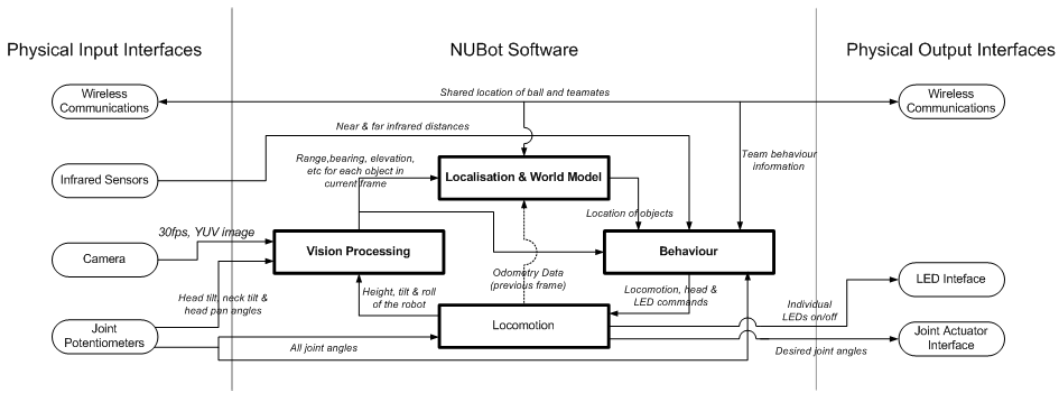

Physical Input Interfaces
NUBot Software
Physical Output Interfaces
Wireless Communications
Shared location of ball and teamates
Wireless Communications
Near & far infrared distances
Infrared Sensors
Range,bearing, elevation, etc for each object in current frame
Localisation & World Model
Team behaviour information
Location of objects
Camera
30fps, YUV image
Vision Processing
Behaviour
Odometry Data (previous frame)
LED Inteface
Head tilt, neck tilt & head pan angles
Height, tilt & roll of the robot
Locomotion, head & LED commands
Individual LEDs on/off
Joint Potentiometers
Locomotion
Joint Actuator Interface
All joint angles
Desired joint angles

Figure 1: 2005 & 2006 Software Architecture

2 Vision

The 2005 vision module was strong enough that only fixes and enhancements were required for 2006. This section details the major features of the vision system and the reasons behind them.

2.1 Overview

The 4 legged league is unique in that we must deal with the limitations of the specified hardware. In the case of the vision module we must be able to extract as much information as possible about the environment from a poor quality, 208x160 pixel YUV 24bit colour CMOS camera at a rate of 30 fps.

Our vision module follows the process: *classification, blob formation, soft colour blob filtering, object recognition and line detection.*

2.2 Image Quality and Camera Settings

2.2.1 Camera settings

To achieve the best quality image for the speed of movement and lighting conditions the following camera settings are used. **Shutter speed:** This is set to fast. This minimises blurring but unfortunately also minimises brightness of image. **Camera Mode:** This is set to fluro as the other settings have a blue impact on the images. **Gain:** The gain is set to high as it has the best effect on images for our use.

2.2.2 Field Lighting and Effect on Images

RoboCup rules state that the field is lit to a minimum of 1000 lx. Lighting dramatically effects image quality. In lab testing environment we prepare for the 'worst case scenario' as a system that deals well with poor conditions can only perform better in improved conditions. The lab field lighting is on average 1000 lx (lower in shadowed regions, higher in the middle of the field), however lighting conditions at most RoboCup events have been far better. Figure 2 shows the improvement in image quality from the NUbot lab to the 2006 RoboCup field.

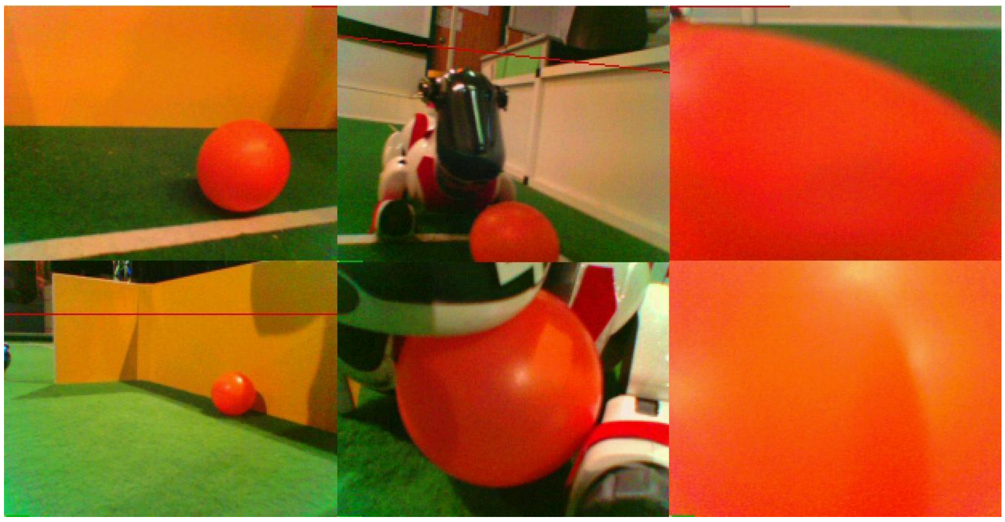


Figure 2: Effect of lighting on images. (Top row) Images taken from lab. (Bottom row) Images taken at RoboCup 2006 with brighter lighting.

2.3 Colour Classification

In the colour coded environment of the 4 Legged League (4LL) it is essential for a robot vision system to extract colour information for the identification of objects. Colour classification is the process that maps pixel values of an image to a colour label that corresponds to a region of colour space pre-defined in a look up table (LUT). Colour Classification systems, typically are based on hard colour classification, i.e. only classify those pixel values of images that directly identify an object. The problem with this method is how to classify overlapping colour space; the region of colour space that is non unique to an object. Typically these areas are not classified in the calibration process leading to possible misses, or a compromise is made and it is classified as one of the two leading to potential false positives. Both of these options have dire consequence in a game situation where the ball and robots move fast enough that any missed or false information processed in a frame effects the entire performance.

2.3.1 Soft Colour Classification

Soft colour classification accounts for this overlap by fully separating unique colour regions and defining overlapping regions as soft colour. Soft colour is basically 'either colour' and allows for a delay in the colour decision making process until further information is found. The colour decision is then made based upon object colour and shape properties in the soft colour filtering stage.

2.3.2 Colour Calibration

Colour calibration is the process of applying colour classification principals to build a LUT. The described soft colour classification system has been implemented successfully for two years of competition using a manual calibration system; however the classification principals can be applied to any calibration method.

When classifying often a compromise must be made when two colours overlap. These colours tend to be; orange and red, orange and yellow, dark blue and dark green, dark green and black and in the case of 2006 competition red and pink greatly overlapped. When calibrating any overlap between these colours are classified as the soft colour; rorange, yorange, blue-green, shadow-black, and pink-red (respectively). Blobs are formed based on classified colours and then modified in the *soft colour blob filtering* stage

2.4 Blob Formation

Grouping of classified colours is used for the transfer of information from an image to object recognition. When forming blobs the colour, size and area information is required. Blobs are formed based on classified colour of the classified image. Undefined, white and field green and the soft colour shadow-black are not included in blob formation. This is due to the fact that size and shape information is not required for objects of these colours. It is also due to the nature of distribution of these colours. They are all abundantly spread throughout images and scattered; forming blobs on images would involve unnecessary processing.

2.4.1 Blob Formation Logic

Blob formation checks every pixel of the image when forming blobs, this means we can form blobs as small as 1x1, however, we only use blobs greater than or equal to 3x3 pixels. This allows us to see the ball the entire length of the field. Furthermore, we can trust the reliability of the colour information due to soft colour classification.

The basic colours that are used in blob formation are termed object blobs and the blobs formed on soft colours are termed soft blobs.

For the width and height of the image each pixel is made the 'current pixel' and compared to that of the pixel prior horizontally and vertically in the methods 'check horizontal pixel' and 'check vertical pixel' respectively to see if it is of the same colour. The pixels can be associated if the classified colour values are the same and they are then allocated the same blob number and the blob object parameters are expanded. This logic is carried out for the entire area of the image. As blobs are formed they are pointed to by an array of blobs with the size equal to the amount of blobs formed.

2.5 Soft Colour Blob Filtering

Once Blobs are formed a decision must be made for soft colours. Soft colour blob filtering sorts blobs by object and according to its colour and shape properties. Each object has particular object-soft colour blob formations depending on lighting, shadowing, angle and distance. These characteristics were thoroughly studied and tested using collections of image streams to create conditions for the object blobs and soft colour blob association.

2.5.1 Ball Object

The key colour that identifies the ball is orange. Orange is in between red and yellow in colour space and so depending on lighting conditions orange can overlap with either colour. For red and orange this tends to occur in images where the ball is in shadowed situations such as; the underside of the ball is under shadow due to the shading properties of a sphere, up close to ball, the ball being held by another robot, etc. For yellow and orange overlap occurs when the ball is brightly lit and reflections occur on top of the ball; again this occurs due to three dimensional properties of a sphere.

When filtering blobs the object colour blob is checked for overlap to the soft colour blobs by comparing maximum and minimum x and y coordinates. Rorange is compared to orange for lower overlap (shadow under ball), entire overlap (up close image of ball) or any partial overlap besides upper and inner. Yorange is compared for upper overlap (reflections) Figure 4. Unused rorange blobs are not ignored until they have been used in the red uniform check, unused yorange blobs are not ignored until checked by yellow goal check. Figure 6 shows images of the ball under different lighting conditions and the application of soft colour classification.

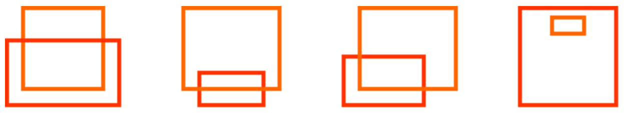


Figure 3: Acceptable overlap of orange and rorange.

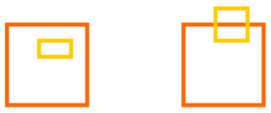


Figure 4: Acceptable overlap of orange and yorange.

2.5.2 Red Uniform

Shades of the red uniform of a robot if not classified correctly can be mistaken for the ball. This has major consequence in a game. Red uniforms when being classified must be completely separated to orange any overlapping colour space can be classified as the soft colour rorange. For a rorange blob to be associated with a red blob it must overlap. Any overlap of red and rorange blobs results in the expansion of maximum and minimum parameters for the red blob. Due to the nature of scattered blob formations of red and rorange blobs a secondary overlap check occurs again expanding the red parameters and from then on the rorange blob is ignored.

Red Uniform Pink Beacon Overlap

The 2006 Robocup pink beacons were a darker pink to the standard. This caused a significant overlap between regions of classified colours pink and red. The principal for classification was to give pink priority when classifying, as a lack of pink beacon information would severely effect localisation, and then modify any red blobs overlapping with pink blobs to be expanded red. This check was completed after the rorange check so that the expanded red blobs were used, hence maximising probability of finding overlap. Figure 7 shows images of the red uniforms, their blob formations and the filtered blobs.

