Object Recognition in Robotic Soccer

Steven Nicklin

A thesis submitted in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Computer Engineering at The University of Newcastle, Australia.

2 November 2005

Abstract

The NUbots compete annually in the RoboCup competition and have done since 2002. The competition involves teams of four Sony AIBO robots who compete in the game of soccer. To achieve this, work must be done in many areas to produce the final result. These main areas are vision, localisation, behaviour and locomotion.

Work has been undertaken in the vision area to create object recognition algorithms for the NUbots robotic soccer team. Algorithms for identifying the ball and goals, and finding the distance, bearing and elevation of these objects have been developed and implemented on the robotic dogs and were used in the 2005 RoboCup competition.

An overhead camera has been utilised and software has been written to interface it with the debugging program and perform image processing of the incoming data.

Research has been done into generic object recognition. This was for use in the almost SLAM (Simultaneous Localisation and Mapping) Challenge in which previously unknown objects must be used for the localisation of the robot.

The resulting software developed was used in the Australian open competition held at Griffith University on the 25th and 26th of April where the NUbots achieved second place. Then later on at the Robocup competition held in Osaka Japan from 13th - 19th July where the NUbots achieved second place in both the competition and the technical challenges.

List of Contributions

The key contributions which I made towards this project are as follows:

- Developed, implemented and tested sanity checks for ball and goal recognition.
- Developed, implemented and tested distance calculations for ball and goal objects.
- Developed, implemented and tested algorithm to scan images to find edges of balls for use in circle fitting.
- Investigated, implemented and tested method for correcting horizontal skew of image caused by movements of the camera.
- Implemented and tested system for reconstruction of goal objects from multiple blobs.
- Developed and implemented system for determining largest open goal space to assist in shooting.
- Implemented and tested least squares line fitting for use in determining presence of goal posts in an effort to reduce false positive results.
- Developed, implemented and tested object recognition for unknown ‘almost SLAM challenge’ objects.
- Created vision system and interface for use of overhead camera to track locations of objects on the field.
- Additions to NUbots vision debugging program EOTN, in particular sensor dialog.

Signed by: Student: _____

Steven Nicklin

Supervisor: _____

Dr. Lu Gan

Table of Contents

List of Figures

1. Introduction

1.1. About RoboCup

Robocup is an international research and education initiative whose goal is to foster artificial Intelligence and robotics research. To do this the game of soccer was chosen as a way in which different research teams could compete. Its vision is to "By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world cup champion team." There are a number of soccer based competitions including Simulation League, Small-size robot league, Middle-size robot League, Four Legged Robot League and Humanoid League. The competitions held at RoboCup also include RoboCup Rescue competitions and RoboCup Junior competitions[1]. This year RoboCup was held in Osaka Japan during July 13 – July 19.

1.1.1. The Four-Legged League

The Four Legged League consists of four Sony AIBO robots compete in soccer against other teams from universities around the world. The game is played on a field that measures 6m by 4m. The field has white field lines, a blue goal and a yellow goal, and four beacons that are colour coded by location and are used by the robots for localisation. This field is shown in Figure 1.1. The robots play with an orange ball that is approximately 85mm in diameter. During the game the robots must be fully autonomous meaning no interference from outside of the field is allowed either physical or electronic. The individual robots can communicate between each other while playing using a wireless connection [2].

