Return of the NUbots!

The 2003 NUbots Team Report

Jared Bunting, Stephan Chalup, Michaela Freeston, Will McMahan, Rick Middleton, Craig Murch, Michael Quinlan, Christopher Seysener, Graham Shanks
2 October 2003
Newcastle Robotics Laboratory
The University of Newcastle
NSW 2308, Australia
http://www.robots.newcastle.edu.au

ABSTRACT

The NUbots repeated a strong performance in 2002 by again gaining 3rd place in 2003. This report describes the architecture and components of the NUbot software system, focusing primarily on the new features for the 2003 architecture. These features include substantial improvements in all the main modules: vision processing; localisation and world modelling; locomotion; and behaviour. In addition, our team used run time alterable configuration files, and wireless TCP/IP facilities for testing, enhancement and debugging purposes.

Contents

1 Introduction

The NUbots joined the SONY Legged League in 2002 and in less than four months designed and developed a competitive team of soccer playing robots that were placed third at RoboCup 2002 [6]. In the 2003 competition in Padova, the NUbots were again placed 3rd after narrowly being beaten in the semi-final on a penalty shoot out. In the preceding qualifying and quarter final games, the team had a perfect defensive record, and overall we achieved the best goal difference of any team in the competition.

In this paper we describe the overall architecture and the individual components of the NUbot software system. In the subsequent sections, we briefly give an overview of last years software, and focus primarily on the motivation for and new features of the 2003 architecture. Performance at the 2003 Robocup can be attributed to: (i) significant improvements in vision processing (with debugging, additional sanity checking and a number of novel features); (ii) an integrated and enhanced Extended Kalman Filter for localisation and world modelling module; (iii) a substantial increase in robot locomotion, both in terms of speed and agility; and (iv) a number of improvements to behaviours including kick selections, position negotiation using wireless, and ball 'saves'.

In Section 2 we describe the overall architecture of the NUbot system. Following this, each module and the interactions between them will be discussed in their individual sections. 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 new locomotion and behaviour modules in Sections 5 and 6 respectively.

2 The System Architecture

2.1 2002 Architecture

Last year's architecture relied heavily on OPEN-R objects. The main functions, namely vision processing, localisation and world modelling, behaviour control, and locomotion, were each assigned a separate object and therefore ran in their own process space. In other words, we adopted a multi-object architecture rather than a single monolithic object architecture.

Object	Description
NUICC	Robot motor initialization and robot meta-state
NUVIS	Vision processing
NULOCWM	World modelling and localisation
NUBVR	Behaviour Control
NUPWALK	A thin wrapper for PWalk with additional soccer ball kicks
NUWAV	Plays Wave files

Table 2.1: Objects in the 2002 NUbots system

2.2 2003 Architecture

On further reflection after the 2002 competition, the multiple object/task[1] arrangement for 2002 was arguably unnecessary. It turns out that with the possible exception of NUWAV[2], each of the objects behave synchronously and with clearly defined dependencies. Because of this straightforward real time dependency, multi-tasking does not seem to offer any real advantages. In addition, the overhead involved in setting up and modifying inter-object communication was a significant barrier to rapid development. Furthermore, there is limited detailed information available on the scheduling system for concurrent processes in OPEN-R, which further acts as a disincentive to use multi-tasking.

With these arguments in mind, we merged all OPEN-R objects into a single, monolithic binary called NUBOT. During this process, large portions of code were rewritten or removed entirely. The main modules remained conceptually separated, although they are now contained in a single OPEN-R binary.

One caveat to keep in mind with our revised architecture, however, is that timing over-runs in any module will impact all modules. For example, 'dropping' vision frames has a serious detrimental effect on the performance of the locomotion engine. However, we adopted the philosophy that reliable timing of (say) the vision module is in any case an important feature, which we were able to achieve. We therefore do not consider this to be a serious restriction imposed by the architecture.

In essence the high-level architecture of the NUbots 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 2.1 below.

[1] Note that in OPEN-R, each object acts essentially as a separate task.
[2] NUWAV was required in 2002 for the challenges. However, in 2003 this module was not required.

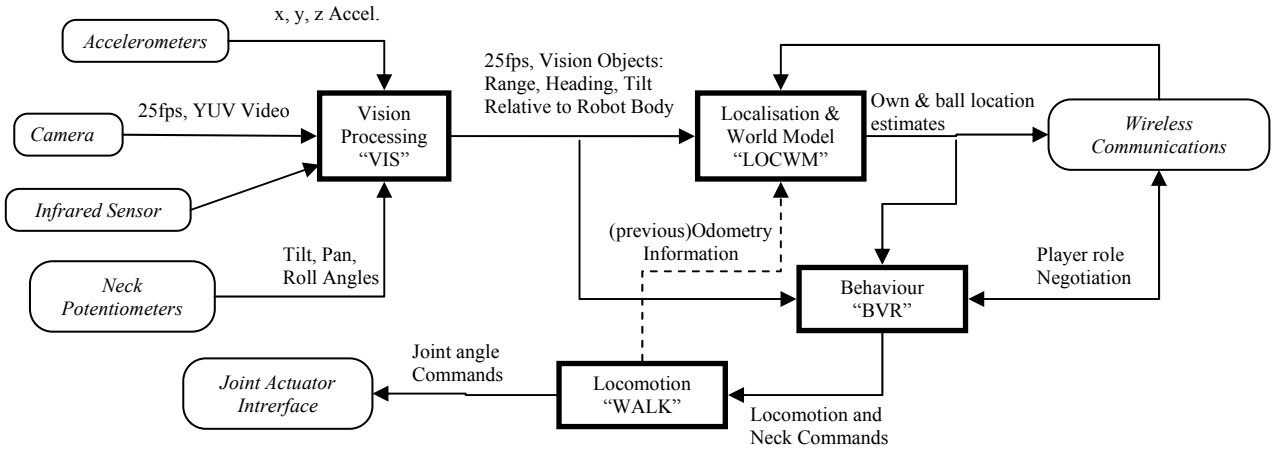


Figure 2.1: Main Software Structures Block Diagram

In addition to the main software modules above, there are two other structures used in software development. These are the incorporation of a configuration file structure, and the addition of debug support using the wireless network.

2.3 Configuration Files

Last year, development was slowed by the necessity of a recompile whenever a constant needed to be modified. This had a negative impact on the tuning of many sections of code, with locomotion being the most obvious.

Configuration files allow for the modification of certain variables on the robot without the need for a recompile or an operating system reboot. This is accomplished using the debug tool illustrated below in Figure 2.2. As development continued, a large number of software variables became configuration options as the usefulness of runtime alteration became more apparent.

2.4 Wireless Network Debug Support

Wireless debugging was greatly improved by the newly built-in telnet function, thus eliminating the clumsiness of debug boxes.

Support for direct TCP/IP connections in the OPEN-R SDK also aided debugging enormously. Debug tools can now be written in any language without the necessity of an extra program to bridge between it and the OPEN-R environment. Furthermore, direct TCP/IP connections are considerably faster than the original TCPGateway method.

The increased network throughput meant data logging was now a useful feature for debug tools (since practically all data from the robot now reaches the PC). This allows certain simulations to be executed off-line – for example, MATLAB^{tm} analysis of world model and vision data.

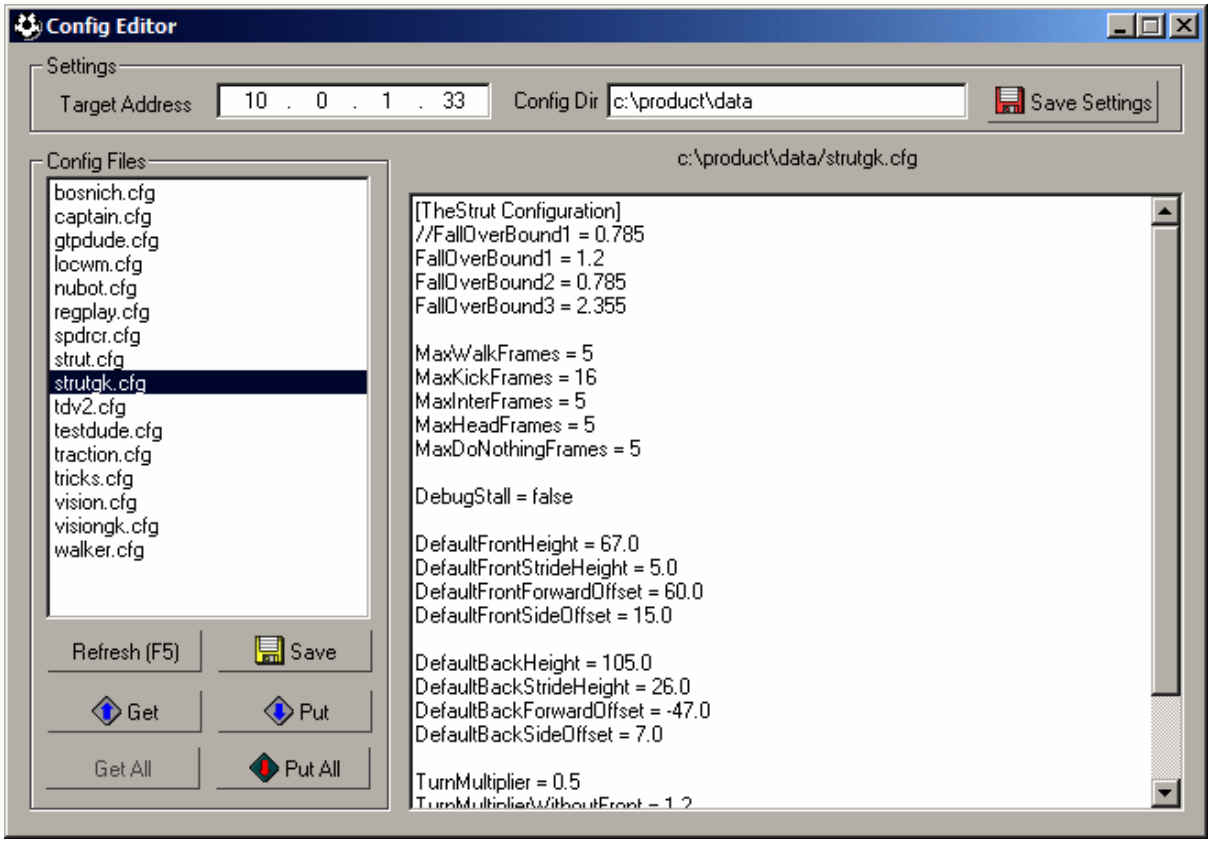


Figure 2.2: Screenshot of ConfigEditor with walk configuration displayed

3 Vision

The visual processing subsystem, which operates on the AIBO, is the result of a cumulative effort of a number of members of the NUBot team over the past two years. This system, like many others on the robot, whilst simple to explain is reasonably complicated in its implementation. Humans process images with apparent ease, quickly filtering out useless information, identifying objects based on their shape and colour and then passing this information on to the brain for processing. When implemented on a robot this process is not as easy as it initially appears

The processing of an image consists of a number of steps. These steps can be described as follows:

- Colour Classification
- Run Length Encoding
- Blob Formation
- Object Recognition

Before discussing these modules in detail we first describe a preliminary study of the camera system itself, and it's characteristics.

3.1 Camera Study

As an initial stage of enhancements to the vision system, a study into the characteristics of the camera itself was undertaken. This study was performed to get a better fundamental understanding of the behaviour of the camera and its associated limitations.

With the current operating system, and CMOS camera installed in the AIBO ERS-210A, we have the following vision system parameters:

maximum resolution	176 × 144 pixels
frame rate	25 fps
format	YUV bitmap
lens aperture	2.0
focal length	2.18 mm
Field of View	±28.8° horizontal, ±23.9° Vertical

The camera settings used in our study were:

White Balance:	4300K fixed (FL_MODE)
Shutter Speed:	1/200 sec fixed (FAST);
Gain:	0 dB fixed (LOW/MID)

3.1.1 Camera Response Analysis

We performed tests of the camera's response to several sheets of coloured cardboard stimuli. The cardboards chosen had colours that were similar to important game colours; such as, orange (ball), white (border), green (beacon), red (dog uniform), and blue (beacon). The trends and figures used in this discussion will be using the orange cardboard response. The resulting trends are similar for all cases and can be generalised from the orange case.

The image of a uniform orange cardboard shown in Figure 3.1 was taken by the AIBO camera under a light intensity measured of approximately 1550 LUX. Looking at this RGB representation of the orange cardboard image, there is a noticeable darkening of the image at the corners. This phenomenon spurred an analysis of the raw YUV bitmap data of the image to get a more objective sense of what was happening to the image. We use MATLAB^{tm} 6.5 to create graphical representations of the YUV data. These graphical representations are shown in Figure 3.2 through Figure 3.4. Each graph shown represents a map of the Y, U, or V value at each pixel in an image.

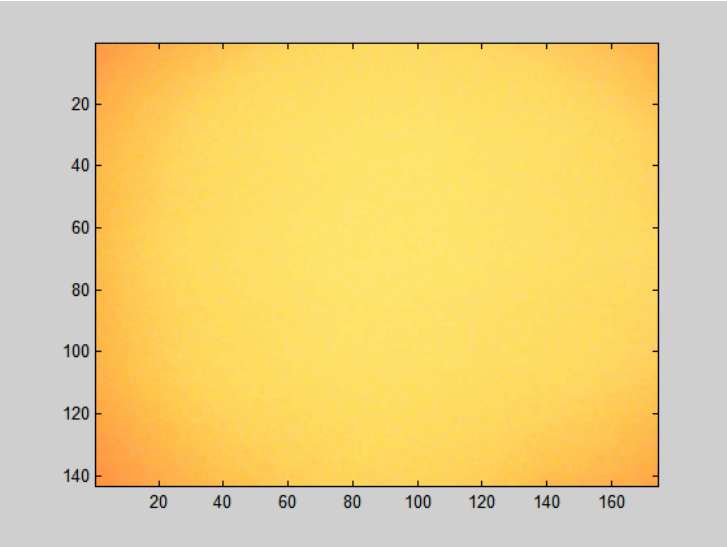


Figure 3.1: Image of Orange Cardboard taken by AIBO camera

In the graph of the Y data there is definitely a drop in Y values as we approach the edges and the corners of the image. Similar trends of data values either increasing or decreasing as they move towards the edges and corners of the image occur in both the U and V, but are less pronounced. (Whether the U and V values follow an increasing or decreasing trend depends on what the colour of the cardboard is.) There is also a large region towards the centre of the image that is fairly constant over all three data components. Additionally, though it is impossible to see in these figures, there are rows and columns of pixels at the edges of the image that consistently give false information. To be on the safe side, a small buffer of these edge pixels should be ignored so that this spurious information does not negatively affect the performance of the vision system.

