When NUbots Attack!

The 2002 NUbots Team Report

Stephan Chalup, Nathan Creek, Leonie Freeston, Nathan Lovell, Josh Marshall,
Rick Middleton, Craig Murch, Michael Quinlan, Graham Shanks,
Christopher Stanton, and Mary-Anne Williams
Newcastle Robotics Laboratory
The University of Newcastle
NSW 2308, Australia
http://www.robots.newcastle.edu.au

1 Introduction

The NUbots joined the SONY Four 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. The success of the NUbots can be attributed to a highly motivated and talented multidisciplinary team which involved academics, postgraduates and undergraduates from business, computer science, electrical engineering, and mathematics.

In this paper we describe the overall architecture and the individual components of the NUbot robot system. The architecture implements a sense-think-act-cycle that relies on modules dedicated to the main functions, namely *vision processing*, *localisation and world modelling*, *behaviour*, and *locomotion*.

The strength of the NUbot robot system stemmed from (i) an efficient and robust vision module, (ii) an effective localisation and world modelling module, (iii) an aggressive goalie design, (iv) an efficient ball chaser behaviour which formed the backbone of almost all the soccer behaviours, and (v) soccer playing strategies based on strong positional play.

In addition, important elements of the NUbot's preparation were weekly practice matches and participation in the Australian RoboCup Open which was held in the Newcastle Robotics Laboratory three weeks before the international RoboCup 2002 competition in Fukuoka.

We describe the overall architecture of the NUbot robot system in section 2, and in sections 3 through 6 we describe each of the system functional modules. In section 7 we describe the main debugging tools we developed, and conclude with a summary in section 8.

2 The System Architecture

Our system architecture is simple and modular, and as such it facilitated development by a team. Its modular nature also supported effective debugging procedures and debugging tool development.

The system architecture is designed from the perspective of the functions of a soccer-playing robot. This architecture relies heavily on OPEN-R objects (hereafter referred to as objects). The main functions, namely vision processing, localisation and world

modelling, behaviour control, and locomotion are assigned a separate object, and therefore run in their own process space. We adopted a multi-object architecture rather than a monolithic single object architecture which led to a natural division of soccer playing functions, however since each object utilises a separate execution thread, our system is heavily multi-tasked.

As well as the main functional objects there are several other, less critical, objects that control other aspects of the robot such as playing sounds and initialisation.

The architecture can be viewed either from the point of view of a sense-think-act-cycle or as a set of interacting objects. We will briefly describe both views the goal being to develop a rich picture of the system architecture. First however we will give a brief overview of the objects involved in our architecture; each object is examined in detail in the ensuing sections.

2.1 The Objects in the NUbot Architecture

Object	Description
NUICC	Robot motor initialization and robot meta-state (see below)
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

2.2 A *Sense-Think-Act-Cycle* 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 in a serial fashion using an internal model. The relationships between the objects are illustrated in Figure 1 below. The main source of sensory data is the AIBO robot's onboard camera and each cycle is completed once every time a frame is received from the camera, i.e. 25 times per second.

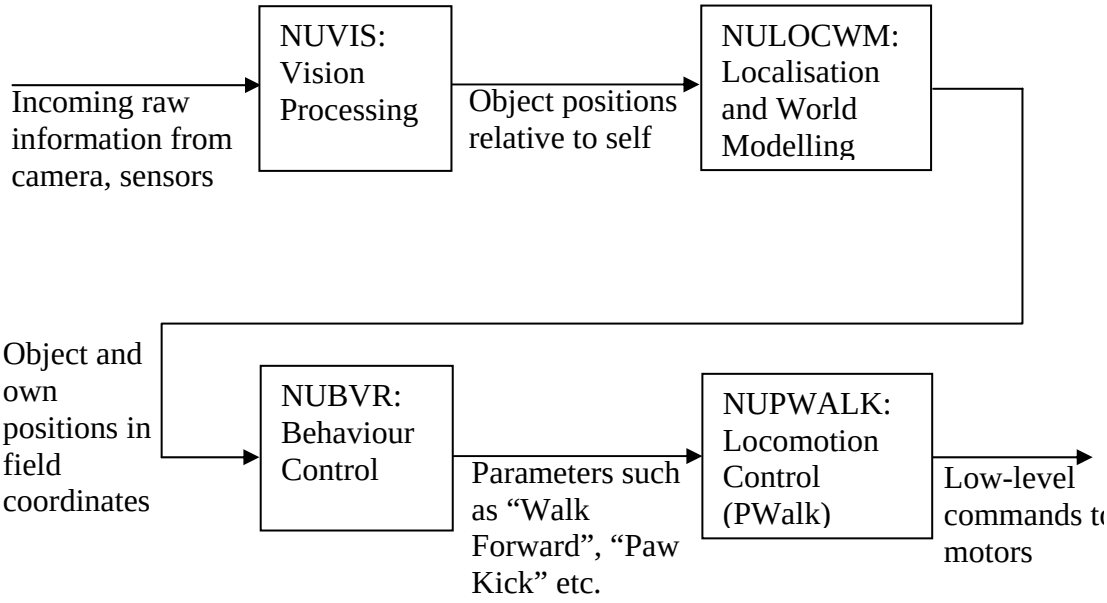


Figure 1: A High-Level Cycle in the NUbot System Architecture

The weaknesses of this architecture are well known. For example, if a single module fails then the system as a whole will fail, and maintaining the world model in NUVIS and NULOCWM is computationally expensive. Moreover the robot is limited in its ability to respond to rapid changes in the environment. Robots playing soccer need to be highly responsive to changes as they operate in and interact with the dynamic field environment. To compensate for the potential failure problems our system has a number of single point failure modes.

2.3 Relationships between Objects

Apart from the fact that there are some objects not represented in Figure 1 (such as NUICC, the initialization and meta-state object) the depicted cycle only represents an idealisation of the relationships between objects. The objects interact in more complex ways than suggested in Figure 1. Sometimes objects require information from objects further downstream (e.g. localisation relies on feedback from the actual robot motion which requires information from either behaviour or locomotion) and sometimes the cycle is not uni-directional (e.g. locomotion may finish carrying out the current action and then request a new action from behaviour even though behaviour has not sent a new action). Furthermore, the simple cycle is an incomplete description because it does not show how the global world model that the behaviour module uses is constructed. In particular, it does not show how information received from other robots over the wireless network is acquired.

Figure 2, below, provides an overview of the relationships between the objects in the NUbot system and the messages that they pass to each other. The object NUICC is responsible for maintaining the meta-state of the robot. It indicates the current high level state of the robot, e.g. initialisation, team red, team blue, ready, play mode, etc. NUICC is also responsible for the joint gain initialization of the robot on start-up as well as recovering the robot when it falls over.

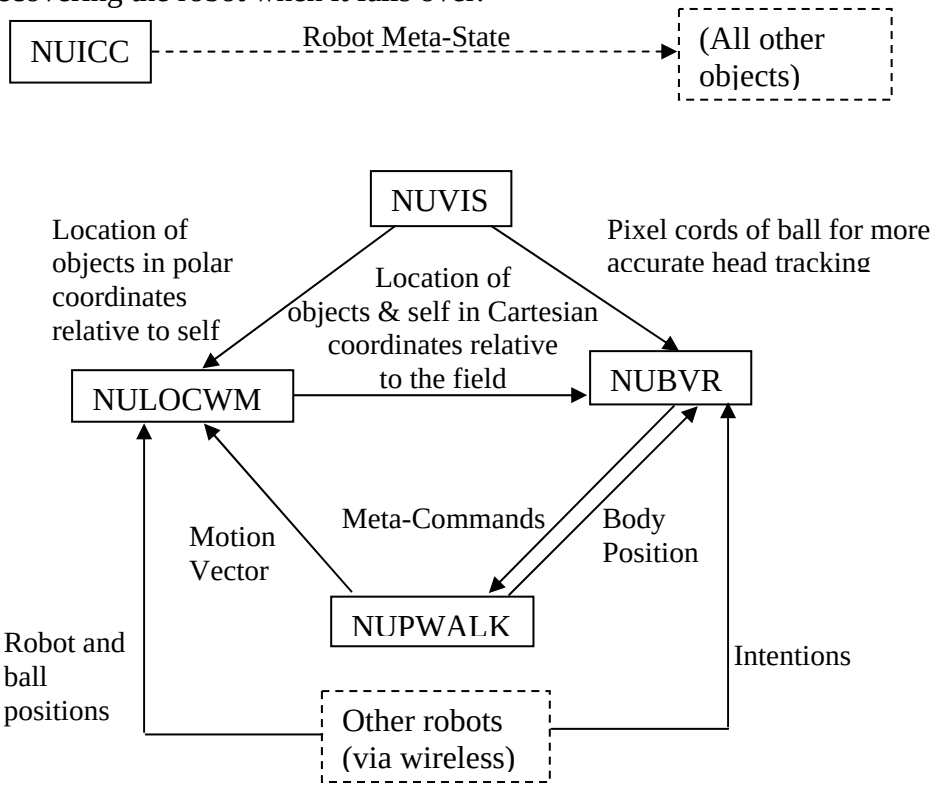


Figure 2: Interrelationships between NUbot Objects

3 Vision

The vision system is charged with converting raw camera images in YUV format into meaningful information about physical objects that the robot "sees". There are a number of sequential steps that are followed in order to meet this requirement. At the lowest level, the raw YUV image is transformed into one containing only the colour information required by the robot to recognise certain objects on the field, e.g. ball, beacons, goals. With this new representation of the image, we perform *run length encoding* followed by *blob formation*. Finally, we use a variety of methods to identify different objects, e.g. ball, beacon, etc, and to determine their distance and angle relative to the robot.