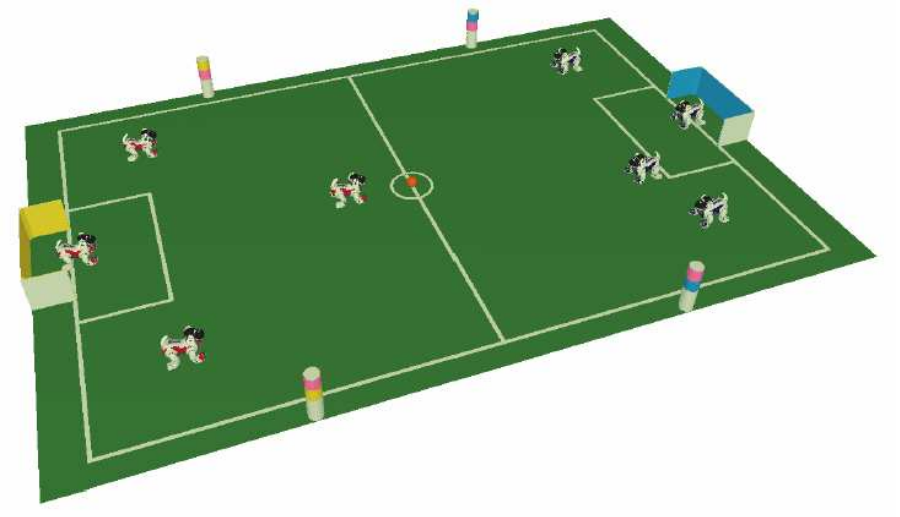


Figure 1.1 - Four-Legged League field, taken from [2]

1.1.2. The Sony AIBO ERS-7 Robot

The robots prescribed for use in the Four-Legged League is the Sony AIBO ERS-7. These robots were originally built for entertainment purposes; but because they provide relatively inexpensive, stable and standard platform for robotics work and are now used in many areas of research and teaching. These robots contain a number of sensors including a 350,000 pixel CMOS camera, accelerometers, temperature sensor, infra-red distance sensors, two microphones and multiple buttons and pressure sensors. The processor is a 64-bit RISC microprocessor running at 576MHz and has 64Mb of SDRAM available to it. The robot also includes a built-in IEEE 802.11b wireless LAN card [3].

Figure 1.2 - The AIBO ERS-7 robot from *http://www.rockingstone.nl*

1.1.3. The NUbots

The NUbots have been competing in the Four-Legged League since 2002. For the robots to play soccer many different software modules are required. The NUbots already had all of the elements required from the previous year, however it was decided that most of the code was to be re-written for the 2005 competition as the code had become overly complicated and difficult to maintain and improve. The NUbots basic software structure for his year can be seen below in Figure 1.3. This software is written using both C++ and Python programming languages. Python is used primarily for behaviour as it allows much faster development. Other elements are written in C++ for greater low-level control and increased execution speed.

The main focus of this report is the vision processing module. The vision pipeline can be seen in Figure 1.4. Firstly the YUV image is classified into more basic colours such as Ball Orange, Field Green etc. During the next stage areas of classified colours are used to form "blobs". These blobs are used to store the location and size of the coloured area as well as other details such as the number of correct pixels. Once these blobs are formed they are filtered and then the object recognition algorithms are run on the blobs. For more information on classification, blobs, blob formation, and related algorithms used by the NUbots, see [4].