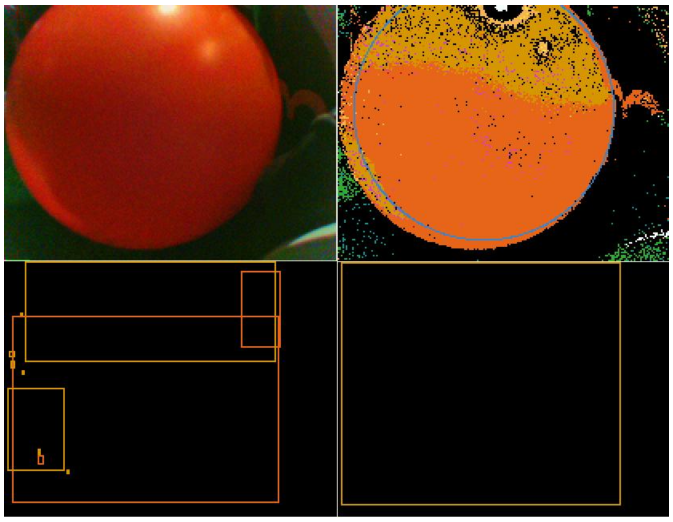


Figure 5: (Top left)Image of shadowed ball.(Top right) Classified image. (Bottom left) Blobs formed. (Bottom right) Filtered and size modified blobs.

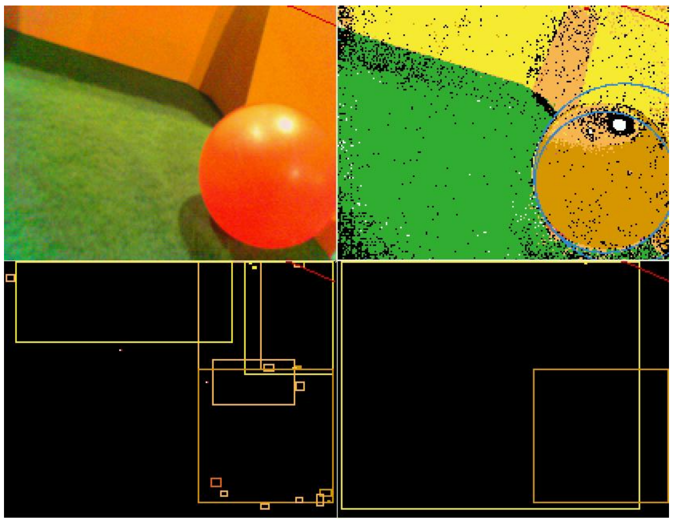


Figure 6: (Top left) Image of ball in yellow goal with reflection. (Top right) Classified image. (Bottom left) Blobs formed. (Bottom right) Filtered and size modified blobs.

2.5.3 Yellow Goal

The beacons and goal are classified yellow where possible. Where possible means where the classified colour *yellow* is unique to the goal area (and beacons) only; however there are shades of yellow that occur in images of the goal and other places

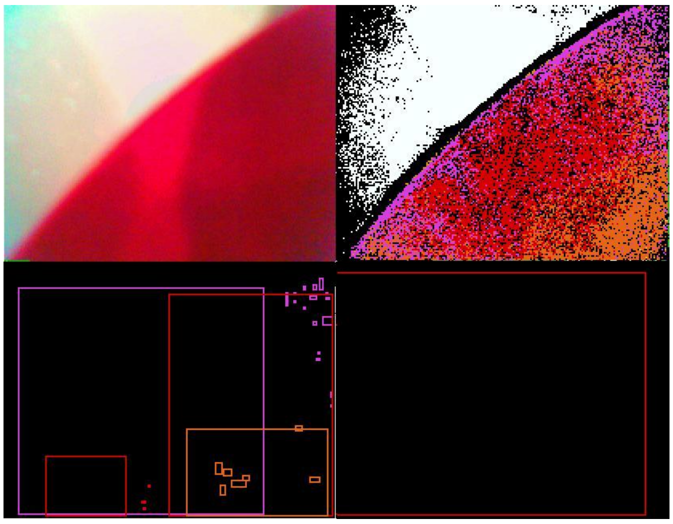


Figure 7: (Top left) Image of red uniform. (Top right) Colour classified image. (Bottom left) Blobs formed. (Bottom right) Filtered blobs.

on the field, e.g. reflections on the ball, reflections off white surfaces, very close to goal, etc. These overlapping regions tend to occur in the shadowed area of the goal, i.e. the corners, posts and up close to the goal. Any overlap (even if a yellow-white overlap) is classified as yorange.

By studying blob formation of goals it was noted that any overlap patterns between the object blob and soft blob were acceptable. For goals and beacons the conditions are: if the soft colour blob is not inside or outside the yellow blob then it must be overlapping. Figure 8 shows the application of soft colour.

2.5.4 Blue Goal

Shades of the blue goal can overlap with shades of field green in particular when either are shadowed. Additionally the poor quality camera has a noticeable ring present at the corners of images, this ring distorts pixels. In the case of images of the field the ring tends to darken and add a blue hue to the green in the corners. This overlap is classified as *shadow blue*. If the classified colours green and blue are unique to the field and goal respectively; any overlap between blue and shadow blue can be used as goal information and any overlap between green and shadow blue can be ignored (as blobs are not formed for green). Furthermore, large shadow blue

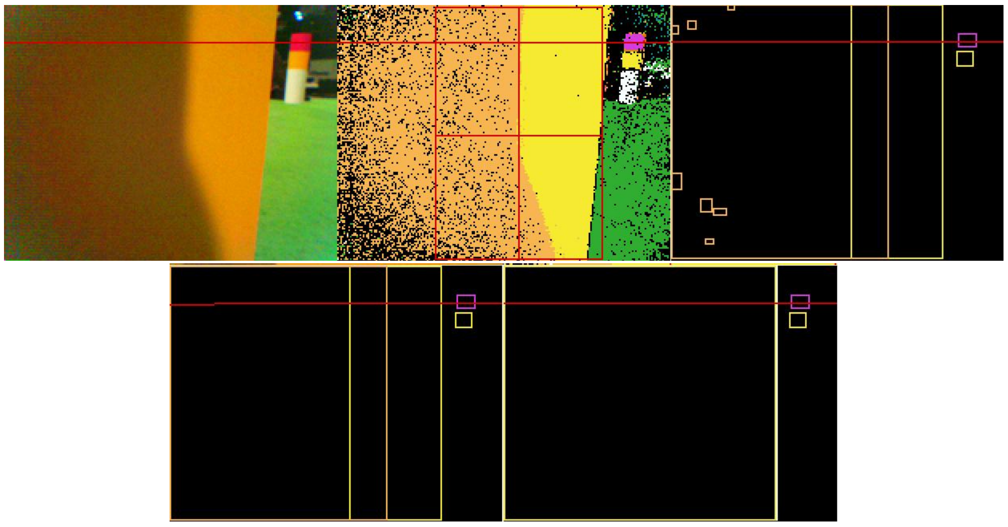


Figure 8: (Top left) Image of close yellow goal post. (Top middle) Colour classified image.(Top right) Blobs formed. (Bottom left) Filtered and size modified blobs.(Bottom right) Recognised goal objects.

blobs have the possibility of being an up close goal or goal post, even if no classified blue is present, these blobs are passed to object recognition also. Figure 9 shows images of blue goals, their blob formations and the filtered blobs.

2.6 Detecting the Ball

The elements that are examined before a blob is identified as a ball include the blobs correctly coloured pixel ratio, height above the ground, colour below the blob, and also its roundness. A confidence system is used to allow individual ratings for each of these elements to be combined into an overall confidence in the identification. The ball recognition code was originally created in Python primarily so that that code could be changed on-the-fly and uploaded to the robots, decreasing the required testing time. This year the code did not require as much development or testing and therefore was ported to C++ to remove the overhead required to move blob information to and from the python environment .

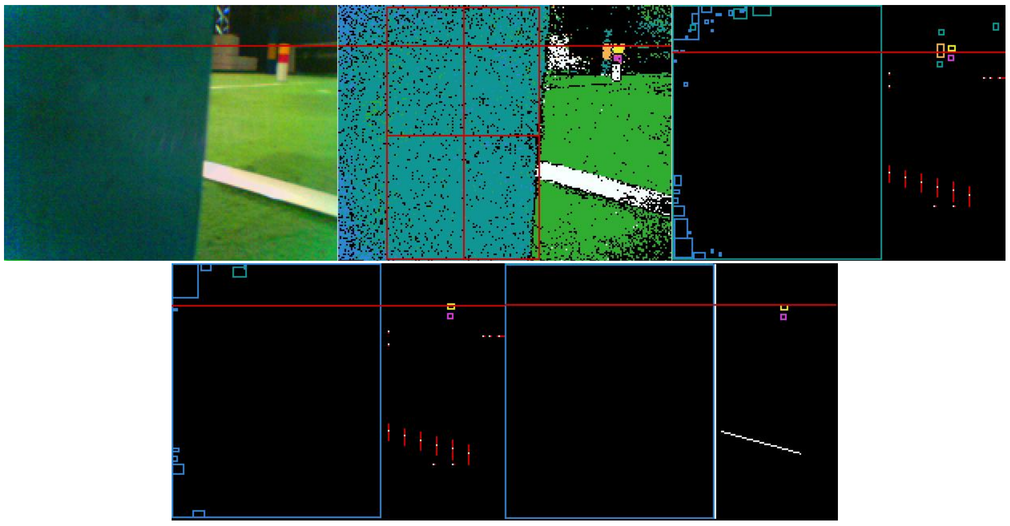


Figure 9: (Top left) Image of close blue goal post. (Top middle) Colour classified image.(Top right) Blobs formed. (Bottom left) Filtered and size modified blobs. (Bottom right) Recognised goal object.

2.6.1 Ball Distance, Bearing and Elevation Calculations

The raw values of bearing and elevation are calculated using the centre x and centre y of the blob in the image, while the raw distance is calculated using the width of the blob in pixels. These resulting values are relative to the camera. They are then transformed using the forward kinematics of the robot to give a relative location in terms of the robots reference frame.

2.6.2 Height Check

The ball rarely leaves the ground during play. Because of this the perceived ball height should be relatively constant. This can be used to identify balls by comparing the calculated height to the expected height. This test relies on both an accurate distance and elevation; therefore any errors in either of these values can cause errors in the height. For this reason a range of heights must be allowed to cater for the noise in the distance and bearing calculations.

2.6.3 Surrounding Colour Test

As previously stated the ball rarely leaves the field during play and most false objects are located outside of the field. From these assumptions testing the colour directly below a blob can be used to determine if an object is on, or off of the field. If the area directly below the blob is the colour of the field, this increases our confidence that the object is located on the field. However this test cannot be used to strictly rule out a blob that does not appear to be on the field as being the ball as there are many situations in which this may still be the case. These include cases where the ball is sitting next to a field line, and cases where robots are occluding the area we expect to be green.