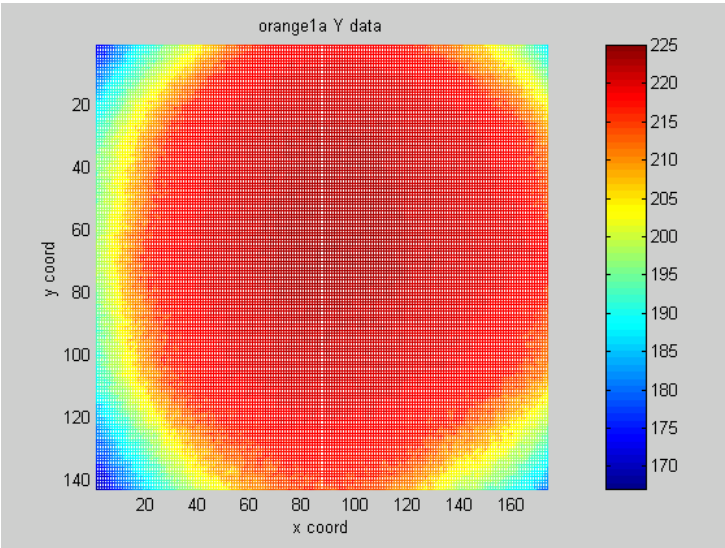


Figure 3.2: MATLAB map of the raw Y data of the orange image

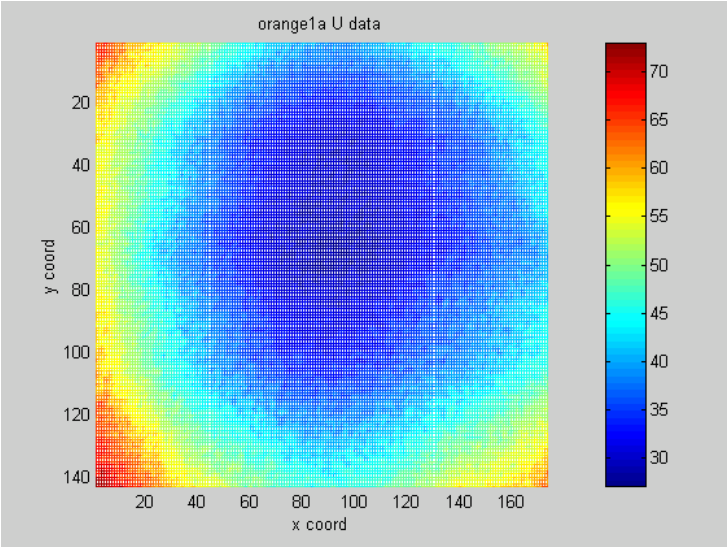


Figure 3.3: MATLAB map of the raw U data of the orange image

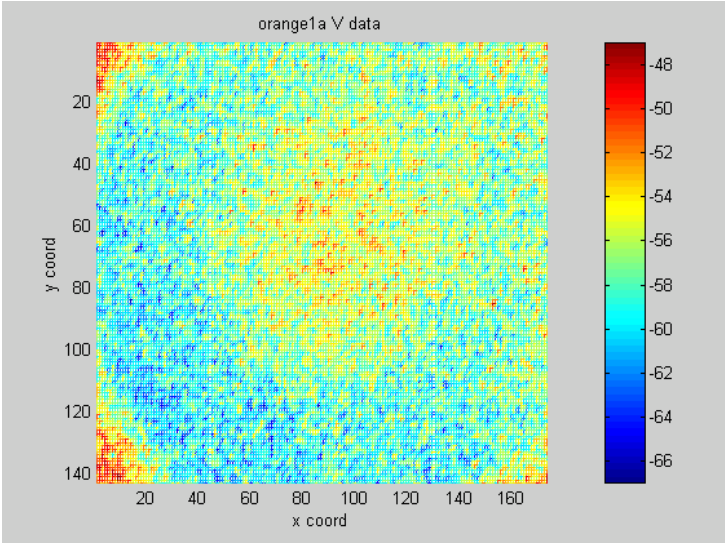


Figure 3.4: MATLAB map of the raw V data of the orange image

This information explains a problem the NUbot team encountered while developing their vision system for RoboCup 2002. Initially, they created classification lookup tables of the colours of important objects (i.e. the ball) by placing these objects in the middle of the image. However, because portions of the camera record the raw YUV data of the image differently than others, this resulted in the vision system not properly recognizing important objects when they were not centred in the image frame. Fortunately, it was eventually discovered that these objects also needed to be placed at the corners of images when creating lookup tables.

From the images above, it at first appeared that some manner of camera calibration may be possible. To this end, we tried several passive algorithms, (i.e. using predetermined compensation) for image compensation to produce a more consistent response. The results produced were of dubious benefit, since the corner distortion observed above depends on the precise conditions (e.g. orientation of cardboard to light source and robot) the image was taken under. For this, and computational reasons, we did not incorporate any image corner compensation in our vision software. Further details of this can be obtained from [4].

3.1.2 Image Deviation/Noise Level Analysis

In addition to a test of the camera response to colour stimuli, an analysis of the noise levels present in an image was undertaken. Improved understanding of noise levels is desirable to determine things such as whether or not smoothing may be desirable, whether finer quantisation of lookup tables is justified and so on. The Video data present for an object can include both temporal (or time based) variations, and spatial (or position based) variations.

We first considered temporal variations. A test was conducted where six consecutive images of an orange cardboard were taken using the AIBO camera, with identical lighting and other conditions. Over these six images, the mean and standard deviation of the video components for each pixel was calculated. The temporal standard deviation for different video components is shown in Figure 3.5 through Figure 3.7.

Again there seems to be a middle section that seems to be providing fairly good and consistent data. The average standard deviation for the Y data is 0.7523.

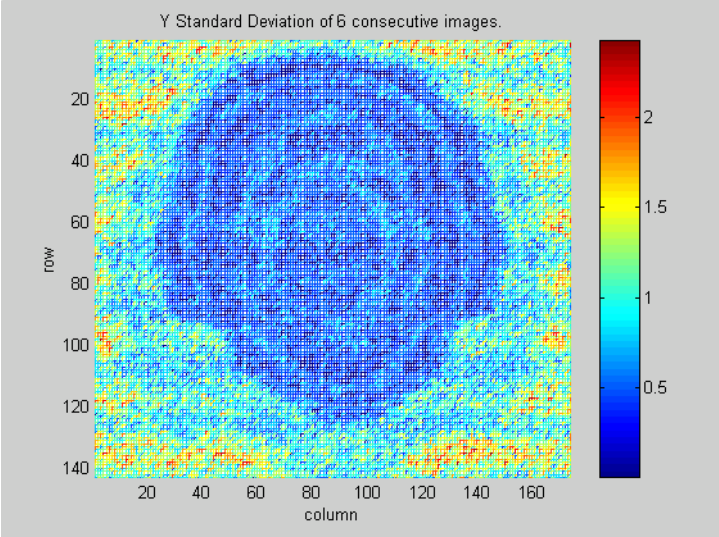


Figure 3.5 Temporal standard deviation of Y value of each pixel

The U data shows a similar trend to that of the Y. However, in the U data the middle section is larger and the noise levels are slightly higher. The average standard deviation for the U data is 0.9524.

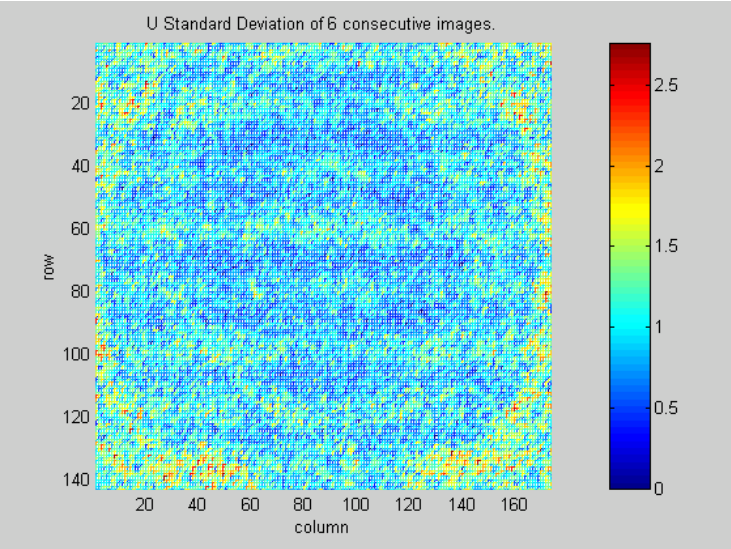


Figure 3.6: Temporal standard deviation of U value of each pixel

The V data does seem to follow any specific trend regarding the middle area and the edges. Instead the noise seems to be randomly distributed throughout the image. The noise levels in the V data are also higher in general. The average standard deviation for the V data is 1.6318.

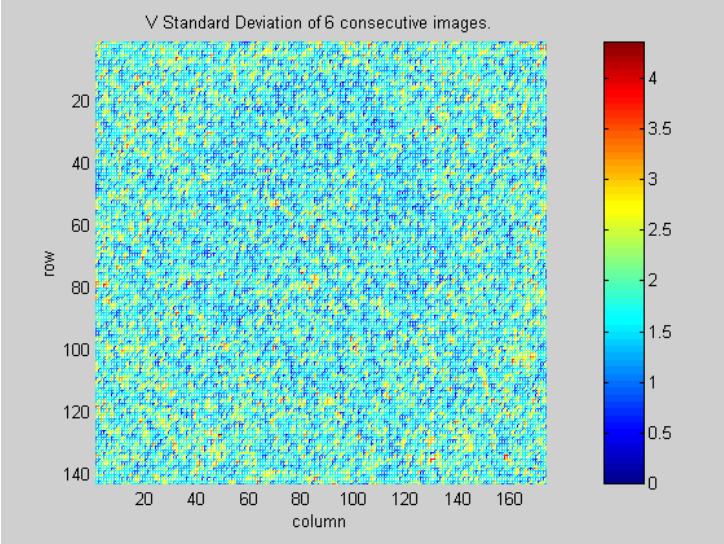


Figure 3.7: Temporal standard deviation of V value of each pixel

Spatial deviations in the colour reported by the camera have been considered. Similar tests to those of the temporal variation are conducted, except in this case, the noise causes locally over an image has also been investigated. This information can be obtained by computing the local arithmetic mean for each pixel in an image of an orange cardboard. Then these local mean values can be compared to the pixel values of the original image. This process was performed using MATLAB and defining local as within 5 pixels of a given pixel. The results are shown in Figure 3.8 through Figure 3.10.

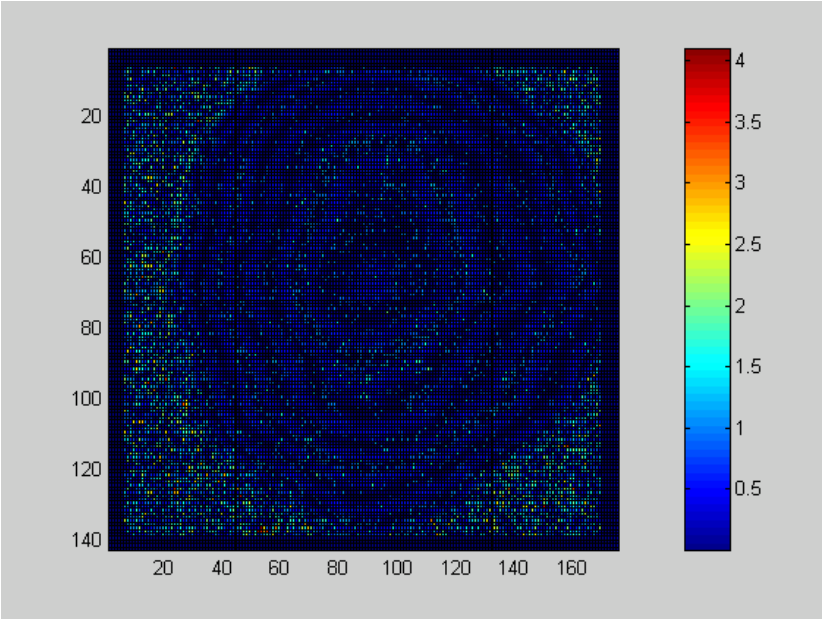


Figure 3.8: Spatial Y component deviation

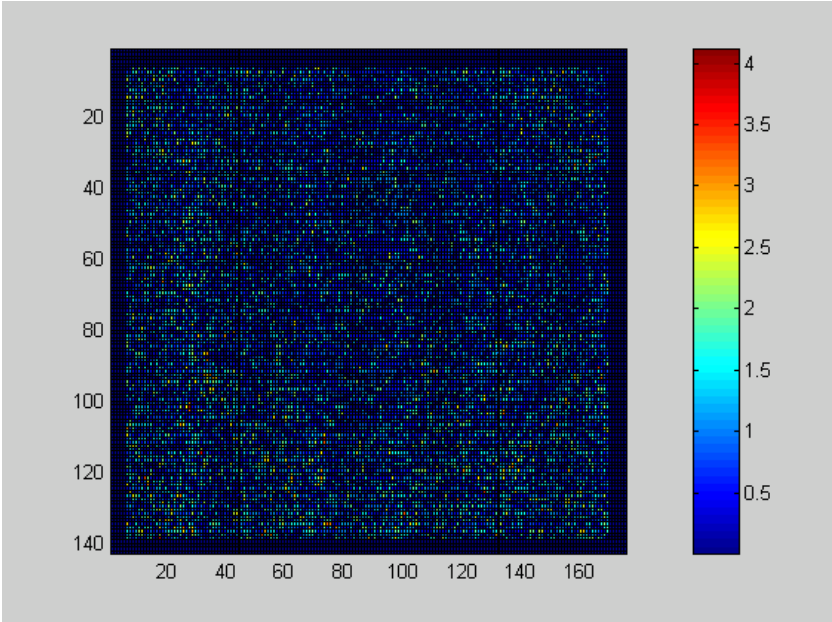


Figure 3.9: Spatial U component variation

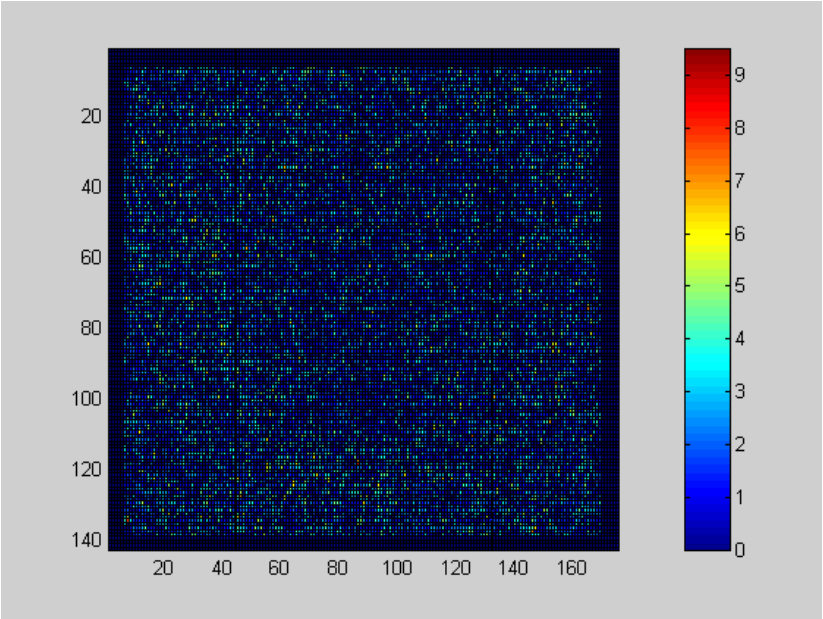


Figure 3.10: Spatial V component variation

Once again for the Y data, the middle section is fairly free of noise, while there is more noise in the outer portions of the image. The overall average deviation in the Y data is 0.4395, while the maximum deviation is 4.0992. For the U and V data, there is no differentiation between the middle and the outer portion of image and sporadic noise is present throughout. The average deviation in the U data is 0.6172 and the max deviation is 4.1157. The average deviation in the V data is 1.3309 and the max deviation is 9.4959.