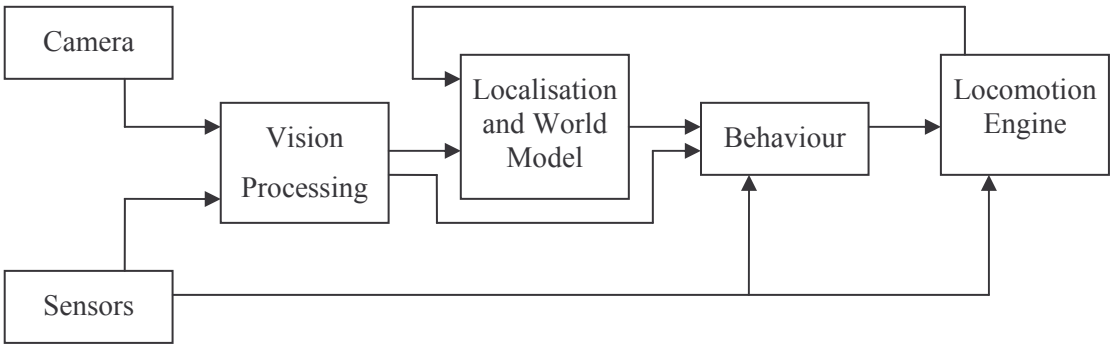


Figure 1.3 – Simplified NUbots software structure

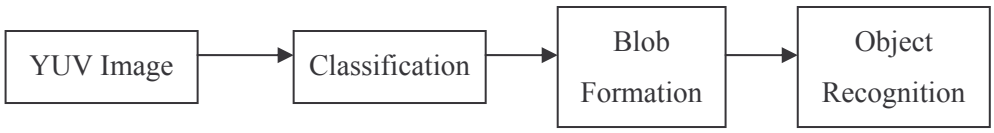


Figure 1.4 – The NUbots Vision Pipeline

1.2. Project Scope

As previously stated the NUbots performed a complete code rewrite for this year's competition meaning new code from scratch for almost all areas. A major part of this project focuses on the Object Recognition component of the vision pipeline seen in Figure 1.4. Recognition of both balls and goals was done along with the calculation of the distance, bearing and elevation of the object in relation to the robot. This recognition relies on the blobs formed on the image.

Other work involved in this project included work for one of the RoboCup challenges involving object recognition. Generic object recognition was required for "the almost SLAM challenge" [3] as the field markers used in this challenge are unknown.

The final part of the project involved implementing an overhead camera to view the locations of objects on the field for use in development with the robots. This involved both interfacing the camera to a debugging applications and creation of the subsequent image processing required for the recognition of objects and calculations of their location on the field.

1.3. Report Outline

This report will first describe the work done involving object recognition on the ball and goal objects. This will include an initial description of the more general object recognition problems and ideas moving on to the objects specific work and the more detailed elements of each.

Following this will be a description of the almost SLAM challenge and the work done for this involving the repeated recognition of previously unknown beacons. Following this is a description of the work on the overhead camera involving camera selection, the interface design, and lastly the implemented vision system.

2. Object Recognition

The Following chapter will give a brief overview of some of the requirements of the object recognition software and general information about the systems used.

2.1. Object Recognition Aims and Challenges

The purpose of the object recognition software is to identify objects in the pictures given and obtain certain data about these objects. In this case the location of the object is required. The major challenge in creating this software is to get the highest possibly true identification rate (objects correctly identified), while keeping the false identification rate (objects incorrectly identified) to a minimum.

True identifications occur when the system is operating as required and is correctly identifying an object as the correct object type. False identifications occur when an object is incorrectly identified as the wrong type of object. Because of the careful colour selection of objects on the field, false identifications more generally occur with objects that are not on the field but rather in the background. These problem objects are generally furniture, walls, spectators, or other equipment.

2.2. EOTN (Eye Of The NUbot)

The EOTN application is used to test the NUbots vision system offline of the robots, increasing the users' ability to test and troubleshoot the vision code. EOTN allows the user to stream images from the robots and store them. It is then possible to move back through these one at a time to accurately tell if the vision code is working. During this project several additions were made to this program in an effort to simplify testing and troubleshooting of other elements of this project. Perhaps the most notable of this is the addition of a sensor dialog allowing the user to view the information of the sensors on the robots in real time during live streaming and later on during playback.

The process of creating this dialog taught me a great deal about the Microsoft Visual C++ dialog code that was later used in the WMD camera dialog. This addition proved particularly useful for viewing the robots pan sensor when investigating and implementing the image skew correction part of this project described in section 0.