3.1 Low Level Vision

3.1.1 Colour Classification

The lowest level of vision processing is colour classification. We take the high-resolution output of the camera and produce a colour-classified image using table lookup on each pixel.

Note that we chose not to classify *white* or *field green*. Although both of these colours may one day be required for edge detection, we saw no reason to include them for the purpose of RoboCup 2002. Surprisingly, we did not appear to require a sanity check for *field green* to enhance ball detection.

Our lookup table allowed 64 discrete values of each Y, U and V, making it 256Kb in size. We developed two methods for generating the lookup table.

The first lookup table generation method involves manually classifying colours in a large number of raw YUV images and storing the classifications. A utility reads all the classified images and automatically generates the lookup table. For each classified pixel, we add a certain value to the likelihood that the classified pixel is the colour specified. We also examine YUV values 'near' the classified one and do the same thing, except the value added is lower. This nearness parameter (the *splash constant*) can be varied on a per colour basis. Once all manually classified images have been processed, we determine which symbolic colour term is most likely to correspond to each YUV value. However, the value assigned to the likelihood of a certain YUV corresponding to a certain symbolic colour term must exceed a threshold value (which can also be set on a per colour basis) or it will not be classified.

A number of problems arise when using this method, but the principal one is that it is difficult to manually correct a deficiency in the lookup table. In order to fix a problem with one colour, it is necessary to manually classify several more images, tweak several constants, and then rerun the look up table generator. This can be time consuming and could take several attempts to achieve success.

A more desirable method is based solely on a tool called *ImageAnalyser* illustrated in Figure 3, which reads and displays a raw YUV image and then allows the user to select a pixel in it. When a pixel is selected, all other pixels in the image that are near it in the YUV space are highlighted. It is possible for the user to define just what 'near' is in terms of Y, U and V values. The user is then able to map all the YUV values of the highlighted pixels to a symbolic colour term. This mapping is inserted directly into the colour classification table.

Although *ImageAnalyser* does not itself connect wirelessly to the robot, its image loading method allows it to quickly load images streamed from the robot and saved in *RoboCommander* a debugging tool described in Section 7. Combined with the remote control utility also described in Section 7, it provides highly efficient and effective vision calibration.

Another important and useful feature of the *ImageAnalyser* tool is its ability to execute most of the robot's vision system which is described in more detail below.

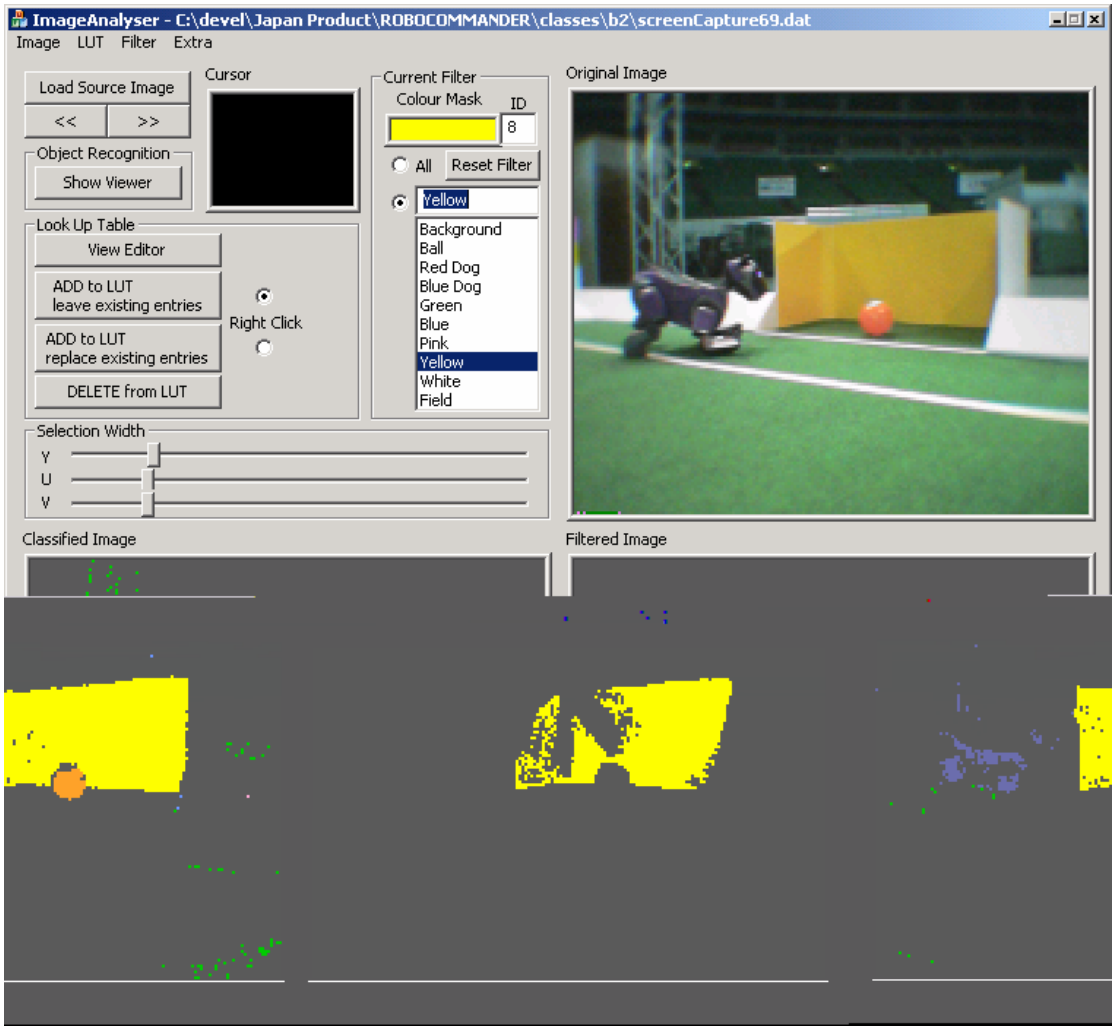


Figure 3: ImageAnalyser in action.

We encountered some difficulties using *ImageAnalyser* at RoboCup 2002 in Fukuoka. Although we do not believe our vision system was seriously affected, we had trouble over-classifying background pixels as *beacon green*. This was in part due to the green seats in the stands; however the green seats did not explain all the observed anomalies, e.g. the classified image in Figure 3 above.

3.1.2 Run Length Encoding and Blob Formation

The colour classified image is run length encoded. The encoding process is the most computationally demanding phase of vision processing, but the algorithm is straightforward. It results in the generation of information about each "run" of pixels. The minimum length of a run is two pixels.

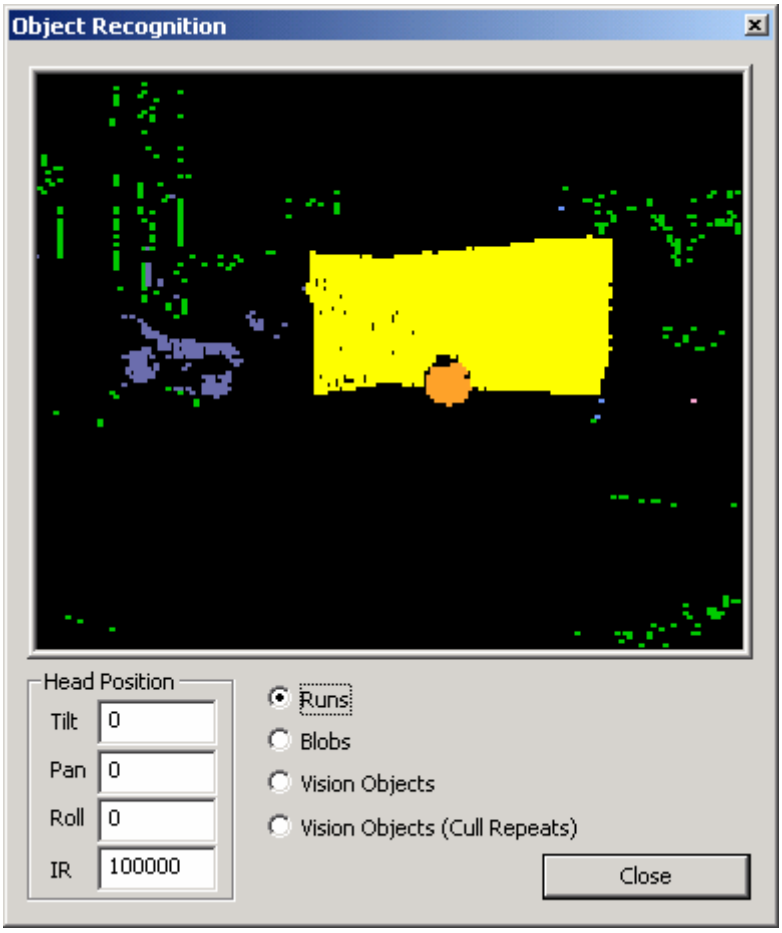


Figure 4: Run length encoded image of the image in Figure 3.

Blob formation is also quite straightforward, although several rarely occurring bugs were only fixed a few weeks before the RoboCup 2002 competition. The algorithm involves simple-overlapping runs on vertically adjacent horizontal lines in the image being merged into a single blob. The process is repeated over all runs, and the result is a set of blobs corresponding to regions of a certain colour. The performance of blob formation was significantly improved by ignoring the colours *white* and *field green*.