3.1.3 Camera Study Conclusions

While the attempts at camera calibration were ultimately deemed inadequate to warrant use in the current vision system, valuable information and knowledge was obtained by this camera study. Throughout the study the information consistently showed that the centre of the image provides more consistent and reliable data than do the fringes of the image. This knowledge of the camera images may make a vision system in which colours and objects detected in the centre of the image are given a higher confidence and priority than those detected at the outer edges of the image. In addition we noticed that for a given individual camera pixel, the time variations in the response of that pixel to a fixed image scenario were fairly small (typically with a standard deviation of around 1-2; and peak deviations of approximately 4 units). However, the spatial variations (i.e the consistency of adjoining pixels) were higher, in some cases having a peak deviation of as much as 10 units in the V component. Further work on this aspect of the camera is warranted to examine whether this is due to slight image or camera inconsistencies. More extensive testing might also give more accurate results due to the larger sample size possible.

3.2 Low Level Vision

3.2.1 Colour Classification

The processing of an image begins when the robot receives notification from the operating system that an image has been taken by the CMOS camera and is ready for processing. Images stored in memory are encoded in YUV bitmap format and have a resolution of 176 x 144 pixels. The image shown in Figure 3.11 is an example of an image taken by the robot's CMOS camera.

Figure 3.11: YUV image received from the CMOS camera

The image retrieved from the camera is then classified according to its matching colour in the YUV look up table i.e. the software indexes into the Look Up Table using the Y, U and V components of the colour assigning to the pixel one of the standard colours (Orange, Yellow, Blue, Red etc) recognised by the robot. This table is essentially a three dimensional array which maps real world colours into the closest matching colour recognisable by the robot.

This process can be seen in the image below where for example the various shades of yellow, which make up the goal and the beacon, are now recognised as simply yellow, which the robot can then identify as goals and components of the beacons.

Figure 3.12: Classified Image

3.2.2 Run-length Encoding

Once colour classification is complete, the robot takes the first steps necessary in recognising objects within the image. Run-length encoding, perhaps the most computationally demanding phase of vision processing, iterates its way through the image row by row finding groups or runs of pixels which are the same colour. The result of the process is an array of solid horizontal lines of a given colour and two points in the image that indicate the start and end points of the run.

3.2.3 Blob Formation

Using these runs the process of blob formation begins, merging adjacent runs of the same colour into blobs of colour, which the robot will eventually recognise as objects. The blobs in the image which represent independent blocks of a single colour are defined by the x and y coordinates (top left and bottom right corners) and their colour. Figure 3.13 shows the result of the blob formation process on the image seen in Figure 3.12.

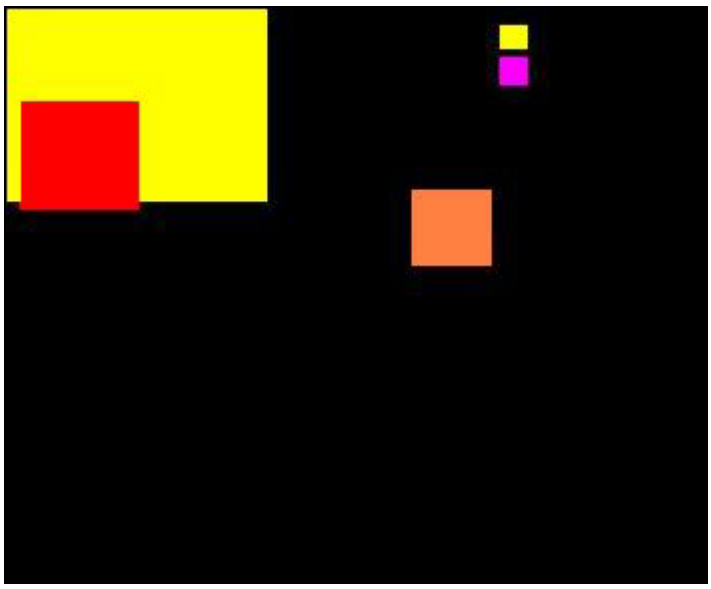


Figure 3.13: Blob Formation

3.2.4 Object Recognition

Following blob formation is the process of Object Recognition. This process represents the conversion of blobs of colour into known objects. The vision system uses a combination of colour, size and position, relative to other blobs, in determining whether the blob represents a physical object in the game.

Orange blobs are first considered to find the most likely candidate for the ball. Secondly, pink blobs are examined to determine whether beacons are present in the image. A pink blob next to another colour (yellow, blue or green) indicates that when the two colours are combined they may represent a beacon. Following the identification of beacons large, independent yellow and blue blobs are examined for the presence of goals within the image. Finally, red and dark blue blobs are considered to determine whether other robots are present.

After all objects in the current frame are recognised, the headings and distances for all objects in the image are passed to the localisation module. Localisation determines the location of the robot on the field using the data provided from the vision system.

The research completed as part of this project, serves to improve the information provided by the 2002 Vision Processing Subsystem to the Localisation module. These improvements can be categorised into two main areas. The first area focuses on improving the distance and heading data of existing data provided to the localisation module. Improvements have been made in the calculation of the distance to the ball, beacons and goals and in the heading calculated to these objects. The second area focuses on providing additional information to the Localisation module, such as information about the corners of the goal box and other robots on the field. The remainder of this report will focus upon several different areas where research has been completed and improvements made.

3.3 Field and Sideline Detection

As stated in the introduction to this section, the sole purpose of the vision subsystem is to process images in order to extract information about the robot's environment. This information is subsequently passed to the localisation module for development of the robot's world model. Unfortunately, situations can arise where the robot may go for extensive periods without viewing critical objects on the field such as goal posts and beacons. This lack of information leads to the robot becoming disorientated and uncertain about its position.

One reason for this lack of information is due to the camera's narrow field of view. The image received from the camera is relatively narrow in comparison to human vision and contains little information. Secondly, the frenetic pace of the soccer match and the manner in which the robot focuses on the ball, once located, can lead to extended periods where the robot may fail to recognise a beacon, whilst locomotion and collisions may have occurred. This combination of lack of visual cues, and errors in odometry can result in the robot pointing in the wrong direction and in a different location to the value determined by localisation. Poor localisation can have disastrous consequences such as kicking the ball in the wrong direction, crossing the illegal defender line and in extreme cases shooting own goals.

The difficulties caused by these problems may be reduced by adjustments to behaviour. Some examples of this are: fixate on beacons (where the robot periodically interrupts play to search for a beacon); pausing to update localisation data before performing critical moves (such as kicking the ball). In both cases, however, there are negative side effects of the behaviour modifications such as slowing play and providing opportunities for the opposition to seize the ball. These problems have prompted research into finding additional points or objects on the field from which to localise.

One such idea was the use of the field and sideline information within the image to provide additional information about the robot's placement on the field. The following sections document the process used in identifying and determining the equation of the field and sidelines within the image.

3.3.1 Sideline Detection Algorithm

Whilst easy for a human, the detection of field and sidelines by the robot is non-trivial and requires extensive processing of the image. Under the implementation discussed in this report, the key to determining the presence of field and sidelines lies in the system's ability to discern field-green/boundary-white and boundary-white/field-green transitions within the classified image. In contrast to this algorithm, techniques such as those used by the German team, focus on the unclassified image and require a significant shift in the Y component for an edge to be recognised, subsequently using a look-up table to identify the edge [12].

The search for points located on the field and side lines, begins with the classification of field green and border white in addition to the ball, goal and beacon colours, which have been previously used. It should be noted that the detection of sidelines cannot simply be accomplished in the same manner as other objects, as the process of performing run length encoding and blob formation on white and green would be prohibitively processor and memory intensive and of little use in the determination of field and side lines. It was for these reasons that a much simpler method of searching the classified image for field and sideline points was implemented.

3.3.2 Basic Sideline Detection

The initial algorithm, first proposed in an earlier project [4], implemented the idea of intelligently obtaining the location of points on the sideline using the presence of the centre (green and pink) beacon. This beacon positioned half way along the field and directly above the sideline, provided a starting point from which to start searching. The algorithm discussed in this paper can be described as follows:

- Locating the centre beacon in the image and if present continue
- Scan the pixels below the centre beacon for a boundary-white to field-green transition
- Record this transition point as a point on the sideline
- Scan columns of pixels left and right of the discovered point for other transitions and points on the sideline

Upon successful discovery of a fixed number of points, the principle of Least Squares Line Fitting can be used to determine the equation of the sideline in the image.

3.3.3 Improvements in Sideline Detection

The implementation of this basic algorithm proved the validity of the original concept, with the algorithm being subsequently extended to handle situations where beacons, other than the central beacon, were present. In this improved algorithm, corner beacons were used as reference points upon which the sideline should be broken in two and a corner formed. The use of a simple case structure permitted the identified points to be split into multiple sidelines where a least squares fit was applied to points located between the beacons.

Improvements to the fitted line were also obtained through the implementation of a least squares fit algorithm which filtered outliers based on their distance from the generated line. Knowledge of the last point found was also used in the quick identification of the location of the next field-green/border-white transition. Using a localised search about the last Y value found, rather than searching the entire column, greatly improved the computationall efficiency of the original algorithm.

While these algorithms were promising in their ability to provide the sideline in the presence of a beacon, they fell short in determining the equation of the sideline in the absence of any beacons. The problem with removing beacons from the image is that there is no longer a reference point from which to start searching for sideline points or for that matter to split the points into multiple sidelines. The algorithm therefore required the entire image to be scanned in order to identify the field-green/border-white transitions and hence the presence of the sideline. Admittedly, it was noted that having identified the presence of one sideline transition, localised searches could be used to find the next transition. It was also observed that using the knowledge of the sideline's previous position, combined with information of how far the camera had moved, would allow the position of the sideline to be predicted and hence a full scan of the image avoided.

In situations where no previous sideline existed, a full scan of the image using a grid structure was performed. In this case, a grid structure was appropriate as useful sidelines transitions are at least a few pixels wide and easily identified when checking only the pixels located in the grid.

3.3.4 Field Line Detection

Significant effort had been invested in improving the performance of the sideline identification algorithms, in areas such as looking at the location of the last sideline and in performing a localised search for boundary-white/field-green transitions. However, the majority of this code was removed in the implementation of the Field Line Detection algorithm.

In the identification of field lines, the possibility exists for multiple disjoint lines to be present within the image e.g. on either side of a robot or around the goal area. As such, it became necessary for the entire image to be scanned to ensure that no transition points were missed and hence no potential information discarded.

In implementing the algorithm, it was decided that for a field line to be present a field-green/boundary-white transition and a corresponding boundary-white/field-green transition must be found. These two transitions marked the change from field to field line and back to field again,

ruling out the possibility that the point was actually part of the sideline. In this way points in the image located on the bottom edge of the field lines are found.

3.3.5 Side and Field Line Data Association

Having acquired the points in the image believed to be part of the field line, the problem of associating points with individual field lines proved to be the next major obstacle. As can be seen in Figure 3.14 the presence of multiple field lines within the image, meant that the points found no longer belonged to a single line.

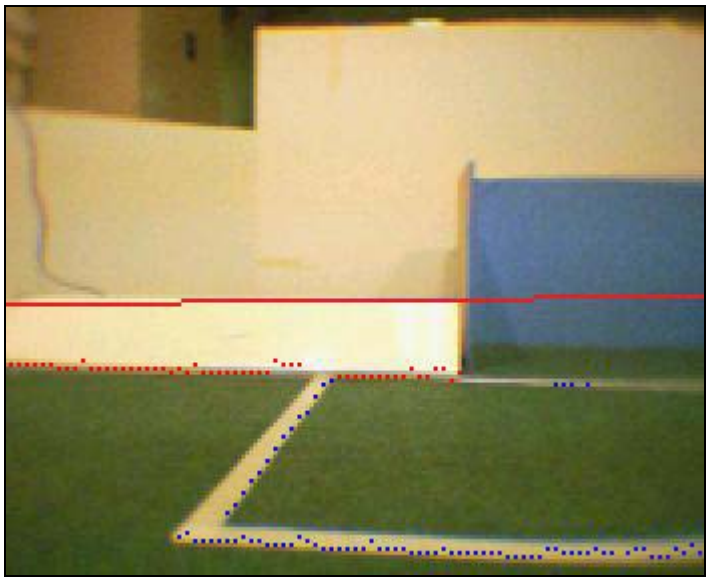


Figure 3.14: Point identification of field (blue) and side line (red) points. The solid red line is the horizon line (see Section 3.4).

The data association problem was solved using a simple clustering algorithm such that if the distance of a point from any previously generated line is too great then a new line will be formed. In the implementation of the algorithm, points are processed in order from left to right within the image to minimise the number of lines generated.

Once all points have been allocated to a line, the start and end points of each line are compared with all other lines, identified in the previous step, to determine whether the points represent a continuous field line. In Figure 3.14 the clustering algorithm results in three field lines. The first field line is reasonably obvious and represents the four points identified in the goalmouth, while the second and third however are not so obvious and represent the two components of the penalty box. The presence of the corner forces the points to be split in two. In processing the points from left to right, the distance of the points from the end of the line eventually becomes too great forcing a new line to be generated.

Following data association, the algorithm checks the start and end points of each line merging the lines if they are sufficiently close. In our example, the two lines that form the corner will be determined to be part of the same line and will be merged. The merge occurs in such away that the overall ordering of the points is maintained with the two ends of the line closest to each other being joined. The reason for maintaining the order will be explained in further detail in section 3.5.1.

Although these algorithms were essentially complete, one further problem related to the filtering of outside environment from the image was presented. The problem experienced and the solution implemented are explored in the following section.

3.4 Horizon Line Determination

In developing the field and side line algorithms it was realised that should a robot look above the white barrier surrounding the edge of the field then pixels classified as boundary-white and field-green could quite easily be confused with field-green or boundary-white of the field and resulting in incorrect transitions being identified. It was therefore necessary to filter these points prior to clustering to avoid processing unnecessary data and including invalid points in line formation.

Although individual elevation checks for each point identified were initially proposed, the use of the horizon line in excluding invalid points and in reducing the overall search area was the eventual solution. The method used to generate the horizon line, which permitted the splitting of the image into useful and un-useful sections, was inspired by the 2002 German Team RoboCup report [12]. In this report, horizon lines were used not to filter parts of the image but to align a grid, which was scanned to identify useful information within the image.

The result obtained for a stationary robot, with adaptation for the NUbots coordinate system, and some other minor fixes, can be seen in Figure 3.15. Whilst the horizon line in this image appears to be correct, the horizon line suffers excessive movement whilst the robot is moving. These problems will be discussed in the following section.

Figure 3.15:The resultant horizon line

3.4.1 Problems with Horizon Determination

The only problem experienced with the horizon line implementation, as described in the German report [12], was excessive movement of the horizon line whilst the robot was moving. The proposed solution for correcting this problem was to use the data from the robots accelerometers to estimate the tilt and roll of the robots body, which when combined with the tilt, pan and roll of the camera should give an accurate result. The tilt and roll of the robot were calculated using the forward, side and vertical accelerometer sensor values with the inclusion of the tilt and roll data manifested in a modification to the original transformation matrix.

This modification had only limited success with excessive movement still present in the horizon line. Initially it was thought that the sensor data was being sampled too infrequently, as under the existing NUBot system only the last sensor reading in each packet of four is utilised (25 samples per second). Upon further analysis it was observed that the data returned from the sensors, was not consistent with the tilt and roll of the body and significantly affected by the robots walk. The walk, which is specifically designed to impact with the ground to increase traction on the slippery field,

causes spikes of over $10ms^{-2}$ in the accelerometer data that was in turn manifested in bad estimations of the tilt and roll of the robot. Modifying the walk to lessen its impact upon the accelerometers, whilst possible, is undesirable as this modification would affect the performance of the walk. The results of the experiments performed on the accelerometers, shown in Figure 3.16, clearly depict the peaks in the raw data returned from the sensors whilst the robot was walking.