It is expected that a ball by itself would be sitting directly above the green of the field. However the robots camera is constantly moving and so the bottom of the image is not always downwards. Therefore the area in which the colour is checked must be rotated so that it does appear below the object. This required a rotation of the co-ordinates of the pixel before the test is done. The area is moved about the horizon using a coordinate rotation to cater for the current positioning of the camera.

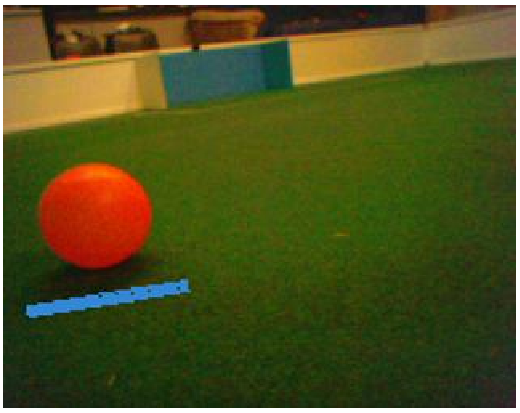


Figure 10: Rotated scan area used for ball recognition (shown in blue).

2.6.4 Circle Fitting

Least squares circle fitting has been implemented to improve the distances, bearings and elevations of balls that are not fully visible in the image. There are two main parts to the circle fitting procedure. Firstly the selection of the points to be fit, followed by the fitting of a circle to these points. These points are gathered during one of various scan types depending on the positioning of the ball in the image. The points are found by searching from the outside of the blob inwards until it finds pixels of the same colour of the blob, or in the case of the orange blobs also one of the soft colours close to orange. The directions for which scans have been created are: *Left to Right, Right to Left, Top to Bottom, Bottom to Top,Simultaneous Left to centre & Right to centre and Simultaneous Top to Centre & Bottom to Centre.* These can be seen in Figure 11. The reason the different scanning directions are needed is so that in images in which the view of the ball is cut off by the edge of the image, points are not selected along the edge of the image in turn biasing the circle fit. For more information on the least squares fitting function used see [Seysener, 2003],[Seysener *et al.*, 2004].

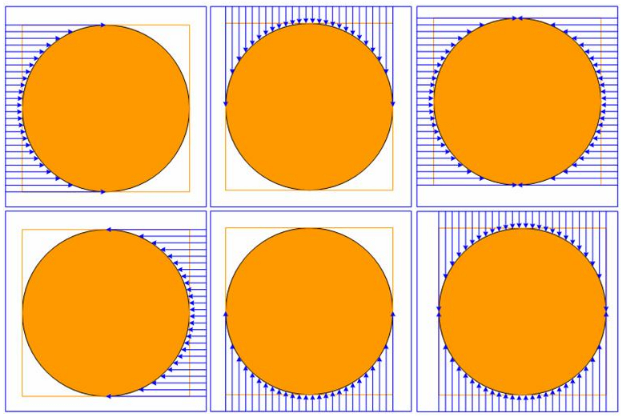


Figure 11: Ball scanning directions, (top left) left to right, (middle left) right to left, (bottom left) left and right to centre, (top right) top to bottom, (middle right) bottom to top, (bottom right) top and bottom to centre.

Once a circle has been fit to these points, the validity of the fit circle is determined. If the diameter of the circle is significantly lower that the width of the blob then it is assumed that the circle fit is not correct.

2.7 Detecting Goals

With the use of soft colour classification and due to obstructions on the field, the goal is often not seen as a single blob, but rather a cluster of blobs. Because of this the blobs must go through a reconstruction process. Once the goal has been reconstructed a series of tests are performed and if the goal has been identified then goalposts can be found. Goal recognition also uses the confidence system, that is as the goal candidate performs well in tests it will increase in confidence and when it performs poorly it will decrease. At the end of the tests if the confidence is greater than zero the candidate is accepted as a goal.

2.7.1 Goal Candidate Construction

The soft coloured blobs are linked to base (i.e. yellow or blue) coloured blobs in a structure called a goal object. These blobs are all combined to get the total size of the object and stored as a goal candidate. These objects are then compared and those that are within a specified range, or those that line up horizontally are merged together to create one object as can be seen in Figure 12.

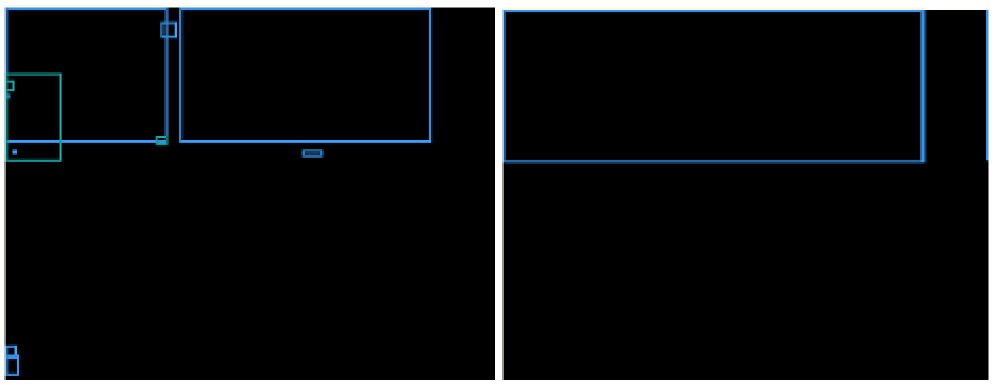


Figure 12: Left: Original blue blobs. Right: Resulting blue goal blob.

One problem that occurs is that when there is a goalkeeper inside the goals it can cause the blob to break into two with a gap at the location of the goalkeeper. To reconstruct this part any goal candidates that line up horizontally (both top and bottom) in respect to the horizon are merged.

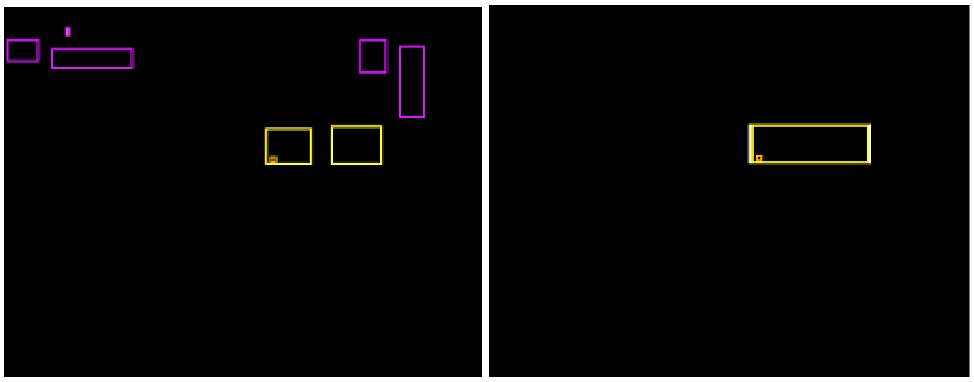


Figure 13: Left: Blobs of goal broken into two by goalkeeper. Right: Resulting goal blob.

2.7.2 Goal Identification

Once the goals candidates are reconstructed they are then tested for identification. The following tests are only applied to goals that do not cover a large majority of the screen since it can be very difficult to get meaningful measurements in this situation. For this reason if a goal candidate appears to be a very close (i.e. fills the camera) it is assumed to be a goal and these tests are skipped. Firstly the colours beneath the candidate are checked, however this test cannot have a high influence on the result as there are many cases in which it will fail. This is because of the number of lines around the goal, and also the large number of robots often found near the goal during play.

Because the goals are stationary, their height should remain relatively constant. For this reason their height in the vertical plane is calculated. This is then compared to a range of acceptable values. If it is not in this range then the candidate is penalised through a reduced in confidence. Elevation checks are then performed; this is because the range of z-values for an acceptable goal needs to be quite large to account for incorrect distances and elevations. Therefore the elevation of the objects is used to help remove the other false goals.

Other tests are done concerning the objects shape and size. This includes comparing the width to the height. A rule previously used was removed that limited the acceptable width for a given height. It was found that this restriction caused many goals to be rejected when looking across the face of the goal. However a similar rule limiting the minimum allowable height for a given width remains as it is possible to recognise when the top or bottom of the goal has been cut-off in the image.

2.7.3 Goal Distance, Bearing and Elevation Calculations

The raw distance of the goal is found by using the height in pixels. The height of the goal is the only feature that remains relatively constant as the robot moves around the field. The problem with this, however is that the robot must see both the top and the bottom of the goal to get an accurate distance. When the robot is close to the goals this can be a problem. Because of this the infra-red distance sensors are used to gain a more accurate distance to the goal when the goal gets very close.

The raw bearing and elevation is found using the centre of the goal blob or in the case of the goal posts their centre points. As with the ball these raw distances are translated back so they are relative to the centre of the robot. If it can be determined that both the top and bottom of the goal cannot be seen, then a flag is set so that this can be taken into account within localisation.

2.7.4 Goal Post Detection

Using the information gathered during the cut-off detection the image is scanned for goal posts. This is done by fitting a line to the sides of the goal using the least squares method. To acquire the points a series of scans are performed parallel to the horizon at regular intervals down the side of the goal. The quality of the line fit is then used to determine the presence of a post. If the line fits the data points well then the side is likely a post. For 2006 the restrictions on the quality of the line fit were increased in an attempt to reduce false positive identifications. This was due to an increase in usage of the posts for localisation.

2.7.5 Goal “Gap” Detection

The locations of the goals play a large part in the NUbots kicking behaviour. Before a robot takes a shot at goal it will first attempt to visually line up the goals. However the centre of the goal (given by the goal detection) is not always the best place to kick the ball as it may send the ball straight to the opposing goal keeper. The best place to kick the ball in a majority of cases is at the largest ”gap”, the goals

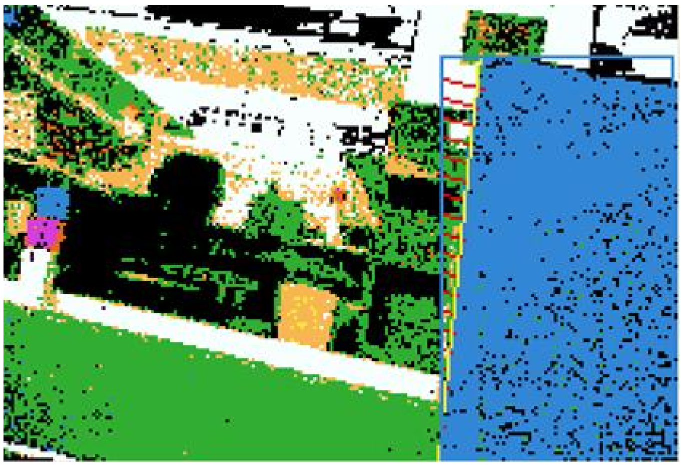


Figure 14: The scanning path for the line fit points are drawn in red, while the least squares line fit is drawn in yellow.

largest unobscured area. This allows the shot to go past the obstacles and therefore increases the chance of scoring.