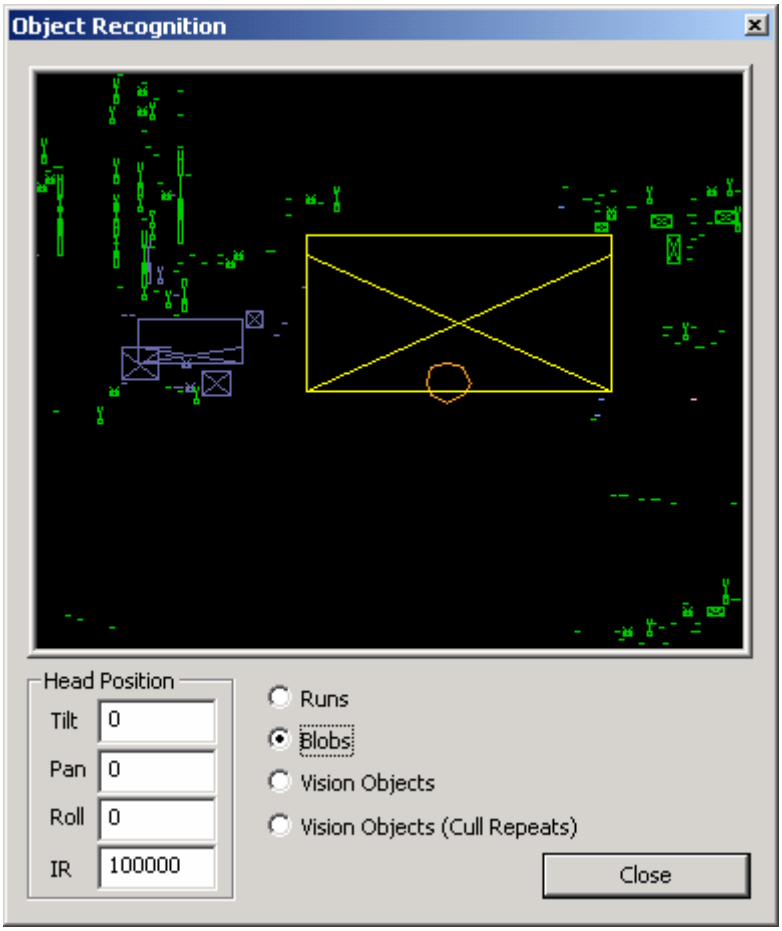


Figure 5: Result of Blob Formation.

There is scope for significant optimisation of our low-level routines, but further improvements in performance did not seem to be required, however we should note that our architecture made it difficult to accurately gauge performance.

3.2 High Level Vision

The AIBO robots possess a single camera at the end of their snout. They are not equipped with stereo-vision, and consequently they cannot make independent and reliable judgements about distances to objects using vision. The NUbot vision system assumes that the dimensions of objects of interest, e.g. balls, beacons, goals, are known in advance. In this way the distance to a non-occluded object is dependent on the size of the blob associated it, in principle.

3.2.1 Ball Detection

In order to identify a blob as a possible ball it had to be at least four pixels high and four pixels wide. We also used an alternate distance measure mechanism when the ball was close to a robot based on three points taken from the edge of the ball blob.

Imperfect colour classification, occlusion, and camera distortion means that no distance measure based on the size of the blob is satisfactory at close range. Instead, it is possible to use the tilt of the head to determine the ball distance. We did not

formalise this method by associating head tilts with distances, however we exploited the head tilt extensively to determine when the ball was close enough to kick.

3.2.2 Goal Detection

The distance to a goal was determined to be dependent on the height of the blob attributed to a goal. We found this metric considerably more accurate than using the width due to the typical occlusion created by the goalkeeper. We also took into account the implicit roll in the image induced by the robot's head being tilted and panned significantly to the side, since we found not doing so led to significant errors in calculating the distance to a goal.

We also had an issue with the bottom half of certain beacons being picked up as goals. By placing a judicious sanity check on goal heights, we managed to avoid this problem in most cases.

3.2.3 Beacon Detection

We had some difficulty with beacon detection due to the skew in camera images, particularly at high head pan speeds.

Beacons are detected when two appropriately coloured blobs are close to each other. However, simple checks relating to the distance between the bottom of the upper blob and the top of the lower blob tend to fail due to image skew. To overcome the problems associated with skew, we calculate the angle between the two blobs and use it as one of the checks for closeness. This method allowed beacon detection to work even at higher head pan speeds.

Distances to beacons were calculated using the distance between the midpoints of the top and bottom beacon blobs. This distance measure turned out to be reasonable accurate and independent of image skew.

3.2.3 Robot Detection

Robot detection – an already difficult task – is made even harder by the colour of the robot uniforms. Under many lighting conditions *uniform red* can appear close to *beacon pink* and it has been noted by other teams previously that *uniform blue* tends to appear exceedingly dark which leads to inaccurate colour classification.

In order to find accurate distance measurements to other robots, we merge blobs of *blue* and *red* into some larger grouping. To do this, we combine blobs that are close to each other in the image. The maximum distance between two blobs under which they are eligible for merging is determined by their size. The larger the blobs, the further away from each other they may be before being grouped together.

Once we have obtained a large rectangle encompassing the robot in the image, we must determine an adequate distance metric. We used a simplistic method which first examined the degree to which the robot was 'side on' or 'front on' by checking the ratio of the width of the rectangle to the height. Combined with this knowledge, we determine the distance using the size of the rectangle.

3.3 Final Result

As noted above, we were able to execute our robot's vision module off board on a PC as demonstrated in the images in Figures 3-5 above which show run length encoded and blob formation images. We were also able to execute our higher level object recognition code on a PC. For example, the image in Figure 6 below shows a blue robot, the yellow goal (plus its two posts) and the ball. Distances to each object are shown. Note the impossibility of the distances shown – the goal is detected as being closer to the image than the ball.

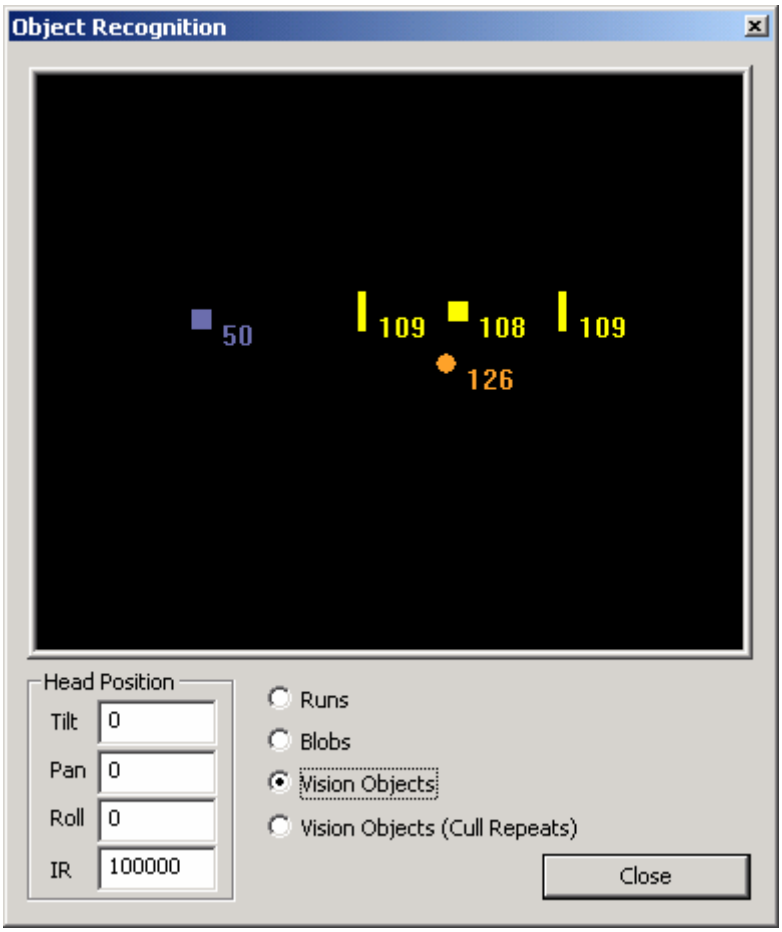


Figure 6: Illustrates Object Recognition in the Vision Module

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 other objects on the field, and combining this information into a world model. For the purpose of RoboCup 2002 the only objects modelled were the robot itself, and the soccer ball.

The main software interactions of this module are illustrated in Figure 7. NUVIS provides vision information (range and bearing) (corrected for head pan, tilt and roll) on a variety of observed objects (beacons, goal 'posts', ball) to NULOCWM. NUPWALK provides odometry information, and NULOCWM fuses this data together with past data to form estimates of the robot location, and heading, as well as the ball location. In the cooperative (wireless) version of the software, the world modelling module also receives the localisation and world modelling information 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 communicated to the behaviour control module NUBVR.

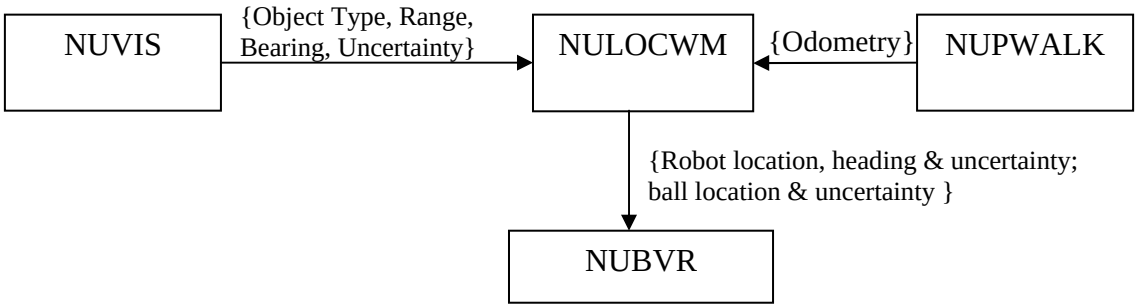


Figure 7: Main software interfaces to the Localisation and World Modelling object (NUPWALK)

4.1 The Kalman Filter Algorithm

The NUbots used 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 [1],[2], and [3] 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 [1]).