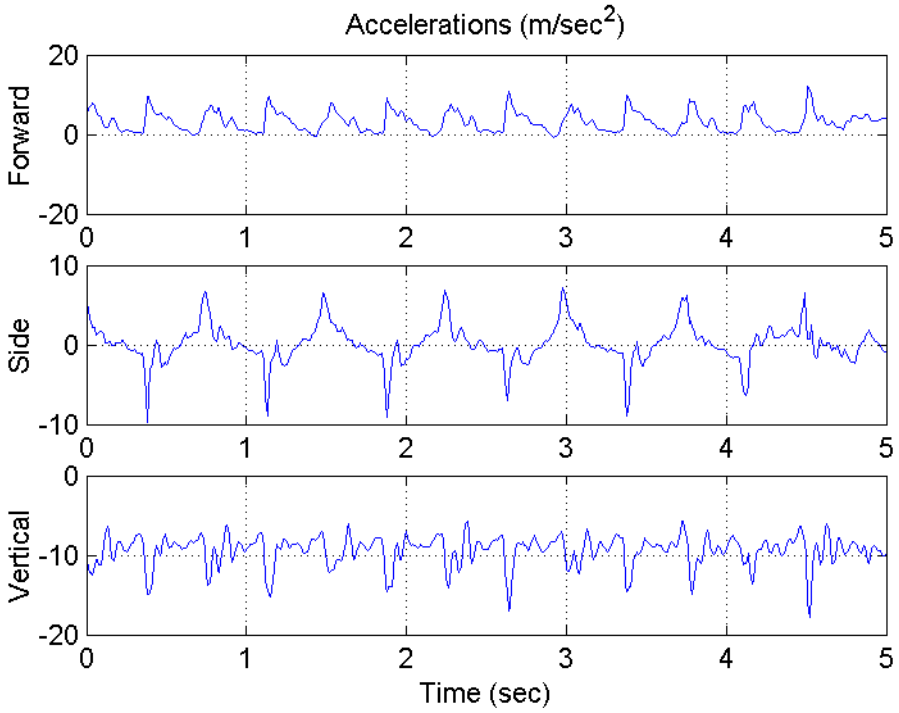


Figure 3.16: Example Raw data from accelerometers

3.4.2 Accelerometer Filtering

Although filters that used the mode, median and average over a frame were considered in the removal of outliers, it was quickly realised that because the peaks were not single outliers but part of a continuous curves that this was not possible. Due to the continuous nature of the curve, it was thought that some form of value capping or limit in the possible change could be used to filter some of the excessive sensor values being fed into the algorithm. Whilst capping did appear to lower the impact of the large accelerometer values and hence the movement of the horizon line, it was insufficient to satisfactorily stabilise the horizon for use in any algorithm.

The use of second order digital filters also proved beneficial in the filtering of extreme values. In using the second order filter, both vertical and forward accelerometers were heavily filtered (with a response time of around a second), such that the estimated tilt would vary only slightly whilst walking. Such a filter had the negative side effect of adding significant lag to the system.

The solution finally used to solve this problem was to calculate the average of the accelerometer values not over the length of the frame, as had been previously trialled but over the length of a single step cycle. While this has the affect of adding additional lag into the system, the lag is such that providing the walk is consistent between successive steps the calculated values for the tilt and roll will be in phase with the step and hence matched to the expected values.

The result was a horizon line, sufficiently stable that it could be used to filter parts of the image that could never contain valid field or sideline transitions. The use of this system also has the added advantage of reducing the average processing time of the image greatly improving the efficiency when the robot is looking straight ahead. Having now determined the presence of field and sidelines

within the image, the following section documents the ways in which this data was used to assist the robot in game play.

3.5 Object Recognition using Field and Sidelines

The provision of additional information to the localisation module has always been one of the key objectives of the Vision processing subsystem. This section considers the way in which the field and sidelines identified by the algorithm in the previous section are used to provide useful information to the localisation module. The ideas discussed below include the determination of the vanishing point to provide the robots absolute orientation, the identification of penalty box corners to assist in goalie localisation, centre circle and centre line identification.

3.5.1 Penalty Box Corner Detection

As stated in the previous section, little data is retrieved from vision when the robot is chasing or focused on the ball and hence unable to see objects such as beacons. This problem is particularly apparent when goalies often end up colliding with the goal or stuck in the back corner of the goal unable to move due to the robots belief that it is already in the correct position. The goalie, located at the far end of the field and confined to the penalty box, has little chance of acquiring information from which to localise. The beacons located at the far end of the field are rarely seen, whilst due to obstruction by other robots and their colour the centre beacons are poor indicators of the robots position. The beacons located at the same end of the field as the goalie are often missed as the goalie tracks the ball as it progresses down the field or moves into the goal to correct for position calculations based on poor data.

In addressing the problem of poor goalie localisation, it was decided that it would be highly beneficial if the goalie had additional information from which to localise whilst standing in the mouth of the goal tracking the ball. The solution was to use information about the placement of the field lines and more specifically the placement of the penalty box corners within the image.

Determination of the points which make up the penalty box proved a non trivial task which as seen in section 3.3.1 relied on the searching the image for field-green/field-line-white/field-green transitions. It should be noted that this algorithm was later incorporated into the sideline determination algorithm to save rescanning the image.

The process of fitting field lines to the points found in section 3.3.1 required an algorithm which used a form of the least squares fitting to fit lines through a series of points believed to be in the same line. Whilst the original algorithm attempted to find the corner by generating two lines that minimised the angle between them by splitting the dataset in two, later algorithms focused on minimising the residual error of the two generated lines and the points that composed the line. In this case, the point which generated the smallest error was taken as the corner point, as the lines generated from either side theoretically fitted precisely the lines forming the corner.

The last step prior to passing the vision object to localisation is to determine an approximate distance to the point based on the placement of the point in the image and the tilt, pan and roll of the robot's head.

This idea is closely related to the determination of the horizon line and assumes that points in the image can be converted to distances and elevation in the real world based on the placement of the point within the image and the tilt, pan and roll of the robot's head. That is to determine the distance to an object in the real world given knowledge of the point in the image, the system performs an inverse transform from the image space into real world space in an attempt to calculate the distance to the point. As with the horizon line, data from the robots accelerometers is used to determine the tilt and roll of the robots body.

Once calculated, the coordinates and distance to the centre circle are passed to the localisation module for inclusion in the model. The provision of additional information in the form of penalty

box corners has greatly assisted goalie localisation. The stability and accuracy of placement of the goalie during the RoboCup 2003 competition is testament to the success of this additional information.

3.5.2 Centre Circle Detection

In keeping with the previous section, this section documents the identification of centre circle using the data obtained from the Field Line Detection algorithm described in section 3.3.4.

The centre circle, as with much of the data currently extracted from the environment, closely resembles a field line. It therefore made sense that the detection of the centre circle, should some how be related to the detection of field lines within the image. This was achieved by observing that the centre circle provided a special case when present, in that from certain angles three sets of field-green/border-white/field-green transitions occurred in the same column.

The presence of the centre circle was therefore determined by the existence of three or more sets of field line transitions. An example of where sets of three transitions are obtained can be seen in Figure 3.17.

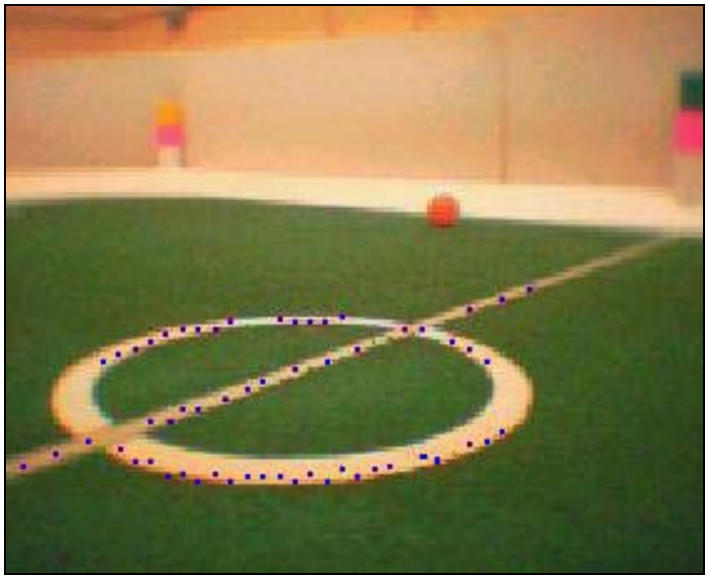


Figure 3.17: Centre Circle Detection

The parameters of the centre circle were found by looking at the sets of triple values found which define the centre circle. The centre of the centre circle was found by taking the pair of triples most separated in the image and bisecting the line fitted through the two middle points. The distance to the centre of the circle is then calculated using the distance to a point system as documented in section 3.5.1.
Although the discovery of the centre circle is only possible from certain angles upon the field, it does provide a point from which further work can be performed in this area. Future work may include research into the fitting of circles or ellipses to accurately determine the presence and parameters of the centre circle.

3.5.3 Centre Line Corner Point Detection

Another area in which field and side line information was used was in the identification of the points where the centre field line intersects with the sideline. These intersection points give further information that the localisation module can use to determine the robots position on the field. An example of the point considered the centre corner point can be seen in Figure 3.18.

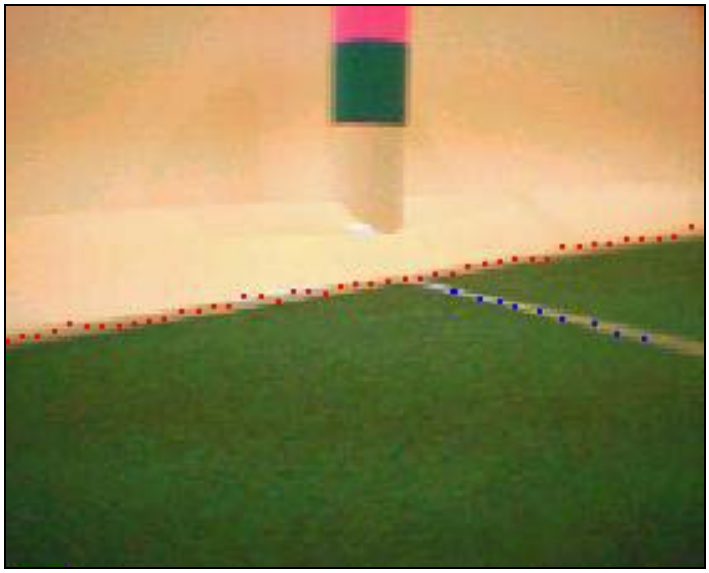


Figure 3.18: Detection of Centre Field Line

The detection of these points requires that a field line must necessarily intersect with a sideline and must not occur in the same image as a goal was identified. The presence of a goal in the image indicates that the intersection of field line with a sideline was not in the centre of the field but the intersection of the penalty box field line with the sideline at the edges of the goal. After checking that the last point of the field line was sufficiently close to the side line, the intersection of the side and field lines is taken as the centre corner point. The distance to this point was found using the distance to a point method described in Section 3.5.1.

The use of additional information such as the centre corner points greatly enhances the robots ability to localise both with and without the presence of beacons on the field. The success of the Edge Recognition system implemented on the robot when combined with the 2003 Localisation subsystem was proven in the localisation without beacons challenge at RoboCup 2003.

3.6 Ellipse and Circle Fitting to the Ball

The process of fitting rectangular blobs to objects within the image, whilst convenient to code and adequate in practice, complicates the process of determining specific details about an object. This results in an overall reduction of information and introduces an error that makes the perceived location of an object less precise. In this chapter, we will discuss an idea implemented on the robot used to reduce the information previously lost in the blob formation process.

3.6.1 Motivation

The loss of information seen in the blob formation process is perhaps most apparent in the transformation of a circular shape, such as the ball, into a rectangle. When other objects in the system are considered, such as goals and beacons, it is noted that these shapes are roughly rectangular and the overall loss of information is minimal. Due to the significant loss of precision in fitting a rectangle to the circular shape of the ball, it was decided that some form of alternate algorithm should be considered to either prevent the loss of information during the blob formation process or recover the lost information once completed. This information is subsequently used to improve the determined properties of the object.

In ball recognition, it was decided that points located on the perimeter of the ball should be identified and a circle fitted through these points. It was hoped that this method would make the system more tolerant of obstructions in front of the ball, such as another robot or the edge of the image. For example, consider the situation shown in Figure 3.19 where the lower part of the ball is obstructed by the robot's leg. The result is a blob smaller than the actual size of the ball and a

resultant distance much greater than the actual distance to the ball. This error in turn affects every other system used by the robot such as localisation (in the determination of the robots position) and behaviour (in deciding which robot should pursue the ball).

Figure 3.19: Obstruction caused by a robot

Fitting a circle using points on the perimeter overcomes this problem, resulting in a better approximation for the radius of the ball.

3.6.2 Determining Fitted Points

In fitting a circle to the ball, it is necessary to know those points (coordinates in the image) which define the perimeter. In the implementation of this algorithm, it was decided that there were two possible options for gaining this information. The first involved storing the information during the blob formation process, whilst the second involved an algorithm running subsequent to blob formation to extract the information from the image.

While the once off determination of perimeter points during the blob formation process may seem the more efficient of the two algorithms, this assumption appears to be incorrect. Although never implemented it was noted early in the design phase that the increased complexity in the algorithms used for run length encoding and blob formation (to store and combine points as the blobs are formed), would have significantly affected both processor utilisation and the memory usage on the robot.

Other problems include the possibility that, due to noise in the image, many orange blobs may be formed. Although these blobs are usually excluded in favour of a large single orange blob, if perimeter points are stored during the blob formation process, unnecessary work is completed in storing the points which define these objects. It is possible that as additional information is required for other objects in the image this algorithm may still yet have a part to play.

The other technique identified for locating perimeter points, involved the recovery of information following the identification of the ball. This algorithm involves performing a horizontal search from the edge of the known blob towards the centre and recording the first pixel found to match the colour of the object (i.e. orange). In using this algorithm, the pixels searched are those shadowed by the blob identified as the ball, which are not orange.

Specific knowledge of the point's location is used to exclude certain points from the set, such as points located on the edge of the image, which are not genuinely on the perimeter of the ball. It is

also possible that once further work has been completed that points found adjacent to a robot could be excluded. Figure 3.20 shows the points determined by the algorithm presented above.

Figure 3.20 - Point determination following Blob Formation

3.6.3 Circle Fitting through Three Points

One idea considered and implemented on the robot was the fitting of a circle using three points on the balls perimeter. This technique, based on the theory that three points on a plane can uniquely define a circle, is described as follows.

Given three points (P1, P2, and P3), the objective is to determine the centre point of the circumscribed circle (the smallest circle that encloses the 3 points). Assuming that the points are connected (with lines P1-P2, P2-P3 and P3-P1) the perpendicular bisectors of lines P1-P2 and P2-P3 can be constructed as shown in Figure 3.21. If either of these lines is vertical (which indicates an infinite slope), the points are renumbered so that the vertical line is excluded. The intersection of the perpendicular bisectors is the centre of the circle.