The gap detection consists of an algorithm that searches from the left hand side of the goal to the right hand side of the goal in the image designated by the current goal blob. An addition to last years goal “gap” detection is the inclusion of multiple scan lines across the face of the goal. It was found that last years single line search was easily disrupted by classification noise and did not always pass through a robot inside the goals due to errors when calculating the height to scan at. This lead to many shots towards the center of the goal and was particularly noticeable in heavily tilted images. Therefore three equally spaced scan lines are now used.

2.8 Detecting Beacons

The four beacons are labeled as pinkYellow (Top = Pink, Bottom = Yellow), yellowPink (Top = Yellow, Bottom = Pink), pinkBlue (Top = Pink, Bottom = Blue) and bluePink (Top = Blue, Bottom = Pink). Because pink is a colour common to all beacons, the beacon recognition code iterates through the pink blobs, taking the largest first. It first checks the pink blobs against yellow blobs then blue blobs. To improve running time the yellow and blue blobs are also sorted by size, largest to smallest.

The recognition code determines which blobs are top and which are bottom by comparing their relative positions. When two blobs are recognised as a beacon, they are flagged as used, and hence ignored by all other tests. When both types of beacon are found for one colour type (eg yellowPink and pinkYellow) the code stops searching for possible beacons of that colour combination, relying on the uniqueness of each of the beacons. Using this method, the worst case operation is $O(2n^2)$.

2.8.1 Beacon Distance, Bearing and Elevation Calculations

The beacon distances are determined using the distance between the centres of the two blobs, while the X and Y bearings are found using the point between the two centres. These values are then transformed using the forward kinematics of the robot to produce values relative to the body of the robot.

Because of under classification and obstructions from both other objects on the field and the robots FOV, in many cases the two blobs formed on a beacon are not equal in size. Therefore to improve upon the distance, bearing and elevation values the smaller blobs size is increased to match that of the larger blob.

2.9 Detecting Robots

The general method of robot recognition begins by finding clusters of robot blobs on the field. Each identified cluster may represent an ERS-7 robot. Clusters of blobs like this are assigned to a robot container. The robot container is a data structure that stores at most three robot blobs, which is sufficient for the code to estimate a robot's orientation. The sufficient condition was derived by observing the possible number of patterns formed by the uniforms as an ERS-7 robot is viewed from different perspectives.

The orientation of robots with red uniforms is easier to tackle as red is a distinct colour and thus red robot blobs are formed around patches of classified red pixels.

Blue robots however are much harder to distinguish from other objects. Due to blue not being a unique colour on the field, and also the confusion shadowing can cause. For this reason edge detection is used for detecting robot blue. The contrast between the white AIBO and the navy blue uniform that is worn in RoboCup competition creates a distinctive contrast. This allows the use of the edge classified image to be used to confirm blue uniforms.

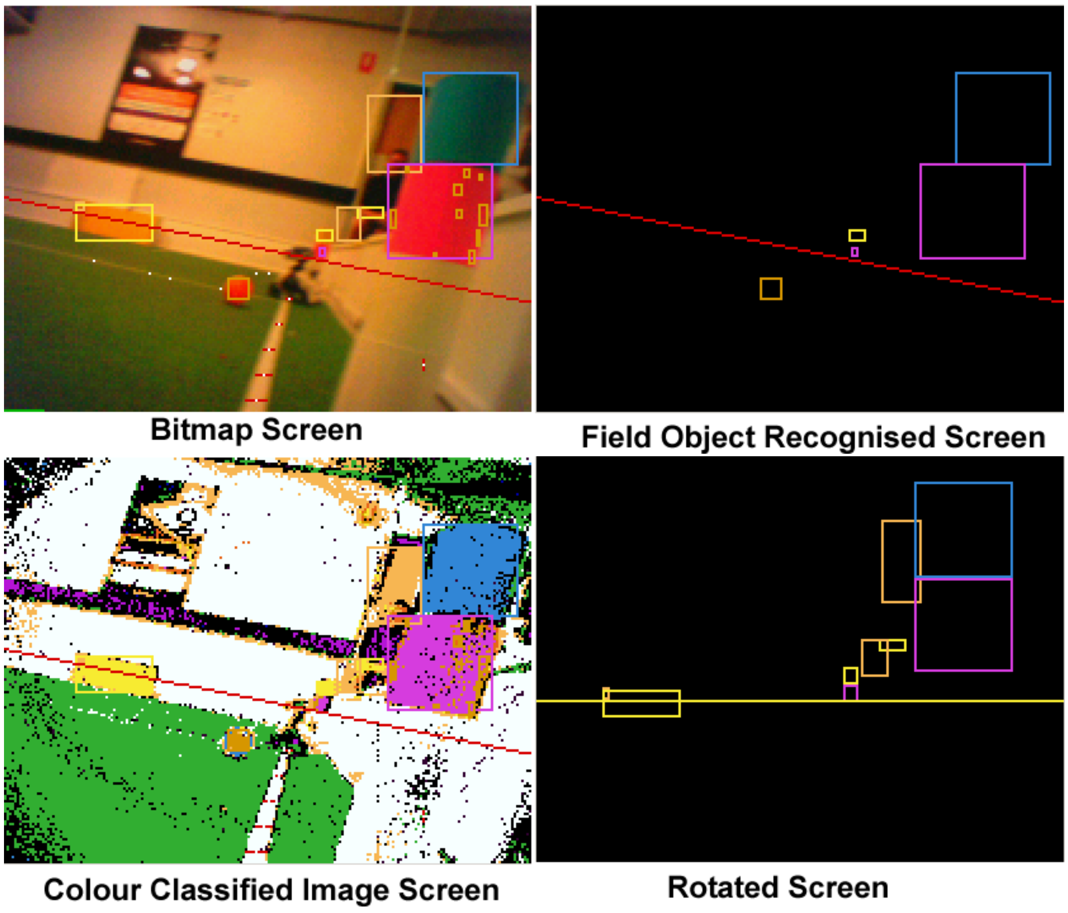

Bitmap Screen
Field Object Recognised Screen
Colour Classified Image Screen
Rotated Screen

Figure 15: The further away the beacon blobs are, the more non-ideal it becomes. Once the blobs have been paired, the smaller beacon blobs are expanded to meet the size of the larger beacon blob (seen in the rotated screen). This creates an ideal scenario, improving the accuracy of measurements.

Orientation comes into play when the assigning code has completed running. Using the blob information stored within the robot container details can then be conferred. The robot recognition code attempts to distinguish the orientations: Left, Right, Front, Side, Back and Unknown using methods similarly described in [Hong, 2005; Quinlan *et al.*, 2005]. In particular scanning of the classified images contributes to the sanity factors that determine the robot's orientation.

Side orientation is defined as being when the ERS-7 robot has moved too closed to the side of another ERS-7 robot that it lacks sufficient information to determine whether the other robot is facing left or right. This can occur when the leg patch is outside the image. Unknown orientation is necessary as not every pattern of robot blobs can be used to determine its orientation.

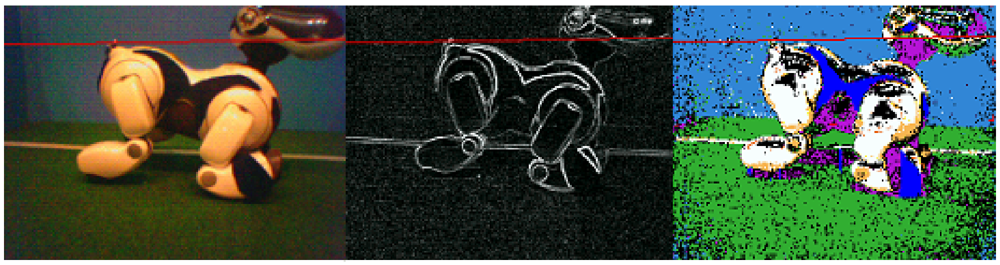
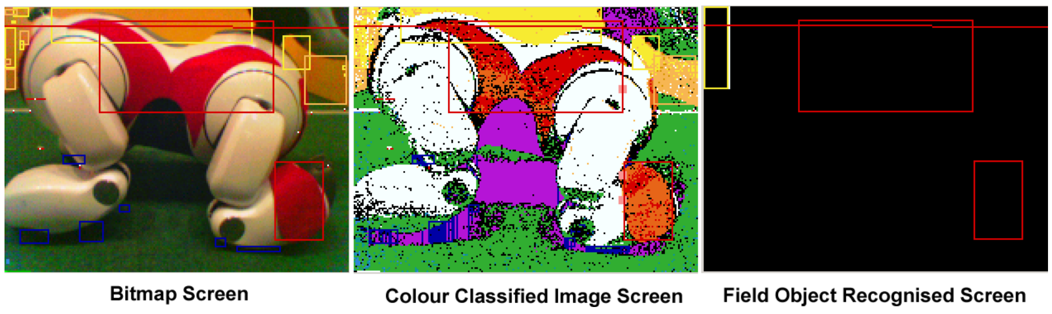


Figure 16: Using the edge classified image the blue uniform is extracted from the image.


Bitmap Screen
Colour Classified Image Screen
Field Object Recognised Screen

2.10 Detecting Field Lines

The detection of field lines is approached in a completely different manner to that of other Objects. When searching for objects, sections of a specific colour are joined together to form a blob. While this works well for most objects, white causes problems due to its abundance in the image and hence is normally ignored. Also the thin nature of lines, and the robot's low Point Of View, makes missing pixels very common within the classified image. Finally the storing of blobs, as a square which bounds the entire object, does not store enough important information about the line.

To over come these issues the detection of lines is based around the two unique features that field lines have; their long thin length and their green-white-green transitions. Using these two details the image can be sparsely searched, since the search line does not need to find every point of the line, just enough to re-create the line. Also the transition information allows many white pixels to be thrown out early in the detection process, thereby reducing the over all load on the processor. This reliance on points and not the entire line also allows the lines to be partially hidden behind another object without effecting their detection.

With these factors in mind, the image can now be efficiently searched in the following steps;

- The image is searched using a horizontal and vertical search grid. The search is restricted to the area in the image below the horizon line and picks out points of sharp contrast which have green-white then white-green transitions. These points are recorded in an array for later use.
- The found points are then checked against each other to form possible lines. Once candidates are found, more points are checked and added until a line is formed. All checks on the lines at this stage is done based on gradient to allow the fastest line formation.
- The lines are cleaned up to make sure all points actually fit on the line. Lines which have too large a number of points not contained within the final line are removed.
- Lines are checked against each other to confirm that they are not just segments of larger lines. The allows lines to be formed that have a break in the middle, such as when a robot is on top of the line.
- Corner points are found by extending lines and finding their intersection locations. The intersections are checked to confirm virtual points are not found.
- An attempt is made to uniquely identify corner points from other objects within the image. While this is not always needed, if a unique identification is made the use of the corner point within localisation becomes much more efficient.

For more details on the exact methods used please refer to the 2005 team report.