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. Linear algebra (matrix multiplications and additions) 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 if two beacons 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) was 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.

4.3 Kalman Filter for RoboCup

Several Kalman Filters have been implemented for robot localisation and world modelling. The robot's own position is modelled by a Kalman Filter and the ball position is also modelled by a Kalman Filter.

4.3.1 Derivation of KF for Robot Own Position (Localisation)

Measurements of beacon distance and angle from current position and locomotion data is used to determine the robot's position on the field. The robot is modelled as a state space model consisting of (x, y, θ); two Cartesian coordinates on the field and heading. The time update of the EKF is completed with information received from the locomotion module about how far the robot has moved. The measurement update is completed using information from the vision module. 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 once every second so essentially the speed in cm/sec or rotation in rad/sec is sent. The forward and left velocities are then resolved into absolute x and y velocities,

based on the heading of the robot. This information is used to complete the time update of the Kalman Filter. Currently the data from the locomotion module is sent after the walking motion has been completed, so the information is slightly lagged.

4.3.2 The Measurement Update

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. The number of measurement updates is repeated according to how many measurements are received.

4.3.3 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 projection can be performed by solving an associated constrained least squares optimisation problem. In this case, we consider only a single linear equality constraint at a time, and the projection is greatly simplified.

4.1.7 Derivation of EKF for Ball Position – Single Robot Measurements

Initially the ball's position was computed using a simple trigonometric calculation on the information from vision. The errors in calculating the position of the ball using this method were rather high, so a Kalman Filter was implemented on the ball to eliminate some of these errors. The ball position Kalman Filter equations are similar to those used for the robot except we included velocity estimation and no heading.

4.1.8 Derivation of KF for Ball Position – Multiple Robot Measurements

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 a Kalman Filter is implemented on ball position measurements from all robots. For this Kalman Filter, 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 implements a Kalman Filter on all the ball (x, y) coordinates it receives (including its own). In this case, the model is linear so the EKF is not needed.

4.4 Implementation and Results

Initially the Kalman Filter for the robot localisation was coded in Matlab^{tm} and tested with simulated vision data. This allowed the code to be easily debugged and the results viewed while the suitability of the algorithm to localisation was determined.

After Matlab simulation the Kalman Filter for robot localisation and ball position was coded in C++ for implementation on the robots.

4.4.1 Debugging: Matlab Simulation – Robot Localisation

In the context of the robots it is exceedingly difficult to observe and obtain information in real time, which makes debugging a complicated algorithm such as the Kalman Filter particularly challenging. Consequently, before coding C++ software for the robots, the application was coded into Matlab and executed using simulated vision data to determine the suitability of the Kalman Filter to localisation and to provide a method of easily debugging the algorithm.

We found that the Kalman Filter estimate is a much smoother estimate than that using trigonometry and the Kalman Filter path follows the actual path fairly closely. In each case the error for the Kalman Filter estimate is significantly lower than that of the trigonometry calculation.

4.4.2 Localisation and World Modelling in C++

The Kalman Filter on the position of the robot and ball is only a small part of the overall world model. As mentioned previously there are a number of objects that can be stored in the robot world model. These are the robot's own position, the ball position, other team member's positions, the opposition team member's positions, the goal position and the beacon positions. Each of these objects is stored with information such as object type, x coordinate (cm), y coordinate (cm), orientation (radians -PI to PI), x coordinate estimate error, y coordinate estimate error and heading estimate error. There are two different world models stored by the robots: the individual world model and the cooperative world model. The individual world model is only based on information available to the robot from its own vision and locomotion, and this world model is sent to all the other robots on the team. The cooperative world model is compiled from the individual world models of all the robots and this is the information sent to the behaviour module.

The individual world model is compiled in the following way. Each robot calculates its own position based on beacon measurements from vision and locomotion data using the Kalman Filter described previously. If the robot can see the ball, the ball position is calculated as the robot's current position plus the ball's relative position using simple trigonometry. If any other opposition robots can be seen these are added to the world model in the same way as the ball, and are clipped inside the field. The individual world model is sent to all the other robots once every ten image frames due to the bandwidth limitation imposed by the competition rules. This allows each robot to have more information by copying data received from other robots into the cooperative world model (particularly the position of the other team members). If a robot cannot see the ball it can use information from the other robots to locate the ball. The cooperative world model is sent to the behaviour module. This world model consists of the robot's own position, team member positions (copied from the data received via the wireless link), the cooperative ball position (which is a Kalman Filtered version of the data received from all the robots), the individual ball position (which is a Kalman Filtered version of data from only one robot) and any opposition robots that any robot can see. The reason for using two different ball positions is that the cooperative ball position estimate tends to jump around as it receives information from different robots who locate the ball at different positions due to natural inconsistencies in their own estimated position and vision ball measurements. This

makes it hard for the robot to kick the ball as it cannot get a good relative ball position estimate. The cooperative ball position is used if the robot cannot see the ball, and the individual ball position is used if the robot can see the ball.

4.5 Kalman Filter Parameter Tuning and Results

4.5.1 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.

4.5.2 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 time lag (the data is not sent until after motion has taken place), slip on the field surface, collision with other objects (walls and other robots) and individual robot motor differences.
- 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 is 60cm even though the robot may not have moved very far.
- 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.

4.5.3 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, Measurement Noise for the ball Kalman Filter can be attributed to:

- Centre finding technique: The centre of the ball is calculated by selecting three points on the circumference of the circle and using circle geometry (perpendicular bisectors) to calculate the centre of the ball. The accuracy of this technique is highly dependent which points are chosen, and bad point selection can distort the measurements.
- Glare: The shiny surface of the ball is highly susceptible to glare, and causes the ball image pixels to be misclassified and hence the distance and angle measurements can be inaccurate.

4.6 Kalman Filter Parameter Choices

4.6.1 Robot 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.

4.6.2 Robot Position Measurement Noise

To determine Measurement Noise, testing was done on the beacon measurements observed by a stationary robot. The head was moved randomly so the beacon appeared in different parts of the image frame and the measurements were logged to a file. This was repeated at five different distances and each time measurements were taken of all six different beacons on the field. The distance error was calculated and it was determined that the standard deviation of distance error (using the statistical standard deviation formula) was equivalent to four pixels variation in the measurement. Therefore, for the distance measurement, the difference between the measurement and the measurement which would be taken if the beacon was four pixels shorter was squared and distance measurement noise set at this value. The angle standard deviation was also calculated by the same method and was found to be approximately 3 degrees.

4.6.3 Robot 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.

4.6.4 Ball Position Model/Input Noise

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. It is assumed that the ball can move 10cm in every second. Although this variable was not 'tuned' as such, observation has determined that this provides a good estimate of the ball position.

4.6.5 Ball Position Measurement Noise

Tests of the measurements of a stationary ball at different distances were done in order to determine the variance of the measurements. A standard deviation of approximately a quarter of the distance was observed for the distance measurement error and the angle error standard deviation was approximately two degrees.

4.6.6 Ball Position Initial Values

These were chosen in the same way as for the robot position.

4.7 Accuracy of Localisation and World Modelling

4.7.1 Robot Position Accuracy

The accuracy of localisation was tested for two situations: when the robot is standing still, and when the robot is following a set path around the field. These are not realistic tests of how the robot will perform in a soccer game as is very difficult to quantify the actual performance of localisation and world modelling for a robot in play, but they do give some indication of how accurately the Kalman Filter can perform.

The robot was switched on in different positions on the soccer field and allowed to localise by panning for beacons. The estimated position of the robot was logged to a file on every vision update and then examined. In some instances the robot could localise to within 1cm of its actual position. In other cases there was over 10cm of error. There was a large variation in localisation depending on position on the field which could probably be attributed to inconsistent lighting conditions.

The other test situation was for a robot following a set path on the field which followed a straight line. The movement data is still relatively accurate, but there are instances where the data moves away from the actual path quite considerably. In this test situation, the robot would walk for a while, stop to obtain measurements and then continue walking. A more accurate result would be obtained if the robot was looking around while it was walking, but the remote control software used to obtain the results was unable to do this. In the actual soccer game, more measurement data would be obtained than in this test situation, and the results would be more consistently accurate.

4.7.2 Ball Position Accuracy

The individual ball Kalman Filter operating on the ball measurements received from one robot has a significant smoothing effect on the ball data received from vision. When the ball is stationary the vision distance measurements can still significantly change from frame to frame, but the Kalman Filter has the desired effect of making the estimate remain stationary. The Kalman Filter can also track a moving ball with high accuracy. Outlier detection has been implemented on the ball so any

measurements that are significantly different from the current estimate do not get included in the Kalman Filter update. At the moment there is no velocity projection on the ball being used. The cooperative ball Kalman Filter provides a fairly noisy estimate of the ball position, but is a useful estimate of the ball position for a robot that cannot see the ball.

4.8 Use of Localisation & World Modelling Information in Team Strategy

The localisation information was used to enable players to position themselves in the correct spot on the field and not to go out of their position boundaries. Heading information was also used to determine which way to kick the ball, as long as the uncertainty was low. We found that the positioning worked very well in the competition. For example, we did not have many illegal defender calls against us. It was also obvious when robots were reaching their boundary and in most cases it was in the expected place.

5 Locomotion

The original solution for locomotion was static and simplistic; have a system of joint angle vectors, and interpolate between succeeding vectors to create motion. Vectors were labeled as stable or unstable, and transitions between motions (such as from a walk to a kick) were only allowed in the stable positions. The process of creating these vectors was primarily trial-and-error, using a protractor on the AIBO.