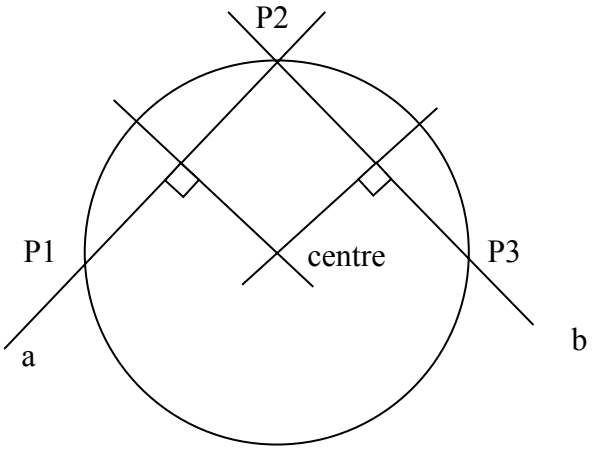


Figure 3.21 - Three Point Circle Fit

This algorithm, although reasonably successful in determining the centre and radius of the circle, was removed following the implementation of the least squares fit algorithm described below.

Whilst theoretically sound, the algorithm poses a number of problems in selecting points appropriate in determining the circle. To accurately fit the circle, three points evenly spaced around the balls perimeter are required. The problem arises in ensuring the points selected are located on the perimeter of the ball and not points on the edge of an obstruction inside the ball. Variation in the position of any of the three points significantly affects the algorithms determination of the centre.

Problems also occur when the three points selected are coincident resulting in two parallel lines and with no resultant circle. The other problem as described above occurs if either line is vertical corresponding to a slope which is infinite. Whilst both of these problems can be checked at run time, the implementation of this algorithm required extensive testing to ensure that divide by 0 errors could and did not occur on the robot. This algorithm was eventually replaced with a robust system utilising the principle of least squares fit. This algorithm as described below uses a greater number of points to remove outliers, obtaining a better fit for the ball.

3.6.4 Least Squares Fitting to the Ball

Although techniques, such as fitting a circle through three points were tested, the method of shape fitting using least squares fitting proved the most successful. Whilst this algorithm is slightly more computationally expensive, the algorithm generates the most accurate fit, especially in cases where obstruction of the ball occurs. It was also noted that as the algorithm uses a higher number of points the generated result is more tolerant of noisy data.

One possibility considered in the development of this algorithm, was the concept of fitting an ellipse to the ball. During testing it was observed that if the ball or head moved whilst the image was saving, the circular projection of the ball skewed, appearing as an ellipse in the image. This phenomenon can be seen in Figure 3.22.

Figure 3.22 - Image skew as seen on the robot

The skew observed is inherent in the reading of the image from the camera, with the values for each horizontal line of pixels being recorded sequentially. An object that is not stationary relative to the head can therefore move from its original position whilst the image is saving. While this skew is technically an error in the image, it is possible that this error can be exploited to provide information as to the relative speed and direction in which the object is travelling from a single frame. Information of this type is usually only available by comparing two sequential frames. As such, it is believed that if an ellipse is fitted to the ball, then the skew can be used to determine the

relative motion and speed of the ball, information that is invaluable when it comes to tracking the ball on the field.

The fitting of ellipses in order to determine this information, does however have a number of side effects when the possibility of ball obstruction is considered. If for example a ball is obstructed by a paw, it is possible that although the ball is not in motion, the obstruction will cause the ball to appear as an ellipse. This fitted ellipse would thus provide incorrect information about the motion of the ball. For this reason ellipse fitting has not yet been implemented and the alternative of circle fitting has been used in its place. In the future, given further knowledge of the obstructions, ball speed and movement information available from the ellipse will be highly beneficial.

Another alternative to the fitting of an ellipse is the fitting of circles through points found on the perimeter. Whilst many different papers have been written about the Least Squares Fitting of circles through a set of points, the paper used as the primary reference for this research was written by Chernov and Lesort [13], which discusses the various strengths, weaknesses and efficiencies of several commonly used Circle Fitting algorithms.

The concept of Least Squares Fitting (LSF) of circles is based on minimising the mean square distances from the fitted circle to the data points used. Given n points (x_i,y_i), $1 \leq i \leq n$, the objective function is defined by equation (3.1):

$$F = \sum_{i=1}^{n} d_i^{\,2} \tag{3.1}$$

where d_i is the Euclidian (geometric) distance from the circle to the data points. If the circle satisfies the equation

$$(x-a)^2 + (y-b)^2 = R^2 \tag{3.2}$$

where (a,b) is the centre and R is the radius, then

$$d_i = \sqrt{(x-a)^2 + (y-b)^2} - R \tag{3.3}$$

The minimisation of the objective function is a non-linear problem with no closed form solution. For this reason, there is no direct algorithm for computing the minimum of *F* with all known algorithms being either iterative and computationally expensive or approximate. It was therefore important that the correct algorithm was chosen for implementation on the robot to minimise the load placed on both processor and memory.

Having reviewed the results in the paper [13] it was decided that a combination of both geometric and algebraic methods should be used. The Levenberg-Marquardt [14] geometric fit method was selected above others such as the Landau and Spath algorithms due to its efficiency. The algorithm is in essence a classical form of the Gauss-Newton method but with a modification known as the Levenberg-Marquardt correction. This algorithm has been shown to be stable, reliable and under certain conditions rapidly convergent. The fitting of circles with the Levenberg-Marquardt method is described in many papers.

It is commonly recognised that iterative algorithms for minimizing non-linear functions, such as the geometric circle-fitting algorithm above, are sensitive to the choice of the initial guess. As a rule, an initial guess must be provided which is close enough to the minimum of *F* in order to ensure a rapid convergence. Research has shown that if the set of data points is close to the fitting contour, then the convergence is nearly quadratic. It was therefore required that in order to minimise the CPU utilisation on the robot, that an initial guess should be provided by a preferably fast and non-iterative procedure. Therefore, the fast and non-iterative approximation to the LSF, provided by algebraic fitting algorithms, seemed ideal for solving this problem.

Algebraic fitting algorithms attempt to minimise the sum of the squares of the algebraic distances rather than the minimising the distances of the sum of the squares of the geometric distances. If the above equations are considered

$$F_1 = \sum_{i=1}^{n} \left[(x-a)^2 + (y-b)^2 - R^2 \right]^2$$
$$= \sum_{i=1}^{n} \left({x_i}^2 + {y_i}^2 + Bx_i + Cy_i + D \right)^2 \quad (3.4)$$

where B = -2a , C = -2b and D = $a^2 + b^2 - R^2$. Differentiating the equation with respect to B, C and D yields the following system of linear equations:

$$\begin{aligned} M_{xx}B + M_{xy}C + M_x D &= -M_{xz} \\ M_{xy}B + M_{yy}C + M_y D &= -M_{yz} \\ M_x B + M_y C + nD &= -M_z \end{aligned} \quad (3.5)$$

where M_{xx}, M_{xy} etc denote moments. The system can be solved using Cholesky decomposition to give B, C and D which in turn gives the parameters a, b and R which define the circle.

The algorithm used on the robot varies slightly from this algorithm in that the implementation uses the Gradient Weighted form of the algorithm. This algorithm is now considered as a standard for the Computer Vision Industry due to its statistical optimality. Further information on Gradient Weighted Algebraic Fit Algorithms can be found in numerous papers.

An example of a circle fitted on the robot can be seen in Figure 3.23. The circle fitted in the image is resilient to noise caused by obstruction and cropping at the edge of the image sensor.

Figure 3.23 - Circle Fitting to the Ball

3.6.5 Ball Distances and Behaviour

Despite the success of the ellipse fitting algorithms, cases have been identified where the algorithms cannot generate a circle that closely fits the ball and as such cannot return an accurate distance to the ball. This situation generally occurs when in excess of 60 percent of the perimeter is obstructed

from view. Although obstructions caused by robots are difficult to detect, it is possible to identify situations where the ball is more than half off the screen. This is achieved by comparing the ratio of the ball's height to its width. A decision is then made as to whether the distance can be successfully calculated.

In cases where the distance is unknown and an accurate calculation cannot be made, it was decided that vision should return a Boolean indicating that the value should not be used. As the localisation module uses a Kalman filter, it is preferable to have a low uncertainty for the ball, knowing the distance to the ball is accurate, than correcting uncertain distances by lowering the certainty on the ball. In the case of the NUBot system, a distance which cannot be accurately determined, is returned as a negative indicating that the Localisation module is to place a lower certainty on the value.

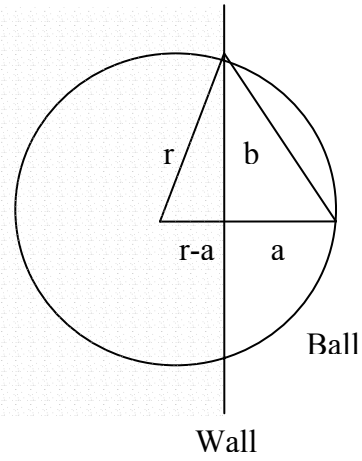


Figure 3.24: Diagram for Ball distance

Whilst localisation excludes this distance from its model, the behaviour system takes the absolute value of the distance and uses the value to determine which robot should approach the ball. As the Circle Fitting algorithm suffered when a small number of points were available, a simple ratio scheme was developed to determine the distance. The algorithm used to determine the radius when the ball is more than half of the screen is as follows.

If we define r as the radius, a as the perpendicular distance of the circles perimeter from the wall and b as half the contact edge (see Figure 3.24), then ratio in (3.5) holds:

$$\begin{aligned} b &= \sqrt{r^2 - (r-a)^2} \\ b^2 &= r^2 - r^2 + 2ar - a^2 \\ r &= \left(\frac{b^2 + a^2}{2a} \right) \end{aligned} \tag{3.6}$$

The algorithm, although vulnerable to providing an inaccurate result in situations where the ball is partially obscured by a robot, provided significantly better results than the 2002 algorithm, which simply returned a distance based on the height of the blob. This distance has proven sufficiently accurate for use in the behaviour model in deciding which robot should chase the ball.

3.7 Infrared Distance Detection

Whilst the distances determined from the image are generally accurate, they can suffer when the height of the object (which is directly used in the calculation of the distance) is affected by an obstruction such as another robot. This section documents the investigation completed into the use

of the robots infrared range sensor and the incorporation of this information into the system. The infrared range sensor is located in the robot's head and returns distance to objects when the measured value is less than 900 mm.

3.7.1 Infrared Sensor Testing

Whilst the original tests of the infrared range sensor were simple and used to determine the accuracy of the device, these tests intensified as problems emerged in the implementation of the original algorithm. In initial tests, the accuracy of the sensor was evaluated by placing the robot a known distance from an object, such as the goal, and comparing the measured value with those returned from the sensor.

Because objects do not always cover the entire image, tests were required to identify the region in the image covered by the sensor. The tests were completed using a robot and a flat piece of board. The board was moved in from four directions (top, bottom, left and right) and the pixel value recorded when the distance returned from the infrared device stabilised. The overlapping region was determined to be the minimum area that an object needed to cover before its distance could be accurately measured by the infrared sensor.

The tests were repeated on a number of robots to prove that the region was robot independent, which was confirmed to be true. The region found during testing is shown in Figure 3.25.

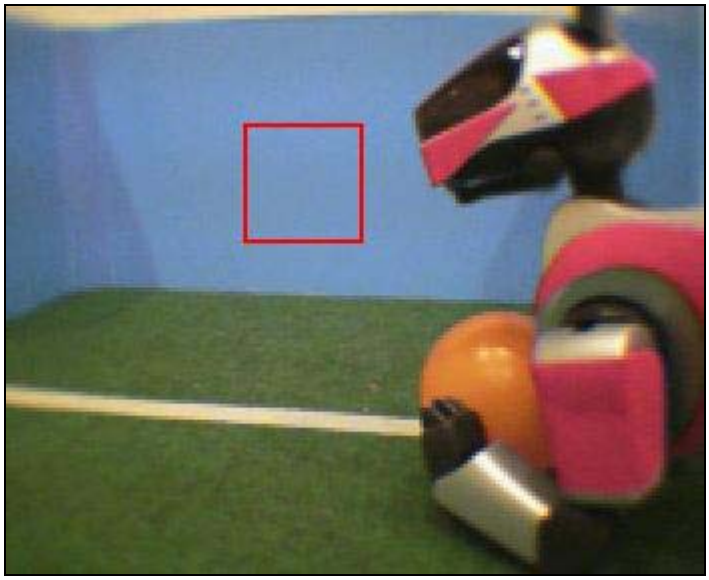


Figure 3.25 - Infrared Range Sensor Region

The offset of the region in the image, is related to the physical offset of the infrared sensor with respect to the camera.

3.7.2 Infrared Implementation

Having completed the infrared testing the use of the data was relatively simple. The algorithm implemented on the robot states that whenever an object covers the infrared region in the image, then the distance determined by the infrared sensor (providing the sensor did not return a value of 90000, indicating an undetermined distance) replaces the distance calculated. Whilst simple in its implementation, special code was still required to handle unique cases in the processing of specific objects.

The first example of object specific coding was the code required to compensate for the circular shape of the ball. The problem observed was that although the blob, representing the ball, covered the infrared sensor region, the circular shape of the ball in the image did not. The problem resulted in measured distances that were greater than the actual distance to the ball.

The algorithm determined if the distance from the centre of the ball object to the four corners of the infrared region was smaller than the radius of the ball. If the four conditions were satisfied then the ball must be larger than the region and the distance was used. Using this system the ball object always covered the image and accurate distances were returned.

Object specific code was also implemented for objects such as the goals. Because rectangular blobs are fitted to objects on the diagonal, it is possible that although the blob may cover the infrared region in the image, the object itself will not be present in this area. A localised search for the objects colour within this region was implemented to ensure the presence of the object.

3.8 Ball on Wall Detection

This chapter describes the simple algorithm written to determine the presence of the ball on or near the wall and the motivations behind this algorithm.

3.8.1 Motivation

The presence of the ball adjacent to a wall during a game can cause a number of problems that deserve special consideration. In the past cases were often observed where a robot, if approaching from the wrong direction, had the potential to push the ball the full length of the field towards the robot's own goal. This action was a result of repeated attempts to grab the ball, which failed due to the collision of the robot wit the wall. The net result is a greatly increased risk of being scored against.

The solution to this problem was to detect the presence of the ball on the wall and to handle the case by walking around the ball before kicking. This results in a controlled approach to the situation, previously impossible when the wall could not be detected.

3.8.2 Wall Detection

The process of detecting if the ball is next to the wall is reasonably simple. It was observed that whenever a ball is located on the wall, the colour on one side of the ball is white and the other field green. The process of checking for the wall is therefore to search left and right of the ball for a white/green or green/white combination.

It should be noted, that this check will not trigger when the robot approaches the ball where the wall is perpendicular to the direction of travel of the robot. In this case the algorithm will detect that white is present on both sides and as will not be triggered. This proved to be highly beneficial, as it is undesirable for the robot to try and walk around the ball in this situation. The final algorithm exploited this feature by performing the searches at 45˚ to the perpendicular access of the ball. The end result is that the check triggers only when the robot approaches the ball with an angle which is less than 30° to the wall.

The search pattern followed by the algorithm can be seen in Figure 3.26. The first image depicts a situation, which satisfies the white/green combination thus triggering the check. The second image depicts a situation where the robot is approaching the ball from an angle greater than 30˚.