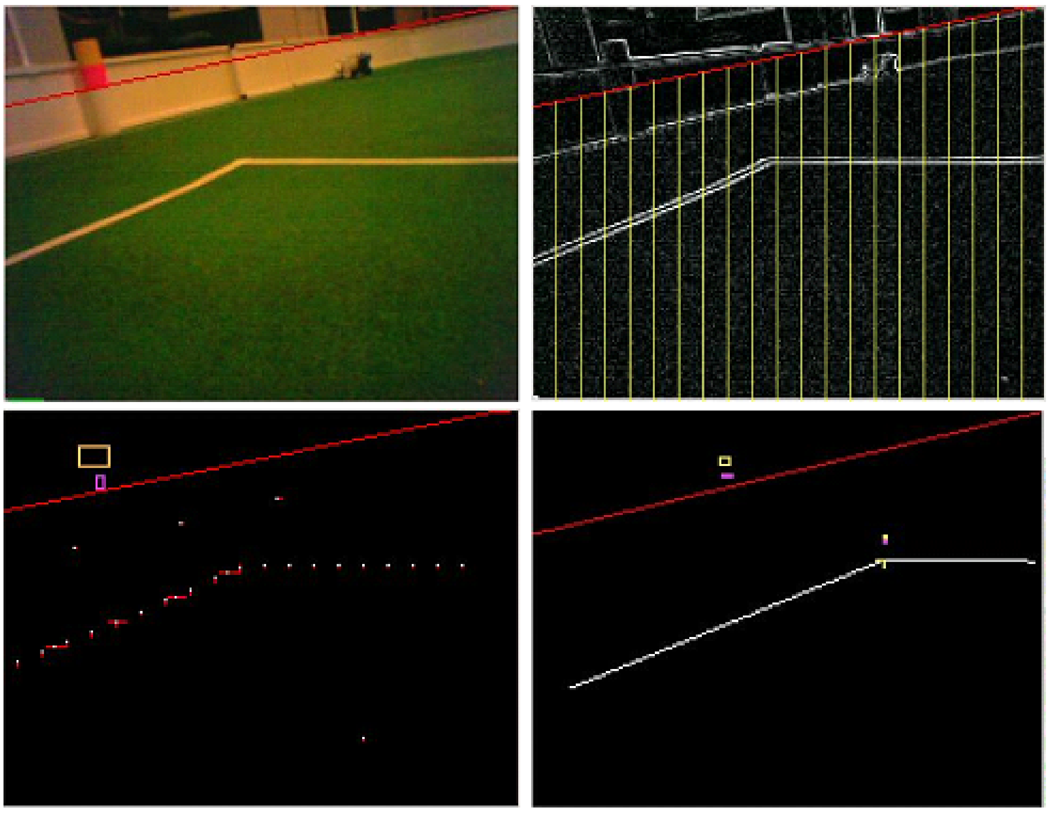


Figure 17: Line detection steps. (top left) original image. (top right)Edge Classified image with vertical scan lines. (bottom left) the found line points. (bottom right) Final lines with the corner uniquely identified.

3 Localisation

For the 2006 competition, there were a number of 'fixes' applied to previous code based on the Extended Kalman Filter (EKF), together with a small number of significant changes. These were of primary benefit in two main areas: *Goal Keeper Position* and *ball handling and grabbing*. Here, we briefly describe these changes.

3.1 Alternate Ball Measurement Coordinates

The most natural measurement coordinates for the ball are those returned by the vision system, namely, range or distance to the ball, d_b and (relative) bearing to the ball, θ_b. These measurements are generally very useful, and have been directly applied to the EKF in the past.

However, during testing, it was noticed that in the case where the ball was relatively close to the robot, that in some cases (particularly for example when the robot has grabbed the ball, moved with the ball, then kicked the ball) the EKF showed very poor transient behaviour.

3.1.1 Explanation

We believe that the reason for this is that the approximations inherent in the EKF, in particular those associated with the Jacobians of the measurement function, can in some cases have substantial errors associated with them. For example, the distance to the ball can be expressed as:

$$d_b = \sqrt{(y_b - y)^2 + (x_b - x)^2} \tag{1}$$

where (x, y) and (x_b, y_b) are the Cartesian coordinates of the robot and ball respectively. Taking partial derivatives of (1) gives:

$$\begin{bmatrix} \frac{\partial d_b}{\partial x} \\ \frac{\partial d_b}{\partial y} \\ \frac{\partial d_b}{\partial x_b} \\ \frac{\partial d_b}{\partial y_b} \end{bmatrix} = \begin{bmatrix} \frac{x-x_b}{d_b} \\ \frac{y-y_b}{d_b} \\ \frac{x_b-x}{d_b} \\ \frac{y_b-y}{d_b} \end{bmatrix} \tag{2}$$

In situations where the uncertainty in the distance to the ball is a significant fraction of the distance itself (which is a very likely situation if the ball is close), the relative errors inherent in using the Jacobian, (2) can be come very large.

Note that very similar considerations apply to the angle to the ball, where for cases where the uncertainty in the range is a small percentage of the range, the Jacobian linearization is accurate, but in other cases, can prove to have large errors.

3.1.2 Solution

One possibility to solve this problem is to use the ‘unscented’[sic] Kalaman Filter, as for example described in for example [Wan and Merwe, 2000], since this avoids the use of Jacobians in implementing the conditional probability update equations. However, the Unscented Kalman Filter requires computation of the square root of the state covariance matrix, which in our case is a 7x7 matrix. This calculation can be avoided if all computations are performed in factorised forms, such as UDU factorisation, however this was felt to be a significant code change with unknown impact on CPU requirements.

Alternatively, we decided that when the range to the ball was small (in our case, we used a threshold of 50cm) to transform the measurement information into the cartesian coordinates of the ball relative to the robot's current location and heading. In this case, the relative ball x[1] coordinate x_b^r can be calculated immediately from the measured range and bearing as:

$$x_b^r = d_b \times \cos\theta_b \tag{3}$$

We then treat x_b^r (and similarly the Y coordinate of the relative ball location, y_b^r, as measurement variables available for the EKF. In this case, the nonlinear equation relating the ball relative x is x_b^r is:

$$x_b^r = (x_b - x)\cos\theta + (y_b - y)\sin\theta \tag{4}$$

where θ is the robot heading. In this case, it can be seen that the relevant partial derivatives are given by:

$$\begin{bmatrix} \frac{\partial x_b^r}{\partial x} \\ \frac{\partial x_b^r}{\partial y} \\ \frac{\partial x_b^r}{\partial x_b} \\ \frac{\partial x_b^r}{\partial y_b} \\ \frac{\partial x_b^r}{\partial \theta} \end{bmatrix} = \begin{bmatrix} -\cos\theta \\ -\sin\theta \\ \cos\theta \\ \sin\theta \\ -(x_b - x)\sin\theta + (y_b - y)\cos\theta \end{bmatrix} \tag{5}$$

(5) shows clearly that there is no singularity in this form of the measurement equations, and therefore, particularly for small ball distances, we expect much better performance using this representation. In addition, in terms of the measurement variances and correlations, our experience is that whilst at longer ranges, the angle variance is effectively much smaller than the range variance, at shorter ranges, this effect is much less pronounced, and it seems quite acceptable to use identical measurement variances for x_b^r and y_b^r, and assume that these measurements are approximately independent of one another.The results of using this algorithm are shown in figure 18.

[1]The actual code uses a different coordinate convention to that described in (3)

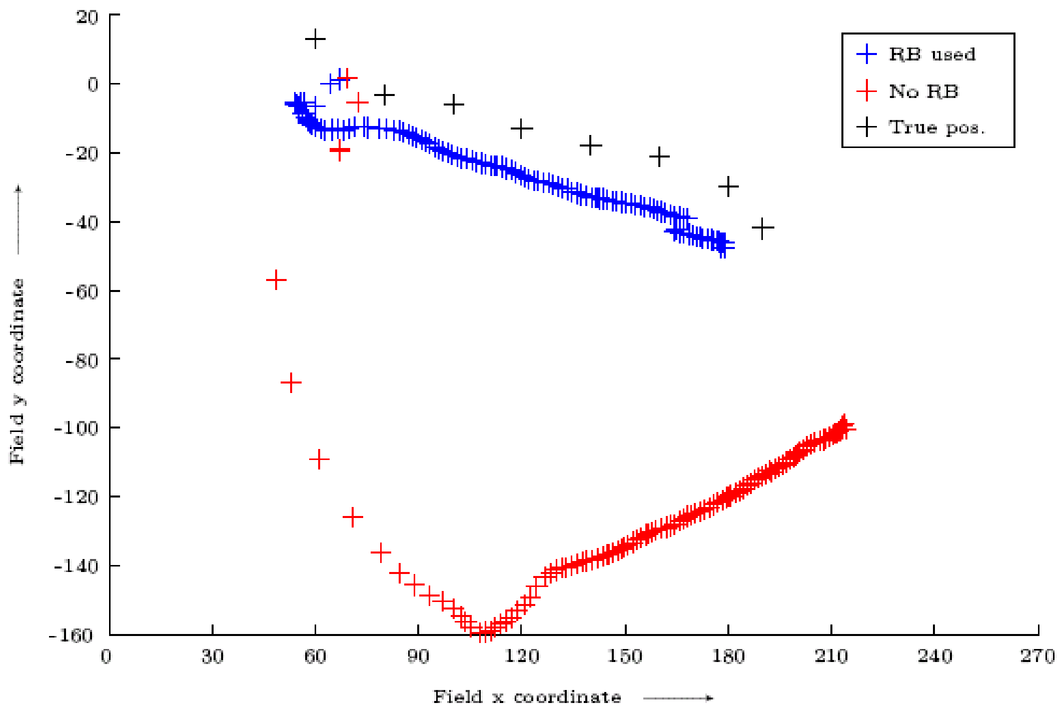


Figure 18: Comparison of ball localisation with and without using the relative ball coordinate system. Truth data obtained manually from video tape.

3.2 Incorporation and Tuning of Deadzones

The information content available to the EKF is very variable in it's quality and 'richness'. When there is insufficient information available from which to determine the robot's own location accurately, or more importantly, when the information available is almost (but not quite) ambiguous, then we have noticed problems with slow estimate 'drift' with time. For example, we noticed that when in the set state (just prior to kick-off), the head is typically fixed, staring at the ball, and it is quite possible for the robot to be only able to observe one or two objects on the field, and these from some distance away.

This resulted in the position estimates for the robot drifting slowly away from the correct position. This 'drift' is directly analogous to a problem studied some time ago in parameter estimation schemes in adaptive control (see for example [Peterson and Narendra, 1982]). One of the schemes devised for countering this drift was to use a 'deadzone' [Peterson and Narendra, 1982], [Middleton *et al.*, 1988] in the estimation part of the adaptive controller. We have therefore implemented this in the EKF with code as follows:

```
void KF::Deadzone(double* R, double* innovation, double CPC,
double eps)
  double invR;
  if ((eps<1.0e-08) || (CPC<1.0e-08) || (*R<1e-08))
   return;
if (ABS(*innovation)>eps)
    invR=(ABS(*innovation)/eps-1)/CPC;
   else
    *innovation=0.0;
    invR=0.25/(eps*eps)-1.0/CPC;

  if (invR<1.0e-08)
    invR=1e-08;
  if ( *R < 1.0/invR )
    *R=1.0/invR;
```

This function takes as inputs the current value of the measurement covariance, R, the innovations *innovation* (that is, deviation of the predicted measurement from the observed measurement), the variance of the predicted measurement CPC and the size of the deadzone, *eps*. The algorithm works by increasing the measurement covariance, and in some cases adjusting the innovations used in the actual EKF update. The rules used ensure that if the innovations are smaller than the deadzone, then the EKF gives zero change to the location estimates (by setting the innovations to zero) and also adjusting the R value used by the EKF.