Commands were passed to the locomotion module using a LocomotionCommand class. This specified the type of movement, and the parameters associated with it.

The LCWalk classes, inherited from LocomotionCommand, allowed speed, direction, and turn and strafe parameters to be obtained.

The process of implementing led to difficulties, primarily in interfacing to OVirtualRobotComm. When one attempted to pass a large number frames, the robot would severely stutter, but when only a few were passed, the motion was extremely slow. The reason for this issue was not uncovered by the time this method was discarded, although it appears that the AIBO robots respond in this manner to joint settings that are grossly discontinuous.

Using this system, a basic walk was implemented, which moved at approximately a centimeter per second; however there was insufficient time to implement a fast walk, along with all of the 'kick styles' that were needed, with only two months remaining until the competition. As a result the LCWalk was replaced by a variant on UNSW's 2001 PWALK [4], encapsulated with an identical interface.

5. 1 Incorporating UNSW’s 2001 PWalk

PWALK [4] was easily incorporated into our design because it was self-contained in one class and fully parameterised. In order to incorporate it we created a layer, NUPWALK, above PWALK that interfaces with the behaviour binary using a locomotion command object that contained parameters PWALK requires. This layer is also responsible for moving the head and making sure the robot returns to a stable position when the state of the robot changes.

Once we had the walk working we were able to tweak the parameters to gain the optimal performance in our laboratory. In addition, we developed a new set of

parameters for each combination of directions (i.e. forward, forward + strafe, forward + turn ….).

At RoboCup 2002 the robots experienced a significant slippage on the competition playing surface which reduced the speed of the walk by approximately 20%.

5.2 Extending PWalk with New Kicks

For RoboCup 2002 we modified one kick and created eleven new motions to the array of actions already contained in the UNSW 2001 PWALK. The kicks described below are those that are used in our games:

Paw Kick

Kick 8 was modified to incorporate a lean back then push forward to increase the power of the kick. The angles of the front paws were also tweaked to increase its reliability. Robots used this kick as our main way of advancing the ball directly up the field and it turned out to be an effective way of scoring

Head Kicks (Kicks 10 and 11)

The robot raises its head straight up, then moves its head in a circular motion while bending its knees to hit the ball exactly 90° to the left or the right. A robot used this kick to clear the ball off the wall when it was sure that the ball was locked between itself and the field boundary.

New Grab (Kick 15)

The action of Kick 9 was slowed down to make the grab more accurate and reliable. We also kept the robot head down to stop the ball popping up during a grab motion.

Test for Ball (Kick 16)

This is used as a sanity check after the grab to ensure the ball is still in the robots' grasp. This move bends the front knees of the robot and squeezes gently on the ball, if the ball is not present the joint will go further then expected indicating that the grab failed and that the ball is no longer in the grasp of the robot.

Spin Kick (Kicks 19 and 20)

This kick is used to whisk the ball 90° up the field. The motion in this kick is the same as a turn except that the last move is modified to make the front paws propel the ball forwards instead of keeping it under its chin. This was an effective and frequently used kick. It enabled the robots to quickly change the direction of our attack and move the ball up the flanks.

Paw Lift (Kick 22)

The robot lifts the ball about 1 – 1.5 inches off the ground in the hope of raising the ball over the limb of an obstructing robot.

6 Robot Behaviour

In this section we describe our high level robot soccer playing behaviours. The behaviours are divided into those that were used when the robots were not communicating via the wireless network and those which were evoked when the robots had access to a global world model that they developed collaboratively by communicating over their local area network. All the robot behaviours relied on a basic ball chaser and were guided by a strong commitment to positional play strategies.

6.1 Positional Play without Wireless

The NUbot system utilised a simple strategy of assigning robots to static regional positions at RoboCup 2002. Figure 8 illustrates a typical team formation.

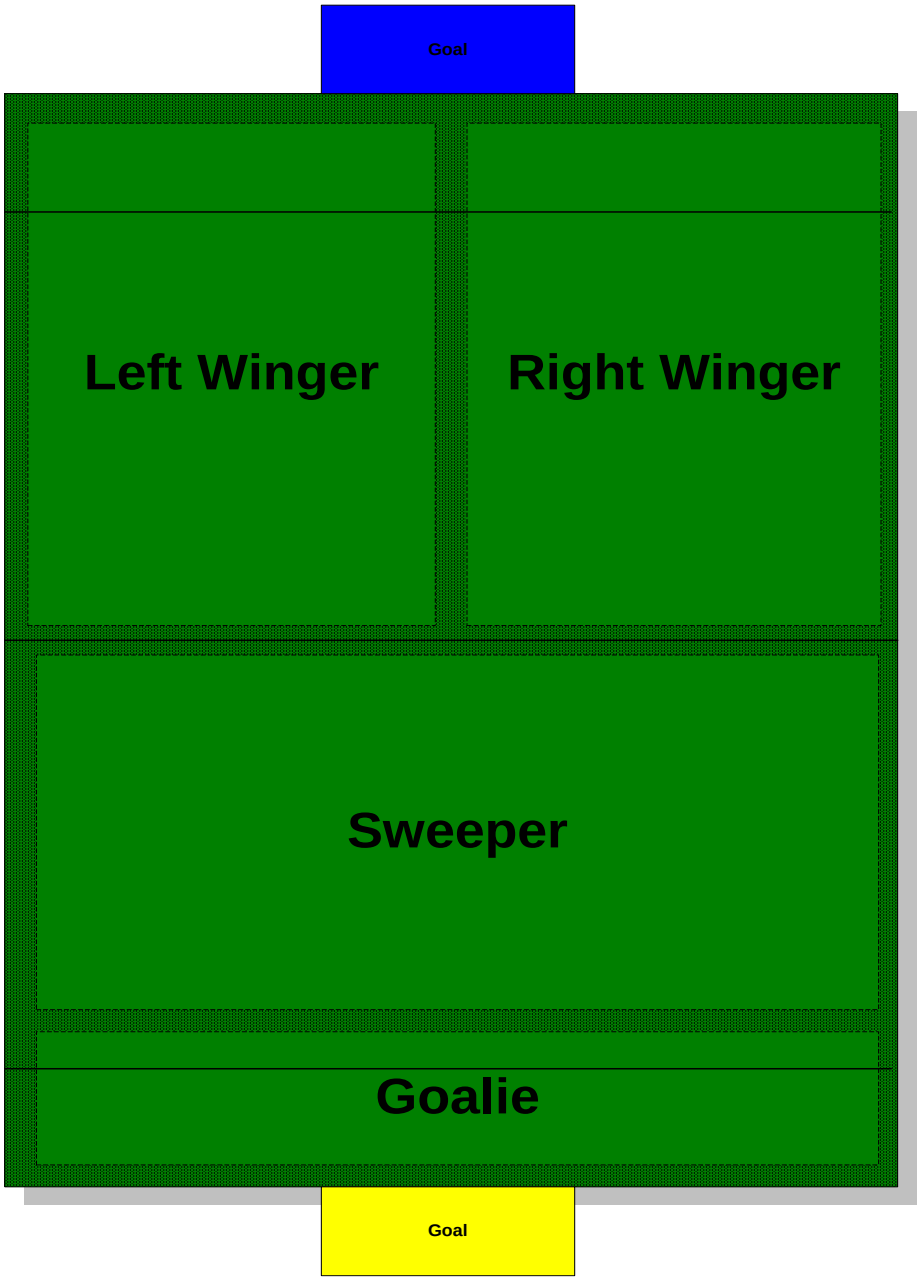


Figure 8: Typical Team Formation for NUbots at RoboCup 2002.

For each robot on the soccer team two rectangular regions are specified at design time. The *chase area* specifies a region on the field in which the robot will chase the ball whenever the ball is in 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. Lastly, a *home position* specifies a location on the field where the robot will return to search for the ball. Figure 9 illustrates the position player concept.

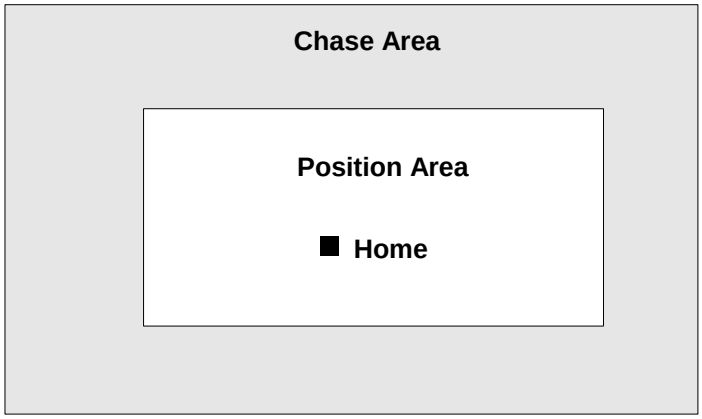


Figure 9: The Positional Player Regions.

The position and chase areas, together with the location of the ball, are used to decide upon a course of action. A decision tree, Figure 10, describes the process.