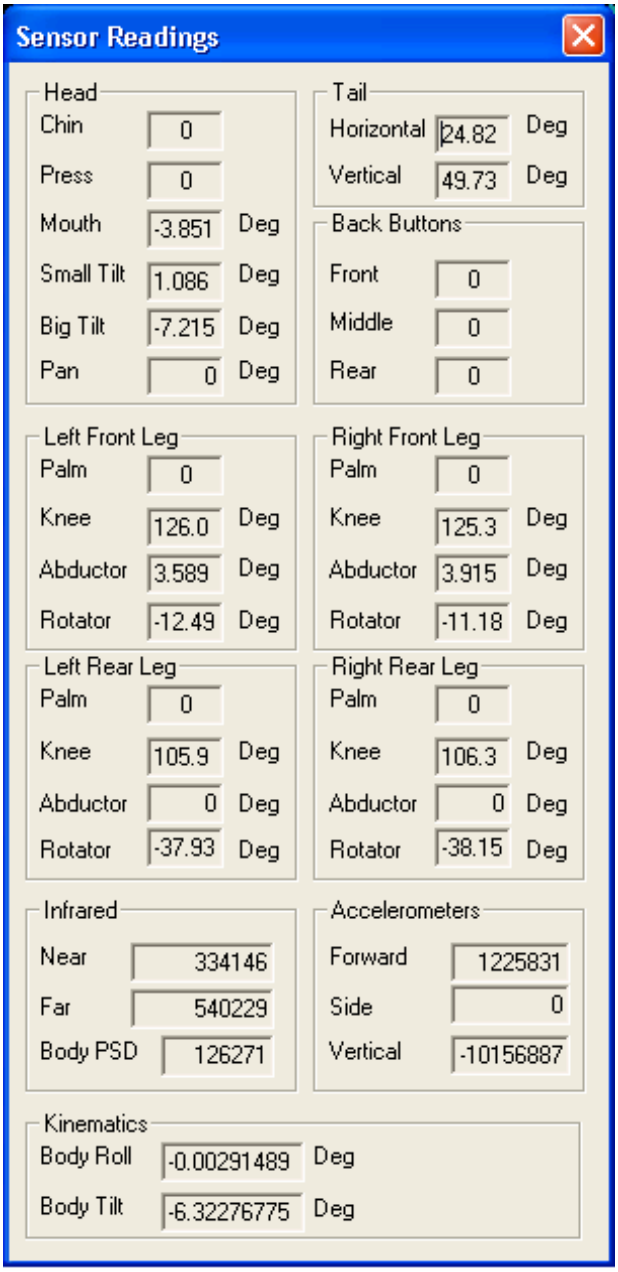


Figure 2.1 - EOTN Sensor Dialog

2.3. The Confidence System

The confidence system works similarly to the sanity factors proposed in [5]. This system is used to allow multiple identification checks to be used to improve simple Boolean yes/no sanity checks used in the past. Using the previous method a failure on any check performed on the object would result on the object being ignored irrespective of the results of previous or following tests.

It is better that the object is allowed to fail one or more of the tests without being immediately rejected as long as it easily passes other tests. This gives an advantage in a few different ways. This method allows objects that may have otherwise have been discarded to be correctly identified if they perform well in other areas, allowing for worse objects to be correctly identified. Another advantage is that the limits of the tests can be restricted as they do not all need to be passed.

This allows more specific tests to be performed that are not crucial for the identification of the object, but can help tip the balance during marginal decisions. The confidence of a candidate object blob starts at zero. When it performs well in tests its confidence is increased and when it performs poorly in tests the confidence is decreased. If at the end of the testing procedure the candidate blob has a positive confidence, it is accepted as the object, otherwise it is rejected and the next blob that is of the correct colour is analysed.

2.4. Distance, Bearing and Elevation

The required measurements of the object include its distance from the robot, its bearing from the robot, and also its elevation from the robot. The distance measurement is calculated differently for each object as the features that can be used to determine this value vary from object to object. The bearing and elevation however are found using a standard method.

This method involves finding the (x, y) coordinates of the centre of the object in the image. The x coordinate is used to find the bearing, while the y coordinate is used to find the elevation. The mathematics used to calculate the angle from the pixel is similar to that shown later in section 3.5.3. These values for distance, elevation and bearing are all relative to the camera. Because the camera can move around, it is required that these values be transformed through a combination of matrix translations and rotations so that they are relative to the more stable body of the robot.

There was an existing function written to perform these calculations called Transform Position, however it was also required for use in C++ and so the function was re-written in C++.

This function takes the raw values of the objects position that are relative to the camera and returns the transformed position that is now relative to the body of the robot.

2.5. Storing Object Data

When an object has been recognised the details of that object must be stored. These include its distance, bearing, elevation, and also the blob or blobs used to form it. Later other information from the localisation and world model code is added such as the x and y values representing the location of the object on the field. Classes were written to store the field objects in both C++ and Python. Some simple class functions such as Set, Reset, and Copy were written as well as interfacing code to allow the field objects to be synchronized in both C++ and Python.