3.3 Implementation of Penalty Corner Objects

The corner points of the penalty box have been identified by new vision processing software. As a first step towards fulling incorporating these objects into localisation, they were used for goal keeper localisation in our 2006 code. Penalty corners were only used when a valid background beacon could simultaneously be seen to give a clear indication of which corner was being observed. Incorporating these into keeper localisation proved to be a substantial benefit to the quality of localisation observed.

3.4 Other minor changes

There were a number of changes to some of the constants used by the EKF:

- Increase in the threshold for treating range measurements as outliers;
- Introduced an algorithm to ignore range measurements to goal objects or posts that are much larger than predicted. This proved particularly helpful for the goal keeper where occasional small size goals are recognised from within view of the keeper's own goal.
- Debugged the use of a sanity value to passed from vision, to alter the R value for objects where there is some loss of confidence in the measurement accuracy. Specifically, if a goal post object is not contained entirely within the image, the uncertainty in the range to the goal post is increased significantly.
- We retuned the checks for EKF reset ('kidnapped robot') to make this occur less often.
- We removed 'clipping' of the ball location, so that it is possible for the ball to be outside the field.

The net result was that the ball velocity estimates were more accurate and consistent; we believe we had a significant reduction in occurrences of a 'lost' goal keeper; and, our field players mostly seemed to have sufficient accuracy in localisation to make good behaviour decisions.

4 Team Behaviour

Every year the importance of team behaviour is growing, this is partly due to increased parity in the low level skills. In 2006 we actually made a substantial change to our team behaviour system, although from the outside looking in it might have been difficult to observe this change. RoboCup 2006 saw the introduction of a dynamic team behaviour system [Quinlan and Chalup, 2006], in theory allowing our robots to adapt during a game. However, the rules were constructed in a conservative manner (only making changes when losing) and as a consequence we saw no evidence of strategy changing during RoboCup.

It is also important to remember that the NUbots prefer to play a style of game that relies on dominating both ball possession and field position. For that reason the behaviours do not rely on outright speed or power, rather they rely everything combining well and producing a relatively complete style of play.

4.1 Positions

The NUbot system defines eight basic positions that the robots can play. These being, *Goal Keeper (GK), Sweeper (SW), Left Back (LB), Centre Back (CB), Right Back (RB), Left Forward (LF), Centre Forward (CF)* and *Right Forward (RF).*

The rules define that only one robot can be a goal keeper (this robot must be identified at the beginning of the game), while the other three robots (to be referred to as *field players*) are free to roam the field. Unlike other implementations [Veloso *et al.*, 2004] the NUbots positions **do not** define roles, for example the robot chasing the ball can still be a sweeper. It is important to keep a distinction between positions and roles as they are not the same in our system. Each position $\mathcal{P}$ is assigned a set of coordinates on the field (x, y) in cm and a heading θ in degrees. These define the general area on the field in which the robot should be positioned:

$$\mathcal{P} = \langle x, y, \theta \rangle \in [-300, 300] \times [-200, 200] \times [-\pi, \pi]$$

For example the positions of Goal Keeper and Left Forward are defined as:

$$\mathcal{P}_{GK} = \langle -265.0, 0.0, 0.0 \rangle \,, \;\; \mathcal{P}_{LF} = \langle 150.0, 80.0, 0.0 \rangle$$

4.2 Formations and High-level strategies

A formation $\mathcal{F}$ is a vector that contains four positions. When playing a game each robot in the team is instructed to play one of these four positions. A high-level strategy $\mathcal{S}$ contains two formations, in general these are a defensive $\mathcal{D}$ and an offensive $\mathcal{O}$ formation:

$$\begin{aligned} \mathcal{F} &= \langle \mathcal{P}_a, \mathcal{P}_b, \mathcal{P}_c, \mathcal{P}_d \rangle \,, \\ \mathcal{S} &= \langle \mathcal{F}_{off}, \mathcal{F}_{def} \rangle \equiv \langle \mathcal{D}, \mathcal{O} \rangle \end{aligned}$$

The current NUbot system contains six standard formations that combine to form six high-level strategies:

$$\begin{aligned}
\mathcal{S}_{normal} &= \langle \mathcal{D}_{normal}, \mathcal{O}_{normal} \rangle \\
\mathcal{S}_{sweeper} &= \langle \mathcal{D}_{sweeper}, \mathcal{O}_{sweeper} \rangle \\
\mathcal{S}_{aggressive} &= \langle \mathcal{D}_{aggressive}, \mathcal{O}_{aggressive} \rangle \\
\mathcal{S}_{offensive} &= \langle \mathcal{D}_{sweeper}, \mathcal{O}_{aggressive} \rangle \\
\mathcal{S}_{defensive} &= \langle \mathcal{D}_{aggressive}, \mathcal{O}_{normal} \rangle \\
\mathcal{S}_{hold} &= \langle \mathcal{D}_{aggressive}, \mathcal{D}_{aggressive} \rangle
\end{aligned}$$

where

$$\begin{aligned}
\mathcal{D}_{normal} &= \langle \mathcal{P}_{GK}, \mathcal{P}_{LB}, \mathcal{P}_{RB}, \mathcal{P}_{CF} \rangle, \mathcal{O}_{normal} &= \langle \mathcal{P}_{GK}, \mathcal{P}_{CB}, \mathcal{P}_{LF}, \mathcal{P}_{RF} \rangle \\
\mathcal{D}_{sweeper} &= \langle \mathcal{P}_{GK}, \mathcal{P}_{SW}, \mathcal{P}_{CB}, \mathcal{P}_{CF} \rangle, \mathcal{O}_{sweeper} &= \langle \mathcal{P}_{GK}, \mathcal{P}_{SW}, \mathcal{P}_{LF}, \mathcal{P}_{RF} \rangle \\
\mathcal{D}_{aggressive} &= \langle \mathcal{P}_{GK}, \mathcal{P}_{CB}, \mathcal{P}_{LB}, \mathcal{P}_{RB} \rangle, \mathcal{O}_{aggressive} &= \langle \mathcal{P}_{GK}, \mathcal{P}_{CF}, \mathcal{P}_{LF}, \mathcal{P}_{RF} \rangle
\end{aligned}$$

Using the described framework a new position, formation or strategy can be added in a few lines of code. It is also possible for a $\mathcal{S}$ to contain more then two formations, for example $\mathcal{S} \cong \langle \mathcal{F}_{off}, \mathcal{F}_{midfield}, \mathcal{F}_{def} \rangle$.

4.3 Roles and Role Allocation

In the NUbot system there are two distinct roles, `chasing` and `positioning`. If a robot is not `chasing` then it must be `positioning`. The ideal situation has exactly one robot per team in the `chasing` role.

$$\texttt{chasing} \cap \texttt{positioning} = \emptyset, \ |\texttt{chasing}| = 1$$

Role allocation is done without negotiation, that is, each robot makes a decision on the information it currently has and hopes that this is the same decision the other robots have made. A single variable, *bvrMessage*, is used to make the role decision, this variable contains a float that represents a modified distance from the robot to the ball [Quinlan *et al.*, 2005]. The `chasing` robot is defined as the robot with the lowest *bvrMessage*. The variable also encodes additional information such as attempting a grab, dribbling and kicking. This extra information allows the `positioning` robots to perform different actions depending on what the `chasing` robot is actually doing. For example, the robots should position differently when the `chasing` robot has gained control of the ball and is dribbling.

It is possible that none or multiple robots might momentarily believe they are `chasing`, but in practice the noise and the dynamic nature of the environment means the problem rarely occurs and it is fixed extremely quickly (fractions of a second). An additional problem occurs when a robot loses sight of the ball at close range. In this case it is likely that a second robot will rapidly enter the fray, this may cause collisions until the non-chasing robot is able to move back to its assigned position. This issue is something of a trade-off, in that a less aggressive strategy (i.e. introducing a delay of $\mathcal{S}$ seconds) would prevent the congestion but it could lead to extended periods of time where no robot is moving towards the ball. A general version of the rule would be:

if $delay > \mathcal{S}$ **then** `chasing` $= MIN(bvrMessage)$

Historically the NUbots have preferred to have a `chasing` robot at all times, $\mathcal{S} < \frac{1}{10}$ second.

4.4 High-Level Strategy Switching

Currently the rules for automatic strategy switching are fairly simple. This is primarily due to the lack of information available to the robot. In 2006 the robots relied solely on the current score and the time remaining in the match; the assumption being that the score is indicative of the effect of a strategy.

Prior to the match the coaches (humans) select a strategy ($\mathcal{S}_{current}$). A hierarchy of rules is then used to change this strategy, for example a rule taken directly from our 2006 RoboCup code (in the Python language) was

```
if (secondHalf and (oppScore > ownScore)):
  scoreDiff = oppScore - ownScore
  if (timeLeft/60.0 < scoreDiff):
    newStrategy = AGGRESSIVE
```

For the purpose of this report we will not discuss the entire rule set, rather emphasise that rules should remain flexible. Our system was designed to make it easy to modify or create new rules for different situations and opponents. It was not designed to completely remove the human element from the strategy decision. It differs from the work of McMillen et al [McMillen and Veloso, 2006] in which the strategies are cycled until a good strategy is found. While this should result in the discovery of a good strategy there exists the possibility of conceding a goal (or two) while rotating through inferior strategies.

4.5 Position Switching

In real football each player is physically different and has different skills and for these reasons they are suited to play particular positions. For all practical purposes each AIBO is identical, and hence the position switching algorithm does not need to consider the attributes of each individual robot. This allows us to base our position switching purely on the physical locations of the robots and the position of the ball on the field.

Again no form of negotiation is used, instead the robots rely on the shared world model being consistent between each other. The system essentially relies on the fact that each robot is capable of replicating the decision of the other robots, and thus the team should function as one.

The first step of the position switching algorithm is to select the relevant formation from the current strategy. Currently this is done by determining a position on the field in which the team should be offensive or defensive. For RoboCup 2006 the following rule was used:

$$\mathcal{F}_{current} = \begin{cases} \mathcal{D}_{current} & \text{ball in defensive half} \\ \mathcal{O}_{current} & \text{ball in offensive half} \end{cases}$$

In this version of the rule only the location of the ball determines offensive or defensive. This could be replaced by a more complicated rule involving possession. For example, if the ball is the defensive half but in the possession of a robot from your team then you should begin a transition to offence.

The goal keeper can not be switched so the algorithm need only consider the positions for the three field players. This is done by using the following rules:

- The `chasing` robot is assigned the position closest to the ball. Note: The goal keeper will never switch positions.

- The `positioning` robots are then free to play any of the remaining positions. This decision is made by minimising the total distance that these robots would need to travel.