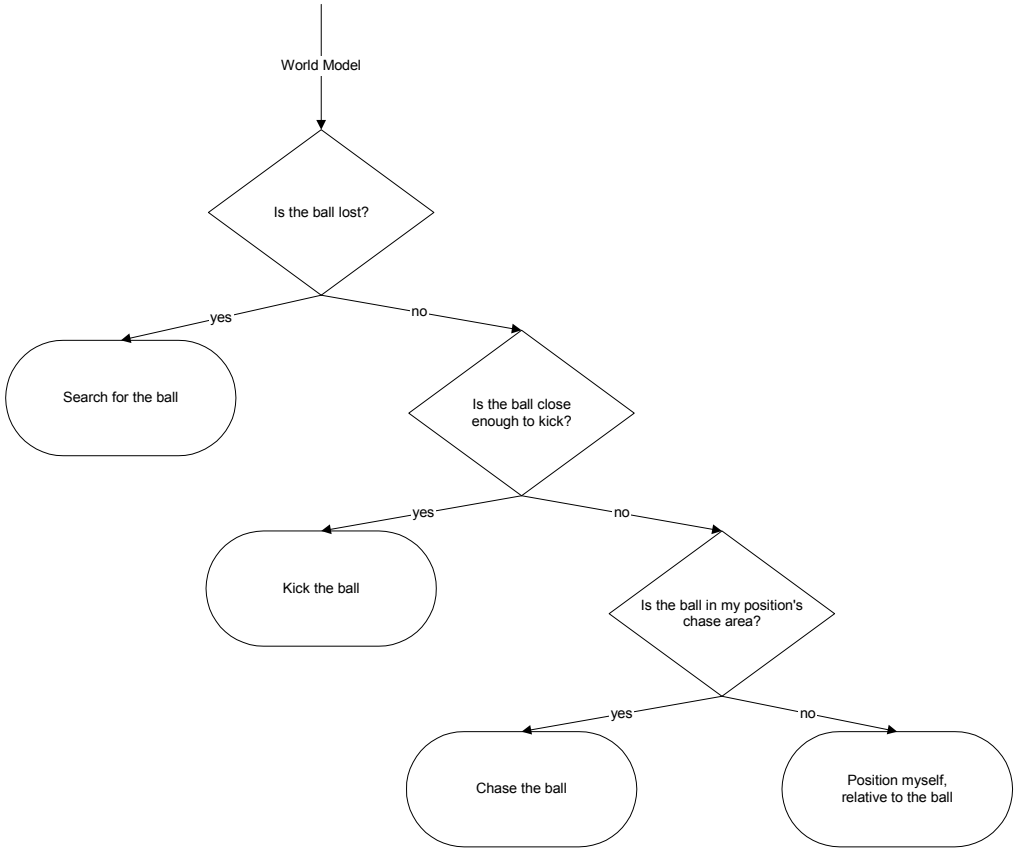


Figure 10: A decision tree illustrating the action selection process of a position player

As shown above in Figure 10, four different states were identified; *Searching*, *Kicking*, *Chasing* and *Positioning*.

6.1.1 Searching

Two different search strategies are utilised, depending upon whether wireless communication between robots is operational. When wireless communication is functioning, the robot utilises shared world model information and looks for the ball in the area of the field indicated by the shared world model. In the case where wireless communication is not functioning, the robot is reliant upon its own vision module to find the ball. When the ball position is unknown, i.e. lost, the robot first searches in its current location by spinning on the spot. If it fails to find the ball, then it will return to its home position. Once the robot has returned to its home position it continues to search for the ball by turning on the spot.

6.1.2 Chasing

The NUbot ball chasing behaviour was inspired by human soccer playing strategies. It involves a simple process of turning to face the ball, and then moving as quickly as possible towards it. While chasing the ball the robot will:

- Never take its eye (or camera!) off the ball;
- Use head pan angles to decide whether to turn left or right;
- Use head tilt angles to judge the distance to the ball when the ball is close;

6.1.3 Kicking

The NUbot kicking strategy revolved around grabbing the ball, localising, choosing the best kicking option, and then kicking the ball. This process mirrors a simple strategy that humans typically use when they play soccer.

When the uncertainty of localisation is low, the robot may choose to shortcut the grab motion and simply kick the ball immediately.

6.1.4 Positioning

If the ball is outside a robot's chase area, then the robot will move as close as possible to ball, but without leaving its position area. If the robot is outside its position area, it will return to it while facing the ball at all times.

6.2 Positional Play with Wireless

6.2.1 Avoiding Ball Control Conflicts

We used a simple class called BallControl that is responsible for determining which robot should proceed to the ball if multiple robots are ready to. The decision process relies on a combination of priority levels based on home positions and the distance a given robot is from the ball.

The BallControl class resides on each robot and contains a variable for control request and distance to the ball for each team member. If the ball is visible in the vision frame and it is deemed to be inside the robots chase area then the value for control request is incremented, otherwise this value is reset to 0. After every 10 frames of vision the

current data is incorporated into the wireless packet sent from NULOCWM and transmitted to the other robots on the team.

When a robot decides to chase the ball it will first request control of the ball, and then it will ask for permission to chase the ball. Requesting control first guarantees that even if permission to chase the ball is not given the system will still recognise the fact that this robot would like control of the ball.

The algorithm for determining which robot has control is as follows:

- If you have a higher priority and the difference in distance to the ball is small then you will be given control.
- Or if you have a lower priority but you are significantly closer to the ball then you will be given control.

In addition to negotiating ball control this class also provides a mechanism for each robot to know if the wireless network is up and running, and if another robot is off the field penalised. A count of how many vision frames since the last message from each robot is kept, if this count goes past 50 (approx 2 sec) then that robot is regarded as off the network until a packet is received again (at this stage control request is reset to 0). If a robot is penalised then it sets control request to –1. If a robot goes off the network while penalised its control request variable will be reset to 0, this is because the behaviour might adapt to handle penalised players which would require the knowledge of when the robot returns to the field. Therefore we decided that other robots would prefer to think that the penalised player is no longer penalised and is simply on the field but not on the network.

Priority was assigned in a back–to–front manner, meaning that the keeper had the highest priority followed by the sweeper then the right attacker and finally the left attacker. We decided on this order of priority because the robot behind the ball is generally in the best position to attack it.

6.2.2 Regional Player

Our fully parameterised wireless player led to a simple model of behaviour which allowed us to construct and customise our keeper, sweeper and attackers from a single behaviour. This class has the following 30 parameters:

homeStrafe_;
homeForwards_;

Step size in each direction to take when return home. Generally the smaller the step the more accurate your movements but the slower they are.

homeStrafeError_;
homeForwardsError_;

The number of centimetres in each direction that is the acceptable error for returning home.

homeX_;
homeY_;

homeHeading_;

The home position and the angle the robot should be facing.

chaseDistance_;

The distance to the ball in which the robot would begin to chase the ball.

faceDistance_;

The distance to the ball in which the robot would stop returning home and face the ball.

faceType_;

The face type of the robot; there are 4 different types :

A. FT_NOTHING

Do nothing just continue as you are.

B. FT_FACE

Turn and face the ball.

C. FT_STRAFE

Strafe until your X is the same as the ball's X.

D. FT_POSITION

Position yourself between the ball and a point on the field (facePosX_ & facePosY_)

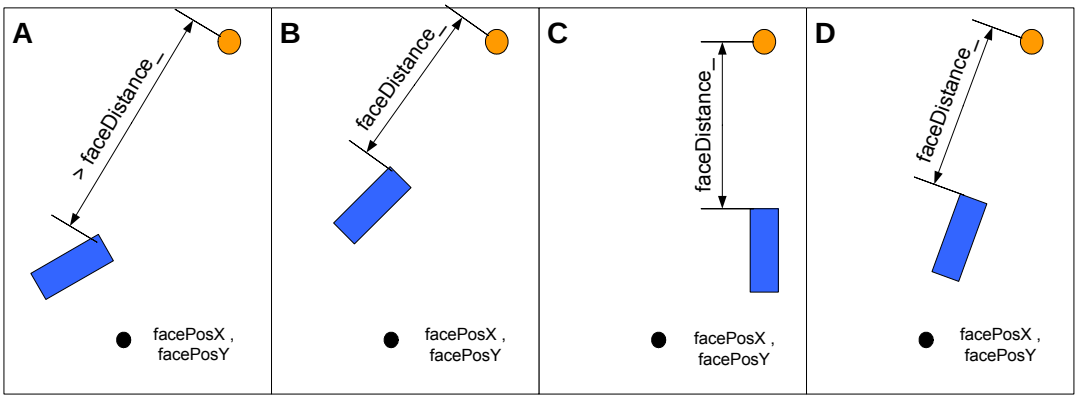


Figure 11: Region Player Face Types

waitSinceLastBall_;

Number of vision frames since you last saw the ball until you change state and begin to return home.

waitAtHome_;
atHomeLocalise_;

If atHomeLocalise_ is true then after waitAtHome_ number of vision frames attempt to re-localise and make sure you were actually at home.

wirelessDistance_;

Distance from the ball at which you would check to see if another robot has requested control over the ball.

upperX_;
upperY_;
lowerX_;
lowerY_;

The ball must be inside these bounds for the robot to chase it.

homeUpperX_;
homeUpperY_;
homeLowerX_;
homeLowerY_;

Instead of using an error range and a static home position, this will allow a robot to return to a home area. The robot will generally approach the bound of the area that is closest to the ball.

turnAtHome_;

If true the robot will only turn and search for the ball once it gets to its home position.

posErrorX_;

Strafe error when you face the ball

positionType_;

Where you should stand when someone else has control

PT_SIMPLE

Back up to positionDistance_

PT_POSITION

Move to a set position (movePosX_ & movePosY)

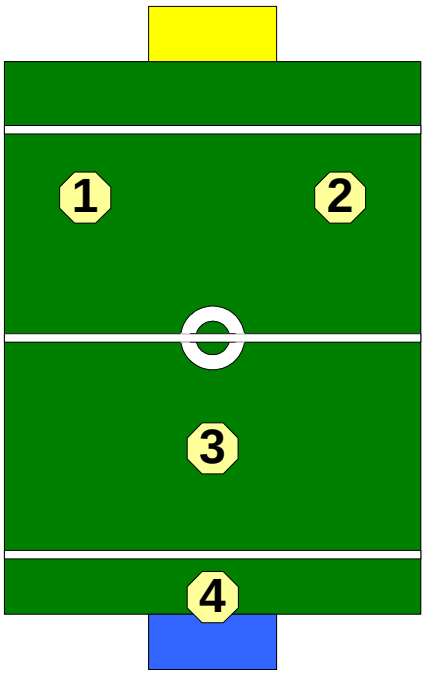
positionDistance_;