3. Ball Recognition

The location of the ball during a soccer game is one of the most important pieces of information for any player to have as it greatly affects the required actions of the player. A number of different elements are used to get an identification of the ball and also measurements of the balls distance, bearing and elevation.

The elements that are examined before a blob is identified as a ball include the blobs correct pixel ratio, height above the ground, the colour below the ball, and also its roundness. A confidence system is used to bring all of these elements together to come to a final decision. The ball recognition code was created in Python primarily for the reason that code can be changed on-the-fly and uploaded to the robots, decreasing the required testing time.

The following chapter will describe the object recognition software developed for use on the ball, giving a description of each tests motivation and subsequent implementation.

3.1. Ball Distance, Bearing and Elevation

The raw values of bearing and elevation are calculated using the centre x and centre y of the blob in the image, while the raw distance is calculated using the width of the blob in pixels. These values are the values relative to the camera. These values are converted using the transform position function. Circle fitting is applied to a ball to improve the accuracy of all three values [5, 6]. When an acceptable circle is found these values are re-calculated using the new centre x and centre y of the circle to find the raw bearing and elevation, while the circles diameter is used to find the raw distance.

3.1.1. Ball Distance Calculation

The distance to the ball is calculated using the balls diameter physical and the diameter of the ball in pixels. The 'Effective Camera Distance in Pixels' is a value calculated using trigonometry and the field of view of the camera along with its resolution. This value represents the effective distance from the camera to the image in pixels. Its calculation is shown in more detail in section 3.5.3. This can then be used as a ratio to convert the diameter of the ball in pixels and the physical diameter of the ball into the physical distance of the ball as seen below.

$$dis\tan ce(cm) = \frac{Effective\ Camera\ Dis\tan ce(pixels) \times ball\ diamter(cm)}{visual\ diameter\ \ of\ \ ball(pixels)}$$

3.2. Ball Height

3.2.1. Motivation for Test

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

3.2.2. Implementation of Test

The ball height is calculated as the z co-ordinate in the robots 3D space. The z-axis goes from the ground upwards in the positive direction. This z-value is calculated by using simple trigonometry multiplying the radial distance (d) of the object by the sine of its elevation (θ) i.e. $z = d \sin\theta$ this can be seen in Figure 3.1. This z-value is relative to centre of the robot, and as such should be below zero.

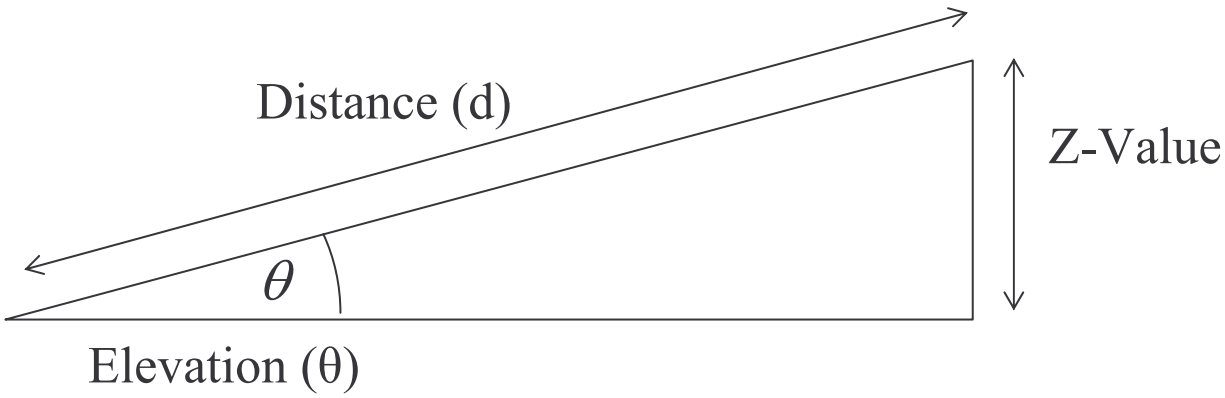


Figure 3.1 - Z-value calculation

3.3. Surrounding Colour Test

3.3.1. Motivation for Test

As previously stated the ball rarely leaves the field during play and most false objects are located outside of the field. From these assumptions testing the colour directly below a blob can be used to determine if an object is on, or off of the field. If the area directly below the blob is field green, then this increases our confidence that the object is located on the field.