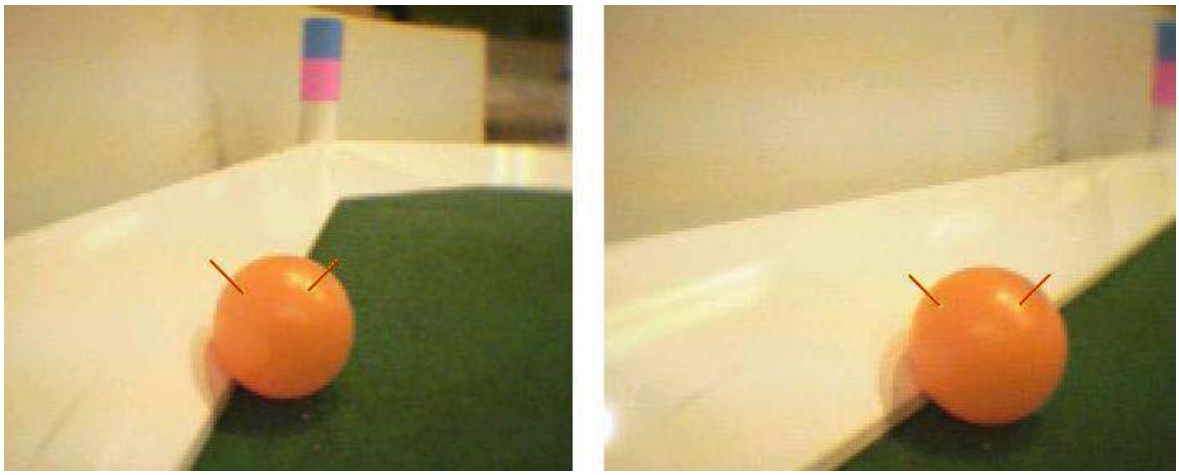


Figure 3.26 – Example of Ball on Wall detection

3.9 Vision Debugging Tools

Over the last 12 months, several tools have been developed in an attempt to streamline development and debugging of the vision subsystem. The result was the aptly named EOTN (Eye of the Nubot) application, which included the following features:

- Colour classification for the generation of look-up tables
- Execution of robot vision off robot
- The generation of classified or unclassified vision steams to be replayed later

3.10 Vision Summary

Over the past twelve months, changes made to vision have served to improve the overall quality and quantity of information provided by the Vision Processing Subsystem to the localisation module. These improvements can be categorised into two main areas.

The first area focuses on improving the overall quality of the existing distance and heading data currently provided to the localisation module. Through the use of the infrared range sensor, algorithms such as ellipse and circle fitting and the correction of problems in blob formation, the distances returned to the ball are now significantly more accurate than was previously possible. This in turn leads to improved localisation and behaviour.

Whilst improving the accuracy of data is important, research has also been completed to provide additional information. This information includes the heading and distance to the goal penalty box corners, centre circle and centre line corners. Additional information has allowed significant improvements to be made in the localisation of the robot and hence the overall flow of the game.

4 Localisation and World Modelling

One of the most important pieces of information for any soccer player is knowledge of where on the soccer field they are and what direction they are facing. This allows the player to make decisions such as where to kick the ball to get a goal and when to defend the player with the ball. Similar information is clearly crucial in robot soccer. The scope of the localisation and world modelling software module is to provide information to allow the robot to make these decisions by determining the current position of the robot and any other objects on the field and combining this into a world model. For the purpose of RoboCup 2003 the only objects modelled were the robot itself, and the soccer ball.

The principle algorithm used, the Kalman Filter, has not changed from 2002. However, the code has been completely rewritten, largely due to the haste with which last year's version was written. Several fundamental changes in the structure of the filter have also occurred, as well as a number of enhancements. There are also a number of extra pieces of data from the vision system that must be processed.

The main software interactions of this module are illustrated in Figure 4.1. Vision provides vision information (range and bearing, corrected for head pan, tilt and roll) on a variety of observed objects (beacons, goal 'posts', ball) to localisation and world model. The locomotion module provides odometry information, and localisation and world model fuses this data together with past data to form estimates of the robot location, and heading, as well as the ball location. With wireless enabled on the robots, the world modelling module also receives the localisation and ball position from the three other robots on the team. This information is processed to determine the current position of the robot, ball and other robots and then communicated to the behaviour control module.

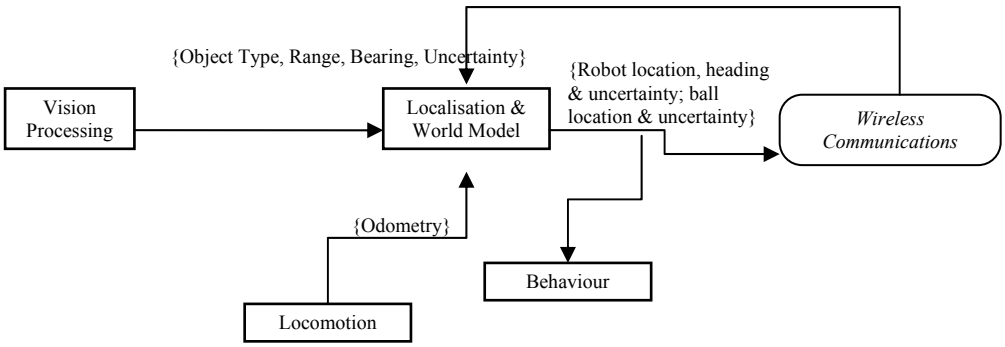


Figure 4.1: Main software interfaces to the Localisation and World Modelling Module

Example sets of vision data in an idealised setting with no robot motion (apart from head panning) are illustrated below in Figure 4.2 and Figure 4.3

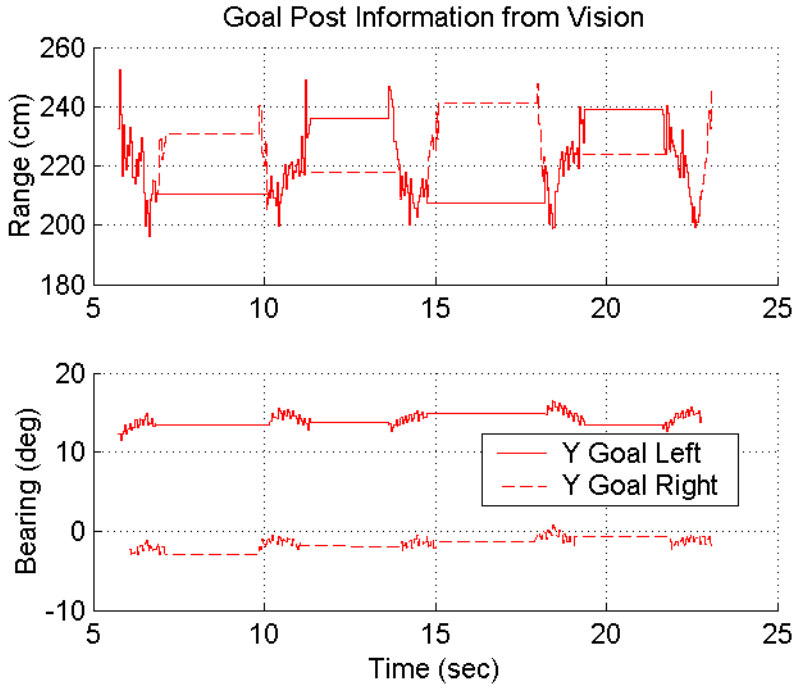


Figure 4.2: Example Goal Post Information for Localisation

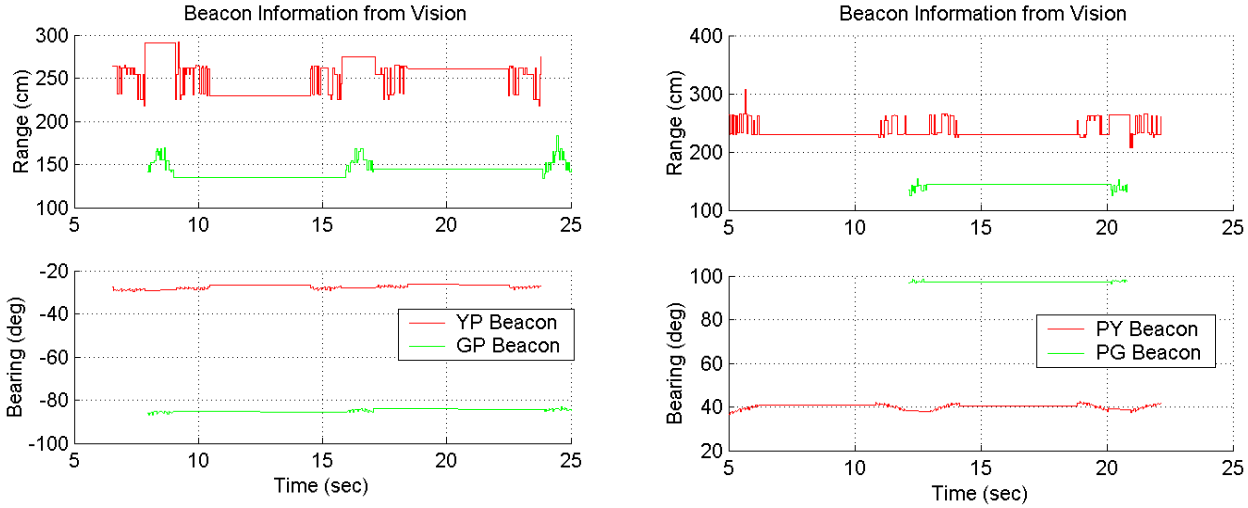


Figure 4.3: Example Beacon Information for Localisation

4.1 The Kalman Filter Algorithm

The NUbots use a Kalman Filter for localisation and world modelling. This section considers the implementation and effectiveness of this Kalman Filter. The Kalman Filter has a long history, with recent references such as [2],[3],[5], and [7] being helpful in understanding the algorithm. It is an effective way to combine information from a number of different sources and also to take into account errors from these sources. It weights the information from trusted sources more than information from less trusted sources and provides an estimate of the accuracy of the output. The Kalman Filter has the advantage that it uses all past time estimates to make a calculation. It also estimates a confidence value and under certain conditions is the optimal method of combining the data. It is in fairly common use in the other RoboCup leagues and is a recognised method of localisation in references on mobile robotics (for example, see [7]).

The Kalman filter is based on a simplified probabilistic description of the robot dynamics, environment and sensors. It takes into account errors in odometry information from the locomotion

model, and errors in the sensor measurements. In both cases, these errors are treated as 'noise' and the filter smooths out the effects of this noise in the position being estimated. This is achieved by incorporating more information from reliable data than from unreliable data. In addition to an estimate of the position, the algorithm also provides an estimate of the position uncertainty.

Another advantage of the Kalman Filter algorithm is that it is not too computationally expensive because it is a recursive solution and does not operate on the data in one large block. Furthermore, the complete probability distribution is represented by a multivariate (n dimensional) Gaussian, which can be completely parameterised by $n+\left(\frac{n^2}{2}+\frac{n}{2}\right)=\left(\frac{n^2}{2}+\frac{3n}{2}\right)$ parameters. Simple linear algebra (matrix multiplications and additions, division by a scalar) are the main operations required.

In order to use the Kalman Filter algorithm, a model of how the system changes over time and also a model of how the sensors operate is required. The parameters passed into the Kalman Filter algorithm are the sensor measurement values (e.g. vision), an initial estimate for the position and uncertainty, an estimate of noise, the input to the system and the system model. The output of the Kalman Filter algorithm is an estimate of the state variable vector and the uncertainty of the state variable estimate) at each time interval. The time/measurement update form of the Kalman Filter was used. The time/measurement form of the Kalman Filter is executed in two distinct updates. The time update projects the state estimate ahead in time based on the input to the system. The measurement update adjusts the time update state estimate to account for any measurements made during the time interval. This form of the Kalman Filter is particularly suited to the RoboCup situation since a variable number of measurements can be incorporated. For example, it is not uncommon for the number of objects relevant for localisation in a single vision frame to vary from 0 to 4 (for example if two beacons, goal post, and ball are observed in a single image frame).

4.2 The Extended Kalman Filter (EKF)

The Kalman Filter equations assume that all the relationships between measurements, inputs and state variables are linear. Unfortunately for the particular case of the robots this is not the case and a different version of the Kalman Filter known as the Extended Kalman Filter (EKF) is used. In the case of RoboCup, the equations were formulated in a form that gave linear time update equations, but measurements were nonlinear functions of the estimated states, due to the trigonometric relationships inherent in the vision system. In our software, this linearization is performed using partial derivatives of the measurement nonlinear equations

4.3 Kalman Filter for RoboCup

In the 2002 version, several Kalman Filters were implemented to perform the localisation and world modelling functions on the robot. These were for the robot's position, the individual ball model and a co-operative ball model. This year these were combined into one Kalman Filter which performed the functions of all three. The new method better represents the dependent relationship between the robot's estimated position and the ball estimate, and is explained in more detail later. Furthermore, the individual and cooperative ball models are no longer distinct concepts. Other differences from last year's implementation will be noted in the following sections.

As with last year, the new Kalman Filter implementation was first modelled in Matlab[tm] and tested with simulated vision data. We feel this is the optimal approach to testing different localisation techniques, since fundamental algorithmic flaws are difficult to detect by observing the behaviour of a complex soccer playing robot.

4.3.1 Derivation of KF for Robot Own Position (Localisation)

Odometry data is combined with measurements of angles and distances to various objects and points on the field to determine the robot's position. The robot is modelled as a state space model

consisting of (x, y, θ); two Cartesian coordinates on the field and heading. The robot localisation routine is called every time a vision update is received which is approximately twenty five times per second.

4.3.2 The Time Update

The locomotion module sends the estimated distance travelled forward, distance travelled left and angle rotated left to localisation and world modelling. This update is sent as often as possible (approximately every frame). In last year's model this information was sent only once per second. This change resulted in the filter's response becoming much smoother and considerably less lagged. In addition, odometry data now accounts for the effect of kicks on the position of the robot. The odometry information is resolved into absolute x and y distances, based on the heading of the robot. This information is used to complete the time update of the Kalman Filter.

4.3.3 The Measurement Update

For each measurement that is obtained from vision, a measurement update is performed.

Previously, measurements consisted only of simple distances and bearings to beacons and goals. This year, the angle between goal posts and beacons is calculated if the two objects are seen in the same camera frame. The reasoning behind this is simple – bearing information tends to be more accurate than distance information, so it is logical to use angle information in lieu of distance information wherever possible.

Improvements to vision have made information about certain 'points' on the field available to localisation. Specifically, bearing to the goal square corners, the centre of the field and the intersection between the centre line and the edge of the field on both sides. Unfortunately, much of this data is ambiguous. While vision may be able to decide that a point is a goal square corner, it can not definitively determine which of the four goal square corner points on the field is currently visible! Localisation and world model must therefore guess which point is current visible based on the robot's current position.

Each time a beacon, (or a goal post which is treated in a similar manner) is "seen", two measurements can be used to update the robot's estimate of its position: the distance to the beacon and the angle of the beacon relative to the current robot heading. For points, only the angle of the point may be used as a measurement. The number of measurement updates is repeated according to the number of distinct pieces of data received.