The distance to stand away from the ball when another robot has control

movePosX_;
movePosY_;

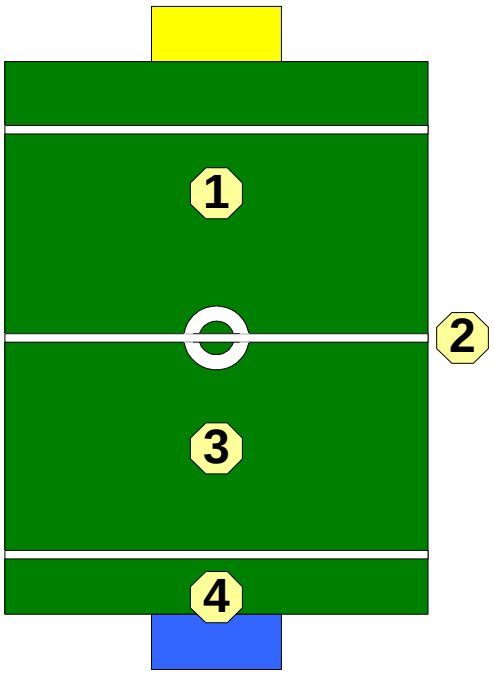
The x & y position's to move to when another robot has control

6.3 Adjusting for Penalised Players



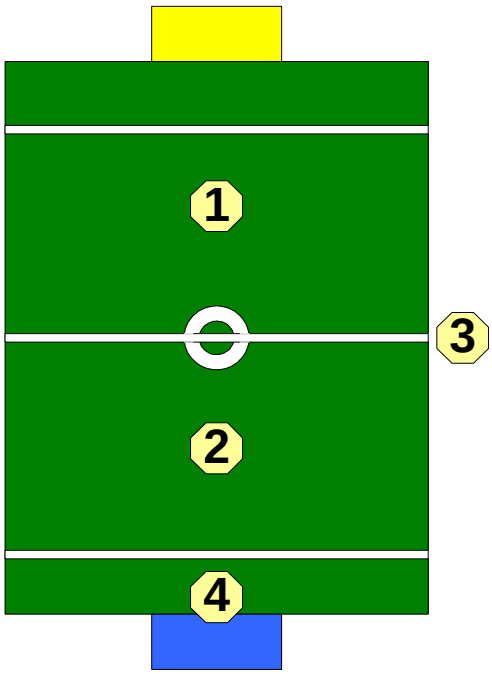
Initial Positions

The numbers indicate each robot's priority level.



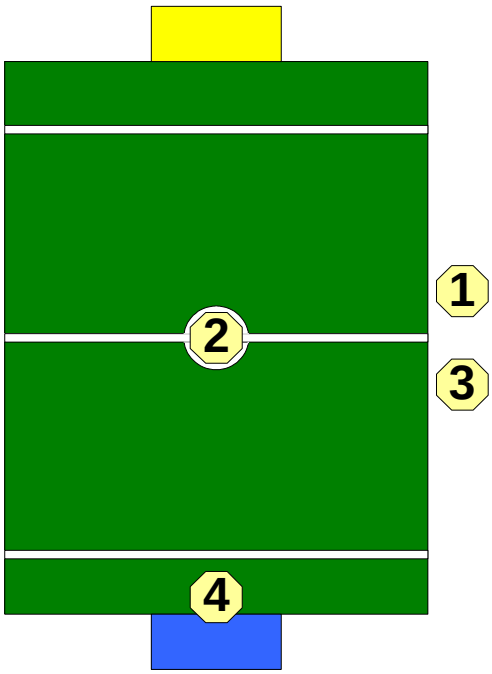
An Attacker is Penalised

The Non-penalised Attacker will change his position bounds to be the width of the field.



Sweeper Penalised

The Attacker with the highest priority (2) will adopt the bounds of the Sweeper. Robot with priority 1 will then change his position bounds to be then width of the field.



Multiple Robots are Penalised

The remaining robot (ie nonkeeper) is allowed to roam the entire field (except behind the defensive penalty line). If only the keeper is left then it will remain solely as a keeper.

7 Debugging Tools

A number of different debugging tools were created during the led up to the RoboCup 2002 Competition. The debugging tools consisted of two separate programs. The first program was RoboCommander, which provided an interface to the robots. RoboCommander could only connect to one robot at any given time, but it could easily change between robots. RoboCommander could also connect to any Gateway anywhere on the robot local area network.

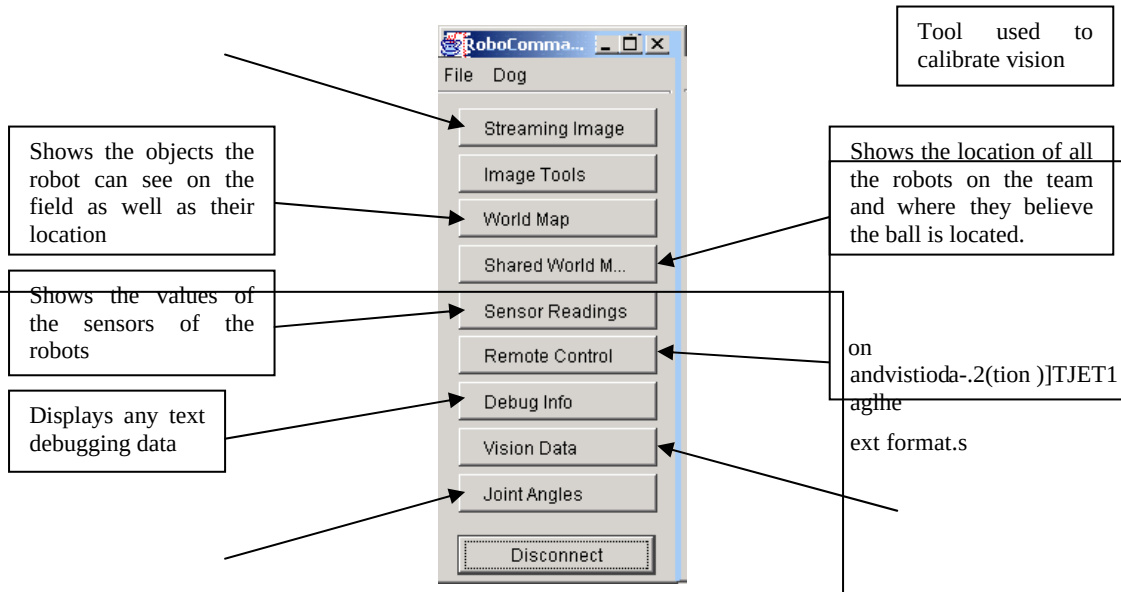


Figure 12: RoboCommander Control Panel

The second program, Gateway, provided the link between the robots and RoboCommander. It received data directly from the robot and formatted it so it could be passed through sockets to RoboCommander. Gateway could connect to any number of robots or instances of RoboCommander.

7.1 Tools for Vision

Several images were useful for visualising the data feed from the robot camera, calibrating vision, and debugging.

We developed a simple visualisation tool that displays images using data from a robots camera. Figure 13 shows the YUV image streamed directly from the robot's camera, and Figure 14 depicts the corresponding Classified Image which shows what the robot "sees" in the image after pixels have been classified into the prespecified range of colours.

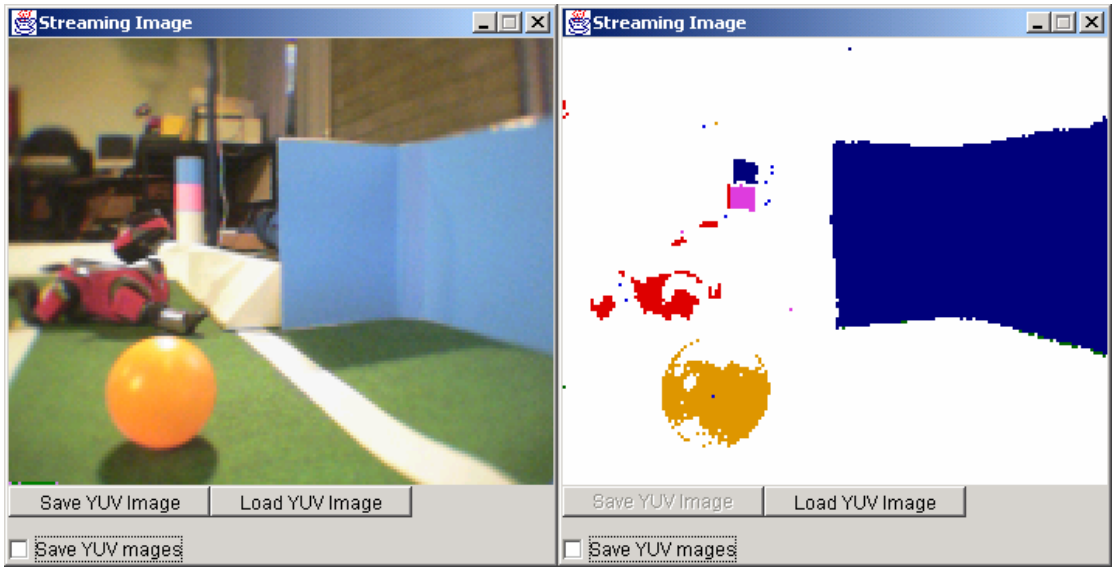


Figure 13: A YUV image streamed directly from the robot's camera.