3.3.2. Implementation of Test

It is expected that a ball by itself would be sitting directly above the green of the field. However the robots camera is constantly moving and so the bottom of the image is not always downwards. Therefore the area in which the colour is checked must be rotated so that it does appear below the object. This required a rotation of the co-ordinates of the pixel before the test is done. The following formulas were used:

$$x_{rotated} = (x - x_c)\cos\theta - (y - y_c)\sin\theta + x_c$$
$$y_{rotated} = (x - x_c)\sin\theta + (y - y_c)\cos\theta + y_c$$

Where θ is the angle of the rotation to be performed, (x, y) is the original point and (x_c, y_c) is the point about which the rotation is to occur. In this case the rotation occurs about the centre of the blob currently being processed. The theta value used is obtained by finding the angle between the x-axis of the image and a plane parallel to the ground represented by the horizon line that is calculated from the robots current joint positions, see [7], [5] or [8] for more detail. Upon implementation of these equations a problem was found whereby once an area had been rotated it may be partially or in some cases fully outside of the image. This required in special cases, where all of this area is located outside of the image that the colour check simply ignore this test. If this is not the case then the number of correctly scanned pixels (i.e. contained inside of the image) is counted as well as those that are found to be of the correct colour. The results are returned as a percentage ratio i.e.

$$\text{Correct Pixel Ratio} = \frac{\text{Correctly Coloured Pixels}}{\text{Correctly Scanned Pixels}}$$

This method greatly improved the accuracy of the scans as the pixels that were rotated outside of the image were originally included in the total of scanned pixels. The nature of this

test means that it should be able to aid the identification of the ball however should not be able to on its own reject a blob as the ball. This is because of the many scenarios where the green below the ball may not be seen either because it is blocked by another object or it may not even exist if the ball is sitting on a white field line.

Figure 3.2 - Rotated scan area used for ball recognition (shown in blue)

3.4. Circle Fitting

Least squares circle fitting has been implemented to improve the distances, bearings and elevations of balls that are not fully visible in the image. The functions used to implement the least squares circle fitting function in the previous years worked well and so they were not re-written. For more information on the least squares fitting function used see [5, 6].

The existing code is written in C++ and at the early stage at which this was required; none of the previous python code had required calls to C++. Interfacing code had to be written to call the C++ code from python. This was done using the examples given in [9]. The function to gather the points on the outside of the ball was also re-written in order to work with the new code structure. These points are gathered during one of various scan types depending on the parts of the ball that are cut-off. Rules to select the scanning direction to use then had to be implemented.

The scans work by searching from the outside of the blob inwards until it finds pixels of the same colour of the blob, or in the case of the orange blobs also one of the soft colours close to orange. Soft colours are colours that can belong to more than one "basic" colour [4]. The directions for which scans have been created are:

- Left to Right
- Right to Left
- Simultaneous Left to centre and Right to centre
- Top to Bottom
- Bottom to Top
- Simultaneous Top to Centre and Bottom to Centre

Once these points are found they are put into the existing circle fitting function that returns a circle object in the form of an x,y coordinate a radius and a standard deviation for the distance of the points from the circle. If the diameter of the circle is significantly lower that the width of the blob then it is assumed that the circle fit is not correct. In the case that it is fitted correctly the new distance, bearing and elevations are calculated from the circle that is returned. These can be significantly different to the original values calculated from the blob particularly when the image of the ball is heavily obstructed.