4.6 Goal Keeper

For 2006 we introduced an entirely new goal keeper. We are the first to admit that our keeper was 'rusty' during the early stages of RoboCup but the keeper performed extremely well in the final. Our goal keeper gives the impression of being over-exitable and over-aggressive, while this is true (in comparison to other teams) we don't believe it is a problem. Rather we like the fact that our keeper will contest balls and tackle opponents.

When building the keeper we had a few key points:

Positioning - The robot should be positioned in manner to reduce the likelihood of a goal being shot past the keeper.

Diving - The robot should dive and stop any shot on goal, however if the ball is coming at its chest, it should catch the ball.

Clearing the ball - When given the chance to robot will attempt to clear the ball from the box.

In truth we have no real secrets on how we achieved these points, rather we emphasise that a lot of the work done in localisation this year was due to problems found while developing the keeper. In particular the quality of the information obtained from the goals is bad when very close (i.e our own goal), and the distance from the far beacons is often noisy. Unfortunately these are the problems most often faced by the goal keeper and therefore the were the ones we chose to work on.

5 Individual Behaviours

As stated in last year's report, we believe that the individual skills of the robots are arguably the most critical element of the game play. The importance of developing and fine tuning these skills should not be underestimated.

The key changes in 2006 revolved around better information, in particular a more accurate estimation in ball position and velocity. The outcome being a more polished and accurate set of skills, which led to less miss-grabs and higher percentage of shots on target.

The three subsections are 'Chasing','Grabbing' and 'Dribbling'

5.1 Chasing

The NUbots have always favoured a possession style of football. As a result a lot of attention is placed into to developing code for chasing and grabbing the ball.

It turns out that good quality chasing and grabbing relies on a high performance of all other modules. Accurate ball distances are required from vision, correct ball velocities from localisation and precise movement and odometry calibration from locomotion.

Our chase for 2006 was essentially an updated version of the 2005 chase. The main task undertaken during the year was 'balancing' the reliance of velocity.

The basic chase code forces the robot to take the largest possible step which will position the ball on the chest. In 2006 we have added some parameters that define the usage of velocity, these can be modified in real-time. The parameter CHASEVELDIV defines how many frames ahead we predict, a value of 30 = 1 vision frame while a value of 1 = 30 vision frames (i.e. 1 second ahead).

Firstly, calculate the relative location of the ball -

```
vX = ball.vX
vY = ball.vY
if (not Strategy.CHASEVELOCITY):
  vX = 0
  vY = 0
relX = fabs(distance) * sin(bearing)
relY = fabs(distance) * cos(bearing)
tempX = relX + (vX/CHASEVELDIV)
tempY = relY + (vY/CHASEVELDIV)
```

We then translate the position into step parameters -

```
strideLength = tempY*10.0
strafe = CROP(tempX*10.0,-50,50)  # Don't strafe more than 5cm
rearStrideLength = CROP(strideLength,-1*MaxChaseLength, MaxChaseLength)
```

```
  frontStrideLength = rearStrideLength*ChaseFMult
  maximumTurn = 0
  sqr = sqrt(pow(rearStrideLength,2)/MaxChaseLength + pow(strafe,2)/50.0);
  if (sqr < 50): maximumTurn = 50 - sqr
  absMaximumTurn = fabs(maximumTurn)
  turn = RAD_T_DEG*(ballHeading)
  turn = CROP(turn, -absMaximumTurn,absMaximumTurn)
```

The ability to go backwards or take small strafe steps was based around a simple idea - “If the ball is reasonably close and not in-line with the chest, then quickly adjust until the robot can continue its normal chase”. Obviously the actual numbers need to be tuning based on the performance of the vision,localisation and locomotion modules. It is important to note that the existence of these steps is all but hidden from a human spectator. The steps are subtle in nature and often less then one full step is required to realign the chase.

5.1.1 ‘Paw-Dribble’

The intention of the this code was to avoid disallowed goals for dribbling the ball over the goal line. We did not have this turned on during the practice games but we after having a few goals disallowed we gradually increased its usage.

The logic for deciding when to ram was essentially “Am I near the goal line, am I roughly facing the goal and have I seen the goal in the last 5 frame)”, in code it became -

```
if (ball.x > 200 and fabs(ball.y) < 38 and
RAD_T_DEG*(goal.orientation) < 10 and (Common.frame-lastGoal.seen) < 5):
    Ramming = True
```

The robot would then select the best paw to use and position its body in such a way to hit the ball with the paw.

5.2 Grabbing

Grabbing can be considered as a form of the *chicken and egg dilemma*, in the respect that to play a ‘grab and move’ style requires good grabbing, but good grabbing is only important in a ‘grab and move’ style.

Ever since 2002 we have employed a ‘grab and move’ style, so for our team the grab is extremely critical. Efficient grabbing relies on everything working well, small errors in any module will severely effect performance. Any changes made in other modules affect the grab.

This year we greatly increased the reliance of ball velocity, and again we solely used the shared ball for grabbing. Apart from the increased reliability of ball velocity the biggest improvements were: moving while grabbing and smoothly widening the legs when required.

For RoboCup an immense amount of time and effort is placed into developing and maintaining the grab code.

5.2.1 Grab Trigger

The point where a robot needs to trigger a grab is on a never ending cycle of adjustment and tuning. In 2006 we continued the use of sanity factors. In this method multiple grab triggers can exist with a weighting applied to each one. Our final version relies upon the relative location of the ball, the velocity of the ball and for the first time the standard deviation in the ball location.

We now add a number to the relative ball location if we are uncertain of the location of the ball. The idea was to prevent false grabbing which was occurring because of bad ball measurements, it was found that the robot was often aware of a possible problem (i.e. high uncertainty).

```
newRelY = relY + (velY/10.0)
newRelX = relX + (velX/10.0)

xExtra = sdBallVarX/10.0
yExtra = 0
if (sdBallVarY > 40):
  yExtra = 3
  newRelY = relY
elif (sdBallVarY > 30):
  yExtra = 2
  newRelY = relY
elif (sdBallVarY > 20):
    yExtra = 1

  if (fabs(relX) < 4 and fabs(newRelX+xExtra) < 3.0
and (newRelY+yExtra) < 15.0):
   grab = True
  if (velY < -15 and fabs(newRelX) < 4 and newRelY < fabs(velY)):
   catch = True
    grab = True
```

When grab is true the robot will perform the grabbing motion. If catch is also true the robot will perform the catching motion.

5.2.2 Grab Motion

This year's grab motion relied heavily on localisation, we would walk in the direction of the shared ball while doing the grab. The benefit being that grab motion itself would try and cover up an error in the grab trigger, it also allowed us to handle balls that were rolling away or towards the robot.

The 2005 and 2006 our grab remained was quite fast, the entire motion could be completed in 10 vision frames ($\leq 1/3$ of a second). However, in 2006 we allowed this number to increase when requried, for example if we were 'moving backwards in chase' then the motion required extra time ($\approx$20 frames).

During the grab the head would first be lifted, allowing the ball to slide under and then the head would be dropped over the ball in conjunction with the mouth being opened. While the head is moving the front legs are moved slightly forwards and outwards to surround the ball. The basics of this code have remained unchanged from last year, see [Quinlan *et al.*, 2005] for the actual python code.

5.2.3 Catch Motion

The catching motion is actually a modification of the grab motion. It also allows the robot to move towards the expected location of the ball but it dramatically changes the location of the legs. Both front legs are now moved 2cm forwards and outwards. Also note that the execution time (represented by *mult*) has been increased to 12-13 frames.

```
if (catch or (velY < -15 and newRelY < fabs(velY))):
  extraWidthLeft = 20
  extraLengthLeft = 20
  extraWidthRight = 20
  extraLengthRight = 20
  mult = 1.25
  minWalk = 20
  newRelY = relY+(velY/10.0)
  newRelX = relX+(velX/10.0)
```

5.3 Dribbling

On of the key features of our play was the omni-direction dribble employed. This dribble was primarily used to line up the goal or to avoid other robots. It should be noted that many of the 'nice' features of our dribbling (i.e. dodging robots, kicking through gaps etc) are not the result of overly advanced vision but rather a combination of simple but deliberate rules (albeit these rules required a lot of effort to create and tune). In testing we have seen over 20 state (rule) changes in 40 dribble frames, this is how our attackers appear to perform highly complex sequences of motions.

The stance used for the chase had to satisfy multiple conditions -

Omni-directional - For optimal use the stance should give full omni-directional movement, this allows the robot to strafe around robots, dribble between opponents or back the ball off sidelines.

Control the ball - Obviously the robot should never drop the ball but finding the right combination between omni-directionality and holding was difficult. We found the optimal solution was to have a 'loose' hold on the ball, that is the ball could bobble between the legs and the head but could never pop out.

Vision Distance - In our system the robot could see goals from over half the field while dribbling the ball. This enabled us to line up long range shots or avoid distant robots.

Walk optimisation was then run to improve the speed of the dribble. In these experiments the robot ran between two pink balls while dribbling the orange ball.

Some parameters such as forntForwardOffset and frontSideOffset were constrained to make sure the ball was held and the final parameters allowed the robot to dribble at approximately 35cm/s

An internal timer was used to monitor the dribbling time, it forced the robot to cancel before it got close to the 3 second limit.

The dribble logic can be summarised as *Check for ball*, *dribble away from sidelines*,*dodge an opponent* and then *line up the goal*.

In 2006 the key upgrades in the dribbling code was actually two vision 'fixes' that we thought we had working in 2005. These were 'goal gap detection' and 'narrow goals'. These fixes instantly allowed our current dribbling code to target goals much more efficiently (reducing the number of off-target shots).

5.3.1 Lining up to shoot at goal

If the goal is visible then turning towards it is relatively straight forward, if the goal is not visible then one must rely on localisation. The important part is that you can construct a more reliable rule for facing the opponent goal then simply turning towards the opponent goal. At first glance this statement may not make much sense, but it is often better to turn away from your own goal. In most situations turning away from your own goal is equivalent to turning towards the opponent goal, but cases do exist where they are not one and the same. Our logic for choosing the turn direction is shown below -

```
angleToTurn = -ownGoalAngle
# Own goal and oppoent goal are in the same direction
if (fabs(ownGoalAngle) > 0 and fabs(opponentGoalAngle) > 0):
  if (me.x > -100): # If attacking .. turn towards opponent goal
    angle = opponentGoalAngle
  else: # If defending .. turn away from own goal
    angle = -ownGoalAngle
```

Our robots will then turn until they see the goal or expect to see the goal based on localisation. If the goal is seen then they line up to the goal gap not the actual goal.

Once lined up we have two main options, the robot can kick the ball or the robot can dribble the ball to a better position to kick.

When attacking we tend to dribble the ball a bit longer in an attempt to further line up the goal. If the robot is in one of the attacking corners, then it will dribble towards the goal but also veers towards the centre of the field. The veer opens up the face of the goal, therefore creating a better target to shoot at.

5.3.2 Obstacle Avoidance - "Infrared"

At RoboCup 2006 we actually removed the vision based avoidance used in 2005 as it was noticed that the majority of avoidance was actually done using the infrared sensors. The infrared also has the added benefit of avoiding any obstacle, i.e. the goal post or a referees leg.