Figure 14: A Classified Image

A tool was also created for vision calibration. It involved taking a YUV image from the robot and classifying each individual pixel according to its appropriate colour. The YUV values were then taken from the image and added to the colour table. See Figure 15 below.

Figure 15: Tool to support vision calibration of YUV colours

7.2 Tools for Localisation and World Model

Two tools were developed for displaying a robot's position on the field. The individual world model is displayed and shows the location of the robot on the field, and the location of the ball and opponent robots (if they were "seen"). The angles and distances to beacons and goals were also displayed when the robot "saw" them. The shared world model showed the location of all robots on a team, as well as where they believed the ball was located and the location of other robots if they could be seen.

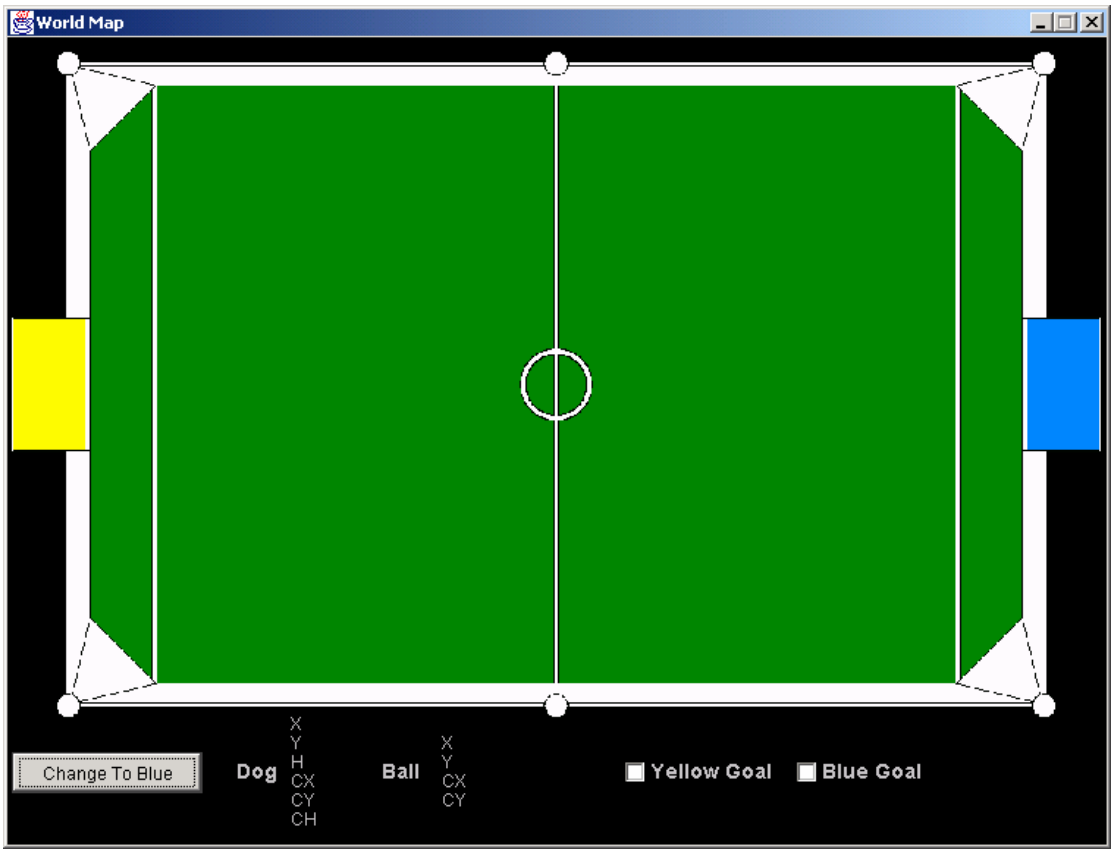


Figure 16: World Model Display

7.3 Tools for Viewing Sensor Data and Joint Angles

We developed a tool that displayed the values of the sensors and joints, see Figure 17 below. The joint angles are shown in either radians or degrees, and can be logged to a file for later viewing. The values were updated every 1.25th of a second.

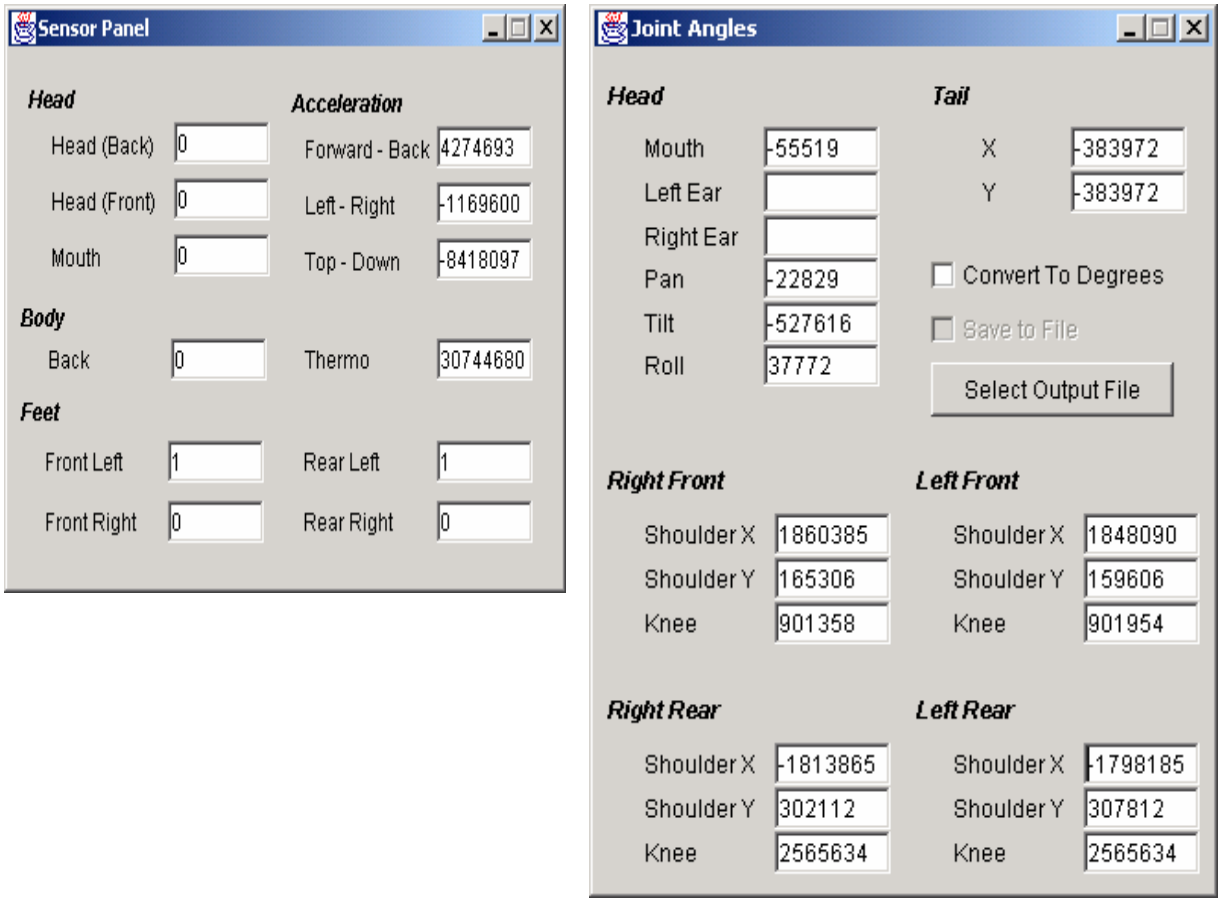


Figure 17: Viewing Sensor Data and Joint Angles

7.4 Remote Control Utility

The remote control facility proved to be exceedingly useful, e.g. it could be used when the robot vision system was not functioning, and in practice games. It allowed full control of the robot. The user was given full control of the head, a variety of different walks and turn speeds, as well as most of the kicks. The remote control utility was used to assist vision calibration and to fine tune localisation.

8 Conclusion

We described the overall system architecture, the individual components of the NUbot soccer system, and some debugging tools that proved to be useful during development. The system architecture implements a sense-think-act-cycle and uses functional modules for vision processing, localisation and world model, behaviour, and locomotion. We designed and developed our own vision, localisation and world modelling, and behaviour modules from scratch, and built on previous work for locomotion by modifying and extending the 2001 UNSW parameterised walk.

Our vision subsystem was simple and robust; it only classified colours found on the ball, the beacons, the goals, goal posts, and robots. Localisation and world model used a Kalman filter approach, and relied on wireless communication between robots to build a global/team world model. The behaviour module was built from a simple, but effective ball chasing behaviour which used neck angles to locate the ball. Higher level soccer playing behaviours and strategies were based on configurable positional play that allowed the goalie to roam and defend at significant distances from the goal mouth if required.

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] Chen, S., Siu, M., Vogelgesang, Yik, T.F., Hengst. B., Pham, S.,B.,, Sammut, C ., The UNSW RoboCup 2001 Sony Legged League Team, October 2001. http://www.cse.unsw.edu.au/~robocup/2001PS.zip

Acknowledgements

We would like to thank SONY Corporation for ensuring that the robots were shipped to us in Australia promptly, and for their rapid response to our queries about the software. We are indebted to our strategic partners: Centre of eCommerce for Global Business, Centre for Integrated Dynamics and Control (CIDAC), Lund University Cognitive Science, Newcastle Business School, School of Electrical Engineering and Computer Science, School of Mathematical and Physical Sciences, and TUNRA at the University of Newcastle Australia. We would also like to acknowledge our gratitude to the local Human Soccer Team, Newcastle United, and Strathfield.com for their support.