Figure 3.3 - Circle fit to the ball can be seen in blue

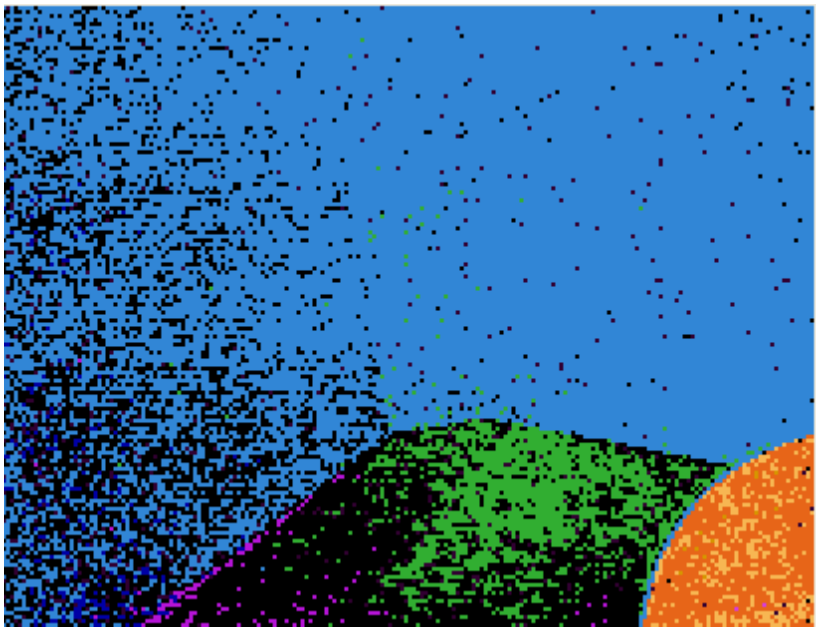


Figure 3.4 - Circle fit of occluded ball shown in blue

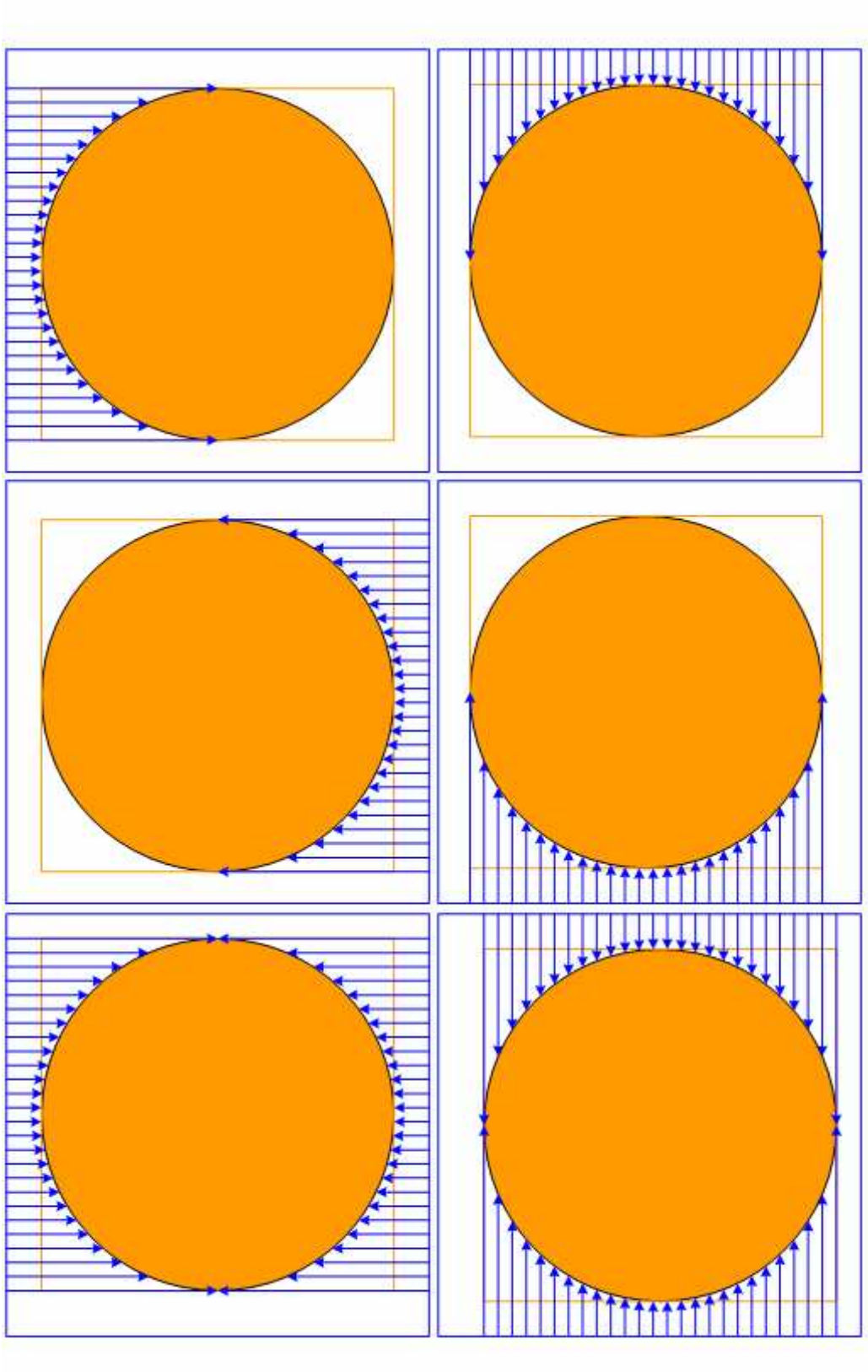


Figure 3.5 - Ball scanning directions, *(top left)* left to right, *(middle left)* right to left, *(bottom left)* left and right to centre, *(top right)* top to bottom, *(middle right)* bottom to top, *(bottom right)* top and bottom to centre.

3.5. Image Skew Correction

3.5.1. Motivation

When the robot pans with speed or the ball is moving sideways, the shape of the ball in the image is distorted as can be seen in Figure 3.6. This distortion occurs because off the way in which the camera works. The pixels are read horizontally from the top left corner to the bottom right hand corner while the pixels are constantly being re-exposed. Therefore in the time between the exposing of the top line of pixels and the exposing of the bottom line of pixels, the balls place in the image may have moved dramatically. This causes problems with the circle fitting algorithm as the ball is no longer a circle, but ellipse.

Figure 3.6 - Ball skew caused by pan speed on stationary ball

3.5.2. Ellipse Fitting

The use of ellipse fitting has been considered by the NUbots in the past, however its side effects have the potential to assume there is relative speed between the camera and the ball when the view of the ball is only obstructed [5].

Ellipse Fitting was re-investigated as a possible solution to the ball skewing previously described. It was also thought that the shape of the ellipse could also be used to determine the relative velocity between the ball and the camera suggested in [5]. However upon further

investigation it was found that a problem arises due to the heavy quantisation of the image in the form of pixelization, and also the expected level of noise encounter in the image.

Mathematically an ellipse can be defined by five points, however because of the quantisation and therefore loss of data this is not correct when it comes to the pixels of an image as each pixel location may represent a range of values. It was found through some analysis and experimentation with the java applet found at [10], that often there a many different possible ellipses that can be fit to the points. Particularly when the full ball cannot be seen, there is a tendency for the ellipse fitting algorithms to form an ellipse that is much smaller than the balls real size as can be seen in Figure 3.7. Because of this perceived problem the use of ellipse fitting was not pursued in this project. For this reason an attempt has been made to instead correct the image before a circle fit has been applied.

It may be possible to place constraints on the fitting of the ellipse to overcome this problem. If this were to be done the method previously developed for skew correction described later in this chapter can be used to determine the amount of skew caused by the movement of the camera. Using this information, along with the skew observed in the ellipse, the velocity of the ball could be determined. This extra data would prove valuable for predicting the path of the ball.

Figure 3.7 - Applet from [10] showing problem caused by an approximated obscured ball in the bottom left hand corner of the frame

3.5.3. Skew-Correction

To correct the skewing of the ball the idea used is to reverse this skew by skewing the image in the opposite direction by an equal amount thus nullifying the skew effect. To find the amount of skew that is present in the image the values available from the pan sensors we logged and evaluated these being the pan angle and the PWM signal sent to the motor. It was found that neither of these provided enough information to predict the skewing of the image. To predict the amount of skew it was found that the velocity calculated from