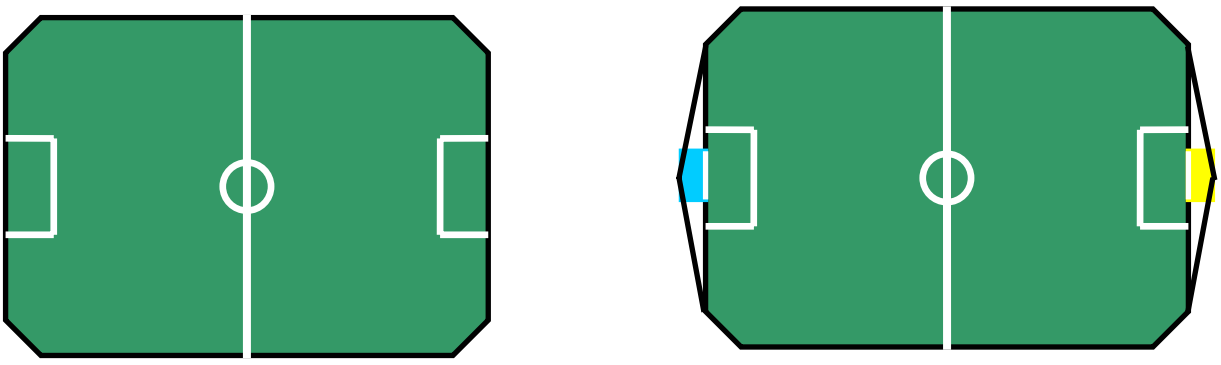
4.3.4 Constraining Robot Position in the Field

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 simplest way of solving this is to clip the individual coordinate that is out of range back into the field. A more insightful solution is to use the fact that there is an estimate of the certainty of the measurements and adjust the robot back onto the field taking into account the uncertainty and correlations in the variables.

The appropriate 'adjustment' is a projection to the most 'probable' point in the field of play. It is well known that this projection is unique for all possible starting points if and only if the region is convex. Furthermore, the action of the projection is guaranteed to produce an estimate that is closer (in the sense of being most probable, as described by the P^{-1} norm) if and only if the region is convex. Thus the exact project to the field boundary, which is a non-convex shape when the goal areas are included, has several undesirable features. Last year's algorithm assumed the inside of the goal did not exist as far as localisation was concerned (see Figure 4.4(a)). This contributed to poor goalkeeper localisation and therefore strange goalkeeper behaviour when the robot was in fact inside the goals. A new solution was proposed this year.

The end borders of the field are each constrained by four straight lines. Two of these lines follow the corners of the field, and the other two run from the inside edge of the corner to the middle of the

goals(see Figure 4.4(b)). Empirical tests of goalkeeper and striker localisation indicate that this method performs more than adequately and is a definite improvement over last year.



(a) Field Shape projection (excluding goal area)

(b) Extended Field Shape projection (including goal area)

Figure 4.4: Field Shapes used for Kalman Filter Estimates Projection

Once we have a convex polygon (or more generally polytope) as the constraint region, this region can be represented as a finite intersection of half planes, i.e. linear inequality constraints. The appropriate projection can be performed by solving an associated constrained least squares optimisation problem. Solving this constrained optimisation, requires solution of a quadratic program with up to 4 linear inequality constraints active. This can be described by equation (4.1) (where X is the state before projection, X' is the state after project, P is the Kalman filter covariance, and the linear inequality constraints are represented by $c_i^T X' < 1; i = 1..n_c$).

$$X' = \underset{X': c_i^T X' < 1; i=1..n_c}{\arg\min} \left\{ (X - X')^T P^{-1} (X - X') \right\} \tag{4.1}$$

To simplify computations we approximate this problem by considering only a single linear equality constraint at a time, as described by equation (4.2):

$$\begin{aligned} & X^0 = X \\ & for\ i = 1...n_c \\ & \quad X^i = \underset{c_i^T X^i < 1}{\arg\min} \left\{ (X^i - X^{i-1})^T P^{-1} (X^i - X^{i-1}) \right\} \end{aligned} \tag{4.2}$$

4.3.5 Lost Robot Detection (or the 'kidnapped robot problem')

One of the dilemmas faced last year was how to detect when a robot had been penalised or otherwise become lost. In principle, this type of situation could be detected by examining where there is a conflict between the measurements expected by the current model (prior to measurement update) and the measurements observed. However, this conflict may have two possible root causes: (i) the measurements could be erroneous (i.e. there are rare occasions when, despite the best efforts and sanity checks in vision, objects will be perceived at completely false locations); or (ii) there may have been major unpredicted changes in the robot or ball location (for example, the most severe case of this is when a robot is penalised, and moved to the centre of the field). Note that this is equivalent to a long studied problem in detecting and tracking jumps (see for example [8]).

In 2003, a relatively simple approach was proposed: Detect several successive (or almost successive) 'bad' (i.e conflicting with the current model) vision measurements to distinct object types. By choosing appropriate threshold values for various constants (such as how many distinct

object types had to be seen, how bad the measurements had to be, etc), the robot is able to detect that it has been manually picked up and moved almost without fail. Also, where 'bad' measurements were on a single object type, this was rejected as an outlier and not used in the model (for example if a beacon is removed and deliberately misplaced on the field).

As well as detecting manual robot repositioning, the detection system allows the robot to quickly regain its bearings after receiving large amounts of erroneous odometry data (as in a severe collision with another robot).

An obvious technique to employ once a 'lost robot' determination has been made is to reset the Kalman Filter uncertainty values. This allows the robot to very rapidly move to its new position without compromising localisation performance in normal situations.

4.3.6 Derivation of KF for Ball Position – Single and Multiple Robot Measurements

The ball's position is now modelled in the same filter as the robot's position. This major change was made because calculations of the ball's position are highly dependent on (i.e. correlated with) the robot's position i.e. if the robot's own estimate is wrong, then the ball position estimate will be wrong as well. The ball's relative position is calculated using distance and bearing measurements to the ball.

The ball position Kalman Filter equations are similar to those used for the robot except that no heading is present.

If communication between the robots is possible via the wireless LAN it is advantageous for them to share information, especially about where the ball is. This is particularly useful if one robot cannot see the ball. In order to share information, each robot calculates the (x, y) position of the ball using trigonometry and sends this information to all the other robots. The use of (x, y) coordinates (as opposed to distance and angle information) is determined by the structure of the software which means the distance and angle information cannot be passed over the wireless LAN link. Each robot then executes its Kalman Filter on all the ball (x, y) coordinates it receives. The information is weighted in such a way that the cooperative ball is virtually always as good as the world model ball calculated by a robot acting alone. This was not the case in 2002, and this improvement allows the concept of shared/non-shared ball information to be removed entirely.

To reiterate: wirelessly received ball data is seamlessly integrated into localisation and world model and is entirely transparent to the robot's behaviour system.

4.4 Kalman Filter Parameter Tuning and Results

4.4.1 Sources of Error

As in all real-world systems there are numerous sources of error which contribute to localisation and world modelling inaccuracies. The sources of error that affect the robot localisation Kalman Filter and the ball position Kalman Filter are summarised in the following sections.

4.4.2 Robot Position Error Sources

The errors that contribute to the robot localisation Kalman Filter Model/Input Noise:

- Robot locomotion data errors: Inaccuracy in locomotion data can be attributed to a number of factors including slip on the field surface, individual robot motor differences, and collisions with other objects (walls and other robots). Of the three, the latter is by far the largest source of error.
- Model linearisation error.

The vision errors that contribute to the robot localisation Kalman Filter Measurement Noise include:

- Misclassified camera pixels: Colour similarities (for example, yellow and orange) can cause objects to be 'seen' where there is no object, or pixels at the edge of a colour block not to be recognised making the object appear a different size to what it is.
- Granularity (limit of reading): The calculation performed to determine a beacon's distance from the robot is constant/(height of pixels) which means the distance measurements are not a continuous function. For example, the difference in the distance measurement between seeing a beacon height of five pixels and six pixels may be as much as 60cm.
- Lighting conditions: Different lighting conditions have a significant impact on the accuracy of the vision information. Shadows and reflections in different parts of the field will make the colours change significantly and cause them to be misclassified.
- Camera velocity: Observations of the distortion of objects when the camera is moving fast indicate that each camera frame update is not very fast. Objects can appear to be 'spread' across the frame which distorts the distance and angle readings.
- Synchronisation of vision and sensor data: Vision angle measurements are adjusted to take into account the angle of the head. When the head is moving fast the head angle and camera frame aren't always synchronised which causes the angle to be inaccurate.

4.4.3 Ball Position Error Sources

The ball position Kalman Filter Model/Input Noise accounts for the fact that there is constant movement of the ball but no knowledge of the input driving this. In addition to the vision errors mentioned already, there are extra vision processing problems involved in measurements of the ball, particularly when the ball is occluded or at close range. These issues are discussed previously in Section 3.6.

4.4.4 Example Localisation Results under Ideal Conditions

One simple test of localisation performance is to see how consistent the results are for a stationary robot. Using the test data in illustrated in Figure 4.2 and Figure 4.3 we also logged using the wireless TCP/IP method the results of localisation. A summary[3] of these results are illustrated below.

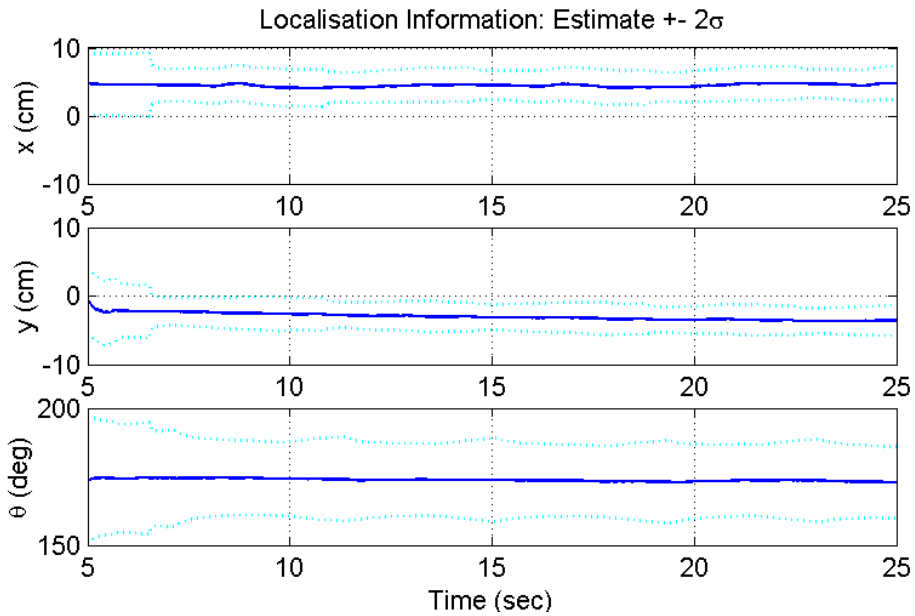


Figure 4.5: Example results of self localisation

[3] Note that a full description becomes confusing, even if we restrict attention to self localisation, there are 3 state parameters, and 6 (if we consider symmetry) covariance parameters to represent.

4.5 Kalman Filter Parameter Choices

4.5.1 Robot and Ball Position Model/Input Noise

Intuitively, if no new information is given to the robot about its position, the uncertainty in position increases linearly with time. That is, if the robot's position is known at a certain time, and then no measurements are made for a period of time after that, the robot could theoretically move at its maximum speed in any direction during that time. The uncertainty should take this into account. This is not true of the Kalman Filter equations, so we adjusted the calculation so that it was.

The value for Model/Input Noise for robot localisation was chosen by simultaneously running a number of test Kalman Filters with different Model/Input Noise values and comparing the results. The position estimates returned by the Kalman Filters were logged to a file and compared to the actual position to determine which performed better. For both the situations of a robot standing still and a robot moving, a small Model/Input Noise performed more accurately. This prevented the robot from changing its position estimate too much when it received inaccurate data.

The lost robot detection system detailed previously allowed a further reduction in the Model/Input Noise value, as large manual changes in robot position no longer have to be accounted for.

The ball model/input noise covariance is fairly high to account for the fact that the ball can move quite rapidly and there is no indication of any change in movement available (odometry for the localisation module) as an input to the Kalman Filter.

4.5.2 Robot and Ball Position Measurement Noise

Measurement Noise values for the robot and ball positions were derived using systematic empirical testing in 2002. However, the testing conditions were unrealistic in that the robot was stationary and the colour classification near perfect. Many of these values have been tweaked and appear to have resulted in better localisation in actual soccer games.

Furthermore, periodic improvements in the vision and locomotion systems have resulted in more accurate input data. This too forced Measurement Noise values to be modified regularly.

4.5.3 Robot and Ball Position Initial Values

The initial value of robot position was chosen to be (0cm, 0cm, 0rad) as this is the middle of the field, and therefore the closest point to all other values on the field. The initial value for uncertainty was chosen to be ($(135cm)^2=18225cm^2$, $(210cm)^2=44100cm^2$, $(2\pi rad)^2\approx 40rad^2$) as it is the maximum possible uncertainty given the robot's estimated starting position.

The ball position and uncertainties are essentially the same as for the robot. However, the ball initial x and y co-ordinates have been changed to (0.01, 0.01) to avoid a division by zero on start-up.

4.6 Localisation and World Modelling Summary

The NUbots system of localisation and world modelling based on the extended Kalman Filter performed well in 2003. A large number of enhancements were made, having an integrated, 5^{th} order model for the robots own location and the ball location. The localisation challenge was performed with essentially no changes to the regular game code for localisation, and we appeared to be 'close' to all required target points (though not close enough to score full points). In an idealised setting, the same algorithm gives positional accuracy of a few cm, and heading accuracy of within a few degrees in self localisation.

5 Locomotion

This section is not intended to describe precisely how the locomotion module functions. Most fast walking teams utilise a method similar to the one described in [9].

5.1 2002 Locomotion

While it is generally true that the best teams have the best locomotion, this was not true of the NUbots in 2002.

Due to time constraints, the NUbot's 2002 entry used a modified version of the University of New South Wales's (UNSW) 2001 walking module, known as Pwalk (for Parameterised Walk). In part due to a lack of understanding of the module, and in part because of a failure to significantly improve it, the NUbots had comparatively primitive locomotion in 2002.

5.2 2003 Locomotion

Development of a brand new locomotion engine for RoboCup 2003 started around October of 2002. The poor performance of locomotion in 2002 was a large motivating factor in attempting to design a superior module for 2003.

The new locomotion engine was christened *TheStrut*, and made use of an elliptical locus. It was considerably faster than the 2002 module. The speed was deemed more than adequate, as it seemed unlikely that many teams would be faster. Furthermore, it seemed almost impossible to improve (despite countless hours of tweaking) and the method by which UNSW had achieved their higher speed in 2002 was not known at the time.

During early 2003, development on *TheStrut* continued. The kicking system was entirely rewritten for quicker kick development and more flexible kick execution. Kicks are now loaded from a file (rather than compiled into code like many teams), and this file can be reloaded at any time. This allows new kicks to be developed extremely rapidly. Interpolation was added in order to better tweak the speed of different parts of a kick motion. This resulted in a greater variety of reliable kicks. Interpolation was also added between certain motion transitions in order to make the robot move more naturally.

Despite the large improvements over 2002, serious problems were encountered with *TheStrut* during the 2003 Australian Open in May.

Firstly, it was almost impossible to make the new locomotion engine line up the ball properly. The workaround was to slow the walk down considerably on approach to the ball. The slow down required was quite substantial, and robots often lost one on one races for the ball because of it. Many days spent tweaking walk parameters failed to alleviate this problem. Even more concerning was the high level of robot dependence exhibited by the chase code: Parameters that appeared to work perfectly for one robot would often fail completely on another. There was also the observation that the robots appeared to chase the ball more effectively after operating for awhile (that is, when they were warmer).

Secondly, NUbots were beaten by rUNSWift II in the Australian Open competition. This time the score was much closer than at RoboCup 2002. rUNSWift's locomotion engine was still a class above anything possessed by the NUbots.

Much concerted experimentation (and the knowledge that a failure to improve would lead to a certain loss) resulted in a trapezoid based walk which was considerably faster again than the elliptical one used previously. Reworked turning mathematics vastly improved the chasing ability of the robots. In hindsight, it seems likely that there were mathematical errors in earlier versions of *TheStrut*.

The benefits of developing our own locomotion engine were more than just increased speed. The experience and understanding gained throughout development had a significant positive impact on behaviour code. Furthermore, it meant we were able to more easily retune our walk for competition conditions.

Having now used both another team's locomotion engine as well as our own, we strongly recommend that all teams at least reimplement a locomotion engine. The benefits are enormous.

6 Robot Behaviour

The robot behaviour may be logically split into high level and low level sections. We discuss high level behaviours first, as low level behaviours are essentially built to facilitate high level ones. Note that in 2002 there was no distinction made in the code structure between high and low level behaviours. We now believe this distinction is crucial, and it is discussed in Section 6.2.

6.1 2002 Behaviour – Region Players

The NUbots experimented with several different behaviours before the 2002 tournament. At the competition, two similar behaviours were available to the team – a wireless behaviour, and a non-wireless one. For reasons that remain unclear, the enabling of the wireless LAN caused a noticeable degradation in lower level behaviour skills and locomotion. Given this degradation and the relative parity between the two behaviours, it was decided that the best option was to use was the non-wireless region playing behaviour known as *RogueV2*.

RogueV2 utilises a simple strategy of assigning robots to static regional positions, sometimes known as a 'zone defence' in regular sports. A typical team formation is illustrated below.

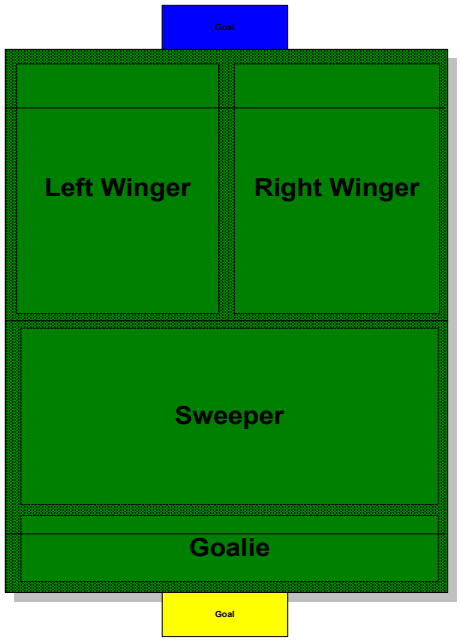


Figure 6.1: Typical team formation for NUbots at RoboCup 2002

For each robot on the team, two rectangular regions are specified at compile time. The *chase area* specifies a region on the field in which the robot will chase the ball whenever the ball is contained within this region. The second region, the *position area*, is utilised whenever the ball is outside the chase area. If the ball is outside the chase area, then the robot will move as close as possible to the ball – but without leaving its position area. Finally, a *home position* specifies a location on the field where the robot will return if it cannot see the ball.

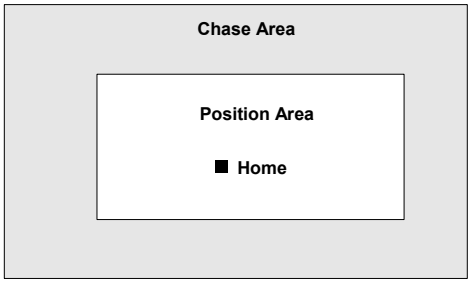


Figure 6.2: The Positional Player Regions.

This simple system helped us achieve third place at RoboCup2002. In particular, this simple scheme in practice gives few conflicts or collisions between multiple robots simultaneously chasing the ball. It also tends to place robots on the field in positions that are often reasonably appropriate tactically. However, it was clear that the system contained many weaknesses:

- There is no attempt made to have the 'best' robot going for the ball.
- Imperfect localization means that boundaries between regions cause either collisions or 'dead spots' on the field (where no robot will go for the ball). This can lead to catastrophic mistakes.
- Static regions imply a lack of adaptability to a given game situation. It is possible to choose between two attackers or two defenders, but the decision must be made at compile (or boot) time.
- The scheme offers no chance of compensation for penalties applied to an individual player (for example, when a robot is removed for 30seconds).

6.2 2003 Behaviour

The NUbots 2002 behaviour architecture had a number of serious shortcomings. It was largely unstructured, and therefore difficult to debug and modify. There were examples of code segments where a single type of action was implemented in different places by replicating code. There was also little or no real hierarchy or philosophy to the code. Last year's code was based around 'personalities' which were selectable only at compile time. Our belief is that the 2002 'personalities' (e.g. *RogueV2*) were inappropriate both in name, and in function, where they mix a range of high level traits (e.g. aggressiveness) and low level skills. We therefore decided that a fresh start was in order, and this module was completely rewritten for 2003.

6.2.1 Shapeless Player (aka Bruce Lee)

High level behaviour is responsible for the overall team strategy adopted by the NUbots. It is concerned with matters such as deciding which point on the field the robot should move to, and if/when the robot should be chasing the ball. With the introduction of wireless communication, high level behaviour has also become responsible for coordinating the actions of multiple robots in an efficient manner. Our different high level behaviours are called 'dudes' (dedicated user defined entities). The primary dude (BruceLee) is described below. One of the architectural improvements in 2003 is the ability to switch between dudes at run-time.

As discussed above in Section 6.1, the main problems with last year's behaviour system were a direct result of its rigidity. Robot positions were static. The robot assigned to chase the ball was based solely on static regions – regardless of whether they could see the ball, and indeed, regardless of their distance to the ball.

As a result, we have tried to make our 2003 behaviour as flexible as possible. We were largely inspired in our effort by the teachings of Bruce Lee [10],[11]:

"The highest technique is to have no technique. My technique is a result of your technique; my movement is a result of your movement."

"Any technique, however worthy and desirable, becomes a disease when the mind is obsessed with it."

"Efficiency is anything that scores."

One instance of applying this philosophy on the NUbot behaviour is that the closest robot to the ball will attack it. Each robot sends its perceived distance to the ball over the wireless network. Other robots will move to an area on the field determined by the captain. It is possible to change which player acts as captain at run-time, but the goalkeeper is typically assigned the captain role.

The captain assigns the chasing robot to the area containing the ball, while the other robots are assigned to other field areas based on their distances to them. These decisions are made using the captain's global world model, although the determination of which robot is currently chasing is made based on wirelessly communicated ball distances. The set of available areas for robots to move to is defined based on the position allocated to robot chasing the ball. If the robot chasing the ball is in the opposition half, there will be two attackers and one defender. If the chasing robot is in the defensive half, there will be two defenders and one attacker.

It is important to realise that there is no explicit negotiation taking place. Since all the robots are equally capable of determining which robot is the optimal chaser, there is no need for an explicit decision making process to take place over the wireless network. This helps considerably in dealing with the latency induced by wireless communication using TCPGateway.

The result is very effective dynamic play in the midfield, while regions are retained only for the penalty box (as required by RoboCup rules). Robots not currently chasing will look for the ball in the direction indicated by their shared world model. They will also move around within their assigned position area based on the position of the ball in order to maximise their chance of gaining possession.

There are limitations with the system, although they are very minor relative to the advantages outlined above. For example, noise present in ball distance information occasionally causes robots to hesitate slightly when deciding whether they should attack the ball (particularly when two robots are approximately the same distance from it). This is very rarely visible and typically lasts for no longer than one half of a second.

Furthermore, if a robot loses sight of the ball at close range, it is likely that a second robot will rapidly enter the fray. This inevitably causes collisions until the non-chasing robot is able to move back to its assigned position. We feel this issue is something of a trade-off in that a less aggressive strategy would prevent his occurring but also perform worse overall. Additionally, there are times when the presence of a second friendly robot is extremely beneficial (as evidenced by the effectiveness of UNSW's play).

6.2.2 Tricks

There were several goals in designing our low level behaviour architecture. Firstly, the ability to share code to perform certain actions (for example, chasing the ball is an exceedingly common task) was deemed an essential feature. Secondly, we sought a generalised mechanism for dealing with actions that require more than one vision frame to complete.

The basis of our solution is the *trick architecture*.

Tricks are essentially actions of varying complexity. They can be as simple as *Step* or as complicated as *ChaseWithKick*. Tricks all inherit from the Trick superclass, and must implement the following methods.

int Start() - Initialises any variables internal to the trick.

int Continue() – Called each vision frame by the trick's owner (can be a dude or another trick). This method tends to handle the main 'work' of the trick. It returns a value indicating the current state of the trick (a value less than one means the trick has completed).

int Abort() – Allows the trick's owner to force the trick to finish

bool IsUsingHead() – Returns whether the current trick controls the head or not.

char GetErrorMsg(int)* – Translates the value returned by Continue() into a string.

char GetName()* – Returns the name of the trick

Each dude has two tricks that it is currently executing: One for only the head, and one 'general' trick. If the general trick is using the head, the head trick will not be executed.

Tricks are organised into "bags of tricks", such as *BallTricks* and *MovementTricks*. These "bags" contain inner classes which are the tricks themselves. For example, *MovementTricks* contains *ChaseBall*, *MoveToPoint*, etc.

There is a special trick named *MultiTrick* which is capable of running an arbitrary number of tricks in sequence. *MultiTrick* can optionally measure the time taken for the trick sequence to execute, making it invaluable for benchmarking and checking for violations of the ball holding rule.

6.2.3 Kick Selection

There was a considerable imperfection in last year's kick selection mechanism. We often used a *flick look*, which involved the robot looking around after it had grabbed the ball to determine which direction to kick. This was very slow, although it almost always resulted in correct kick selection. Unfortunately, opponent robots were given ample opportunity to prevent us kicking effectively. Furthermore, the time taken to *flick look* reduced the amount of time available for turning (keeping the three second ball holding rule in mind).

Our first step was to remove *flick look* and base kick decisions purely on current localisation and vision data (if any). We found that this change produced too many poor decisions in the presence of odometry errors. Additionally, it takes only a very small error in localisation to prevent the robot accurately lining up the goal for a shot.

Advancements in locomotion allowed the robot to turn with its head up while maintaining control of the ball (albeit at somewhat reduced speed), making the aiming of kicks possible in the presence of robot collisions. This immediately suggested another possible enhancement – to delay the actual kick decision until the last possible moment in order to use the most recent localisation information. There are a number of examples of a robot "changing its mind" at the last second during competition: In particular, the 3rd goal in the 3rd vs 4th play-off game against CMPack'03.

After the Australian Open, we decided to reinclude the ability to turn with the head on the ball, mainly because it is roughly three times faster. By using a *MultiTrick*, we were able to seamlessly transition from turning with the head down to turning with the head up. In this way, we were able to turn with the ball very rapidly yet still maintain a high degree of accuracy by looking up shortly before deciding where to kick. We believe it is this feature in particular that allowed us to score so many goals from long distances in the preliminary games and the quarterfinal.

The basics of deciding which type of kick to use and when are simplistic. We use a collection of if statements based on heuristics. Although this perhaps lacks "style", we found it to be the easiest way to guarantee good kick decisions. It is likely that a more sophisticated system will have to be adopted as our high level behaviour is developed.

7 Conclusion

The NUbots 2003 team made substantial changes to all sections of the software, including complete rewrites to substantial portions of the code. These changes can be broadly summarised as:

- Change to a single object software structure, and the incorporation of wireless information;
- Major rewrites to Vision to significantly improve sanity checks and addition of a number of new object recognition features;
- Substantial changes to Localisation and World Modelling to integrate the Ball and Location Kalman Filters, incorporate new object types, and add features for outlier rejection/change detection;
- Complete rewrite of the Locomotion engine with increased speed and agility;
- Complete rewrite of the Behaviour module incorporating both high level and low level behaviours.
- Addition of new debugging and configuration facilities, including the ability to stream vision over the wireless network.

As a result of these changes, we believe we improved markedly on last year's performance, and showed at least competitive performance with all teams in the 2003 competition.

8 References

[1] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*, Cambridge University Press, 1st edition, 2000.

[2] G. Welch and G. Bishop, "An Introduction to the Kalman Filter", Feb, 2001. Available from http://www.cs.unc.edu/~welch/kalman/.

[3] G. Goodwin, R. Middleton, B. Ninness and S. Weller, "Kalman Filters Short Course Notes 2001", Centre for Integrated Dynamics and Control, The University of Newcastle, University Drive, Callaghan NSW 2308, Australia.

[4] W. McMahan and J. Bunting, "So Your Puppy Can't See? Vision Processing for RoboCup '03 Sony Legged League", School of Electrical Engineering and Computer Science, The University of Newcastle, December 2002, http://murray.newcastle.edu.au/users/students/2002/c3012299/

[5] L. Freeston, "Applications of the Kalman Filter Algorithm to Robot Localisation and World Modelling", School of Electrical Engineering and Computer Science, The University of Newcastle, June 2002, http://murray.newcastle.edu.au/users/students/2002/c9806118/main.html

[6] S. Chalup, N. Creek, L. Freeston, N. Lovell, J. Marshall, R. Middleton, C. Murch, M. Quinlan, G. Shanks, C. Stanton, M.A. Williams, "When NUbots Attack: The 2002 NUbots Team Report", http://www.robots.newcastle.edu.au/EE02053.pdf.gz

[7] D. Fox, J. Hightower, L. Liao and D. Schulz, "Bayesian Filtering for Location Estimation", IEEE Pervasive Computing, pp24-33, Vol2, N3, 2003.

[8] A. Willsky and H. Jones, "A generalized likelihood ratio approach to the detection and estimation of jumps in linear systems", IEEE Transactions on Automatic Control, V21, N1, pp108-112, 1976.

[9] B. Hengst, S.B. Pham, D. Ibbotson, C. Sammut, "Omnidirectional Locomotion for Quadruped Robots.", RoboCup 2001: Robot Soccer World Cup V (2002) 368--373

[10] Enter Bruce Lee, http://www.enterbrucelee.com

[11] Bruce Lee's most famous quotes, http://www.fightingmaster.com/masters/brucelee/quotes.htm

[12] Burkhard, H.-D., Düffert, U., Hoffmann, J., Jüngel, M., Lötzsch, M., Brunn, R., Kallnik, M., Kuntze, N., Kunz, M., Petters, S., Risler, M., v. Stryk, O., Koschmieder, N., Laue, T., Röfer, T., Spiess, Cesarz, A., Dahm, I., Hebbel, M., Nowak, W., Ziegler, J. German Team 2002, "German Team RoboCup 2002 Report", October 2002, http://www.tzi.de/kogrob/papers/GermanTeam2002.pdf

[13] N. Chernov and C. Lesort, "Least Squares Fitting of Circles and Lines", University of Alabama, Birmingham, USA. October 2002, http://arxiv.org/PS_cache/cs/pdf/0301/0301001.pdf

[14] D. Marquardt, "An Algorithm for Least Squares Estimation of Non-linear Parameters", SIAM J. Appl. Math. 11, 1963, 431-441.

Acknowledgements

We would like to thank SONY Corporation for their response to our queries about the AIBOs, and for warrantee support. We are indebted to our strategic partners: Centre for Integrated Dynamics and Control (CIDAC), School of Electrical Engineering and Computer Science, and the Faculty of Engineering and Built Environment at the University of Newcastle.