The first important step is tuning the infrared sensor value to a value that indicates an obstruction, the stance of the robot effects this number, ideally you want to maximise this value so that it does not cause the robot to avoid the ground (i.e. at what distance does the infrared cross the ground). We made the robot dribble with the ball and monitored the values of the IR sensors to discover this number. It turns out the sensors vary considerably between robots, so we had to customise this for each robot.

Deciding on the direction to avoid is hard since the IR sensors only give you a distance. In 2006 we made the decision to always move towards the goal, this has the advantage of the robot never running outside the field and you should increase your likelihood of scoring.

The IR sensor avoidance produced the unexpected positive side effect of the robot avoiding the goal post. This allowed our attackers to dodge goal posts and end up inside the goal.

5.3.3 Avoiding sidelines

The key advantage of the omni-directional dribble was the ability for our robots to back the ball off any field lines. We achieved this using our potential field system, the repulsiveness of the lines is turned up hence driving our robots away from the edges of the field. Using the potential field allows the robot to move smoothly, i.e in the corners of the field the robot will move diagonally out of the corner.

We did have to sacrifice some vision distance when moving backwards as the head needed to be placed further down to hold the ball.

5.4 Kick Selection

The effectiveness of our dribbling code simplified our kick selection logic. At RoboCup we actually used 5 types of kicks, 3 of which went forwards at varying lengths, one that went backwards and one that went at 90 degrees.

Typically we tried to keep the ball in play at all times, hence we only used our most powerful kick in two cases: in the back of the defensive half and clearing to the front half or if the robot is lined up to the goal (thus it is a shot on goal).

We used our second most powerful forwards kick (sideswipe with the head that went forwards, which was actually a walk) when we were roughly facing forwards but not going to score a goal. This kick was used to progress the ball forwards and remain semi-in-control of the ball.

The third forwards kick was more of a ‘drop’, it was used when we simply wanted to release the ball a small distance in front of us.

We used ‘slap’ kicks to quickly clear balls from the back of the field. The kicks do not need a grab and can move the ball most of the field at a 90 degree angle. We had the option to turn these kicks on in the offensive half but this was not used at RoboCup.

We also have countless other kicks ($\approx$30), but they all have advantages and disadvantages. The fact that they often do not work on every robot led us to use only a small percentage of them.

6 Locomotion

The principle of the walk engine is the same as in previous years, that is an omni-directional parameterised walk. We will refer you to our previous reports [Bunting *et al.*, 2003; Quinlan *et al.*, 2004; 2005] and the original omni-directional walk paper [Hengst *et al.*, 2002] or the thesis [Quinlan, 2006] for more information.

The single biggest change in the locomotion this year was removing constraints (such as maximum combination of forwards and strafe) that were placed on the walk in the previous year. In 2006 we found out that these constraints were no longer required (they were used in 2003,2004) and removing them greatly increased the omni-directional speed of the robots (i.e. moving backwards, or diagonally).

6.1 Walk Engine

The walk used at RoboCup 2006 was approximately 42 cm/s in a straight line. It is important to note that we have generated many different walks with similar speeds (or faster speeds), but this one was chosen because of its 'flat' body and its apparent power (ability to move while being bumped). It is our opinion that Four-Legged League may have reached the stage that pure speed is no longer the critical feature.

6.2 Head Control

Head control can be broken down into two areas: General movements and Tracking an object.

6.2.1 General movements

General movements include searching for balls, panning for beacons and any other action that does not actually involve reacting to vision. The movements are generally straight forward but we have found that two common traps occur:

Speed Often the head is told to move too fast, this blurs the image. Additonally you need to match the head speed with the body speed. For example, if you are searching for a ball by spinning on the spot, the head movement must be fast enough that can not turn past the ball before the head travels its full path (no blind spots).

Head height By carefully setting the height of your head you can greatly decrease the amount of noise introduced into vision. For example, you won't see people wearing orange shirts if you don't look up into the crowd. We spend time making sure our head movements are as low as possible (yet high enough to see the beacons and goals) during games.

6.2.2 Tracking an object

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:

$$(x_{\text{pan}}, y_{\text{tilt}}) = \underline{f}\left(\theta_{\text{small}}, \theta_{\text{big}}, \theta_{\text{pan}}\right) \tag{6}$$

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 (6). 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:

$$\begin{pmatrix} \delta x \\ \delta y \end{pmatrix} = \begin{pmatrix} 0 & \sin\left(\theta_{\text{pan}}\right) & 1 \\ 1 & \cos\left(\theta_{\text{pan}}\right) & 0 \end{pmatrix} \begin{pmatrix} \delta\theta_{\text{small}} \\ \delta\theta_{\text{big}} \\ \delta\theta_{\text{pan}} \end{pmatrix} \tag{7}$$

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

$$\mathbf{X} = \begin{pmatrix} 0 & \sin\left(\theta_{\text{pan}}\right) & 1 \\ 1 & \cos\left(\theta_{\text{pan}}\right) & 0 \end{pmatrix} \tag{8}$$

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

$$\mathbf{X}^+ = \mathbf{X}^T \left(\mathbf{X}\mathbf{X}^T\right)^{-1} \tag{9}$$

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

$$\begin{pmatrix} \delta\theta_{\text{small}} \\ \delta\theta_{\text{big}} \\ \delta\theta_{\text{pan}} \end{pmatrix} = \left(\mathbf{X}\mathbf{X}^T\right)^{-1} \mathbf{X} \begin{pmatrix} \delta x^* \\ \delta y^* \end{pmatrix} \tag{10}$$

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 (10) 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.

6.3 Forward Kinematics

This year forward kinematics were used for the elevation check and horizon line calculations, therefore there accuracy was heavily relied upon.

The mathematics for the forwards kinematics can be found in [Röfer *et al.*, 2004]. We felt some of the offsets used are debateable so we derived/measured these offset ourselves.

$$\textit{top length of the leg} = \sqrt{69.5^2 + 9^2}$$
$$\textit{bottom length of the front leg} = \sqrt{28.3^2 + 71.5^2}$$
$$\textit{bottom length of the rear leg} = \sqrt{21.3^2 + 76.5^2}$$
$$\textit{rotator offset} = 0.1286 \text{ radians}$$
$$\textit{front knee offset} = \text{-}0.1709 \text{ radians}$$

$$\textit{rear knee offset} = \text{-0.2606 radians}$$

The important part of the forwards kinematics is calculating the contact point on the ground. The nature of the Aibo walk means that the front paw does not touch the ground, but rather some point of the curved forearm touches the ground. Accurately determining this point is important when calculating the tilt and roll of the robot.

To simplify matters we did not implement the equation of the arc, but rather we calculated the height at certain intervals on the arc. The point with the lowest height (below the shoulder) is deemed to contact the ground.

The same process is repeated on the rear legs. In both cases the 'forearm' contact point is compared to the paw heigh, the lowest of the which is deemed to be the contact point (this takes care of the case where the paw is touching the ground).

7 Final Soccer Results

1. **NUbots (Australia)**
2. rUNSWift (Australia)
3. Microsoft Hellhounds (Germany)

Acknowledgements

The NUbots are grateful to all colleagues, friends, previous members, and other supporters of the team including Dianne Piefke, Nick Hawryluk 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 2006 was provided by the ARC Centre for Complex Dynamic Systems and Control (CDSC), BHP Billiton Innovation and a Research Infrastructure Block Grant 2005 and 2006 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

[Bunting *et al.*, 2003] 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. Available at http://robots.newcastle.edu.au/publications.html.

[Henderson, 2005] N Henderson. Digital Image Processing In Robot Vision. Technical report, University of Newcastle, 2005. Available at http://robots.newcastle.edu.au/publications.html.

[Hengst *et al.*, 2002] B. Hengst, S.B. Pham, D. Ibbotson, and C. Sammut. Omnidirectional Locomotion for Quadruped Robots. *RoboCup 2001: Robot Soccer World Cup V*, pages 368–373, 2002.

[Hong, 2005] K Hong. NUbots: Enhancements to Vision Processing, and Debugging Software for Robocup Soccer. Technical report, University of Newcastle, 2005. Available at http://robots.newcastle.edu.au/publications.html.

[McMillen and Veloso, 2006] Colin McMillen and Manuela Veloso. Distributed, Play-Based Role Assignment for Robot Teams in Dynamic Environments. In *DARS 2006*, July 2006.

[Middleton *et al.*, 1988] 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.

[Nicklin, 2005] S Nicklin. Object Recognition in Robotic Soccer. Technical report, University of Newcastle, 2005. Available at http://robots.newcastle.edu.au/publications.html.

[Peterson and Narendra, 1982] B. Peterson and K. Narendra. Bounded error adaptive control". *IEEE Transactions on Automatic Control*, 27(6):1161–1168, 1982.

[Quinlan and Chalup, 2006] Michael. J. Quinlan and S. K. Chalup. Impact of Tactical Variations in the RoboCup Four-Legged League. In *Procedings of International Symposium on Practical Cognitive Agents and Robots*. University of Western Australia Press, 2006.

[Quinlan *et al.*, 2004] M. Quinlan, C. Murch, T. Moore, R. Middelton, Li. Lee, R. King, and S. Chalup. The 2004 NUbots Team Report. Eecs tech report, University of Newcastle, 2004. Available at http://robots.newcastle.edu.au/publications.html.

[Quinlan *et al.*, 2005] M.J. Quinlan, S.P. Nicklin, K. Hong, N. Henderson, S.R. Young, T.G. Moore, R. Fisher, P. Douangboupha, S.K. Chalup, R.H. Middleton, and King. R. The 2005 NUbots Team Report. EECS technical report, University of Newcastle, 2005. Available at http://robots.newcastle.edu.au/publications.html.

[Quinlan, 2006] Michael J. Quinlan. *Machine Learning on AIBO Robots*. PhD thesis, The University of Newcastle, 2006.

[Röfer *et al.*, 2004] T. Röfer, T. Laue, H-D. Burkhard, J. Hoffmann, M. Jungel, D. Gohring, M. Lotzsch, U. Duffert, M. Spranger, . B. Altmeyer, V. Goetzke, O. v. Stryk, R. Brunn, M. Dassler, M. Kunz, M. Risler, M. Stelzer, D. Thomas, S. Uhrig, U. Schwiegelshohn, I. Dahm, M. Hebbel, Nistico W, C. Schumann, and M.Wacher. GermanTeam 2004. Technical report, 2004.

[Seysener *et al.*, 2004] 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.

[Seysener, 2003] C Seysener. Vision Processing for RoboCup 2003. Technical report, University of Newcastle, 2003.

[Veloso *et al.*, 2004] M. Veloso, P. Rybski, S. Chernova, D. Vail, S. Lenser, C. McMillen, J. Bruce, F. Tamburrino, J. Fasola, M. Carson, and A. Trevor. CMPack04: Team Report. Technical report, Carnegie Mellon University, 2004.

[Wan and Merwe, 2000] E.A. Wan and R. Van Der Merwe. The unscented Kalman filter for nonlinear estimation. In *Proc. of the IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium*, 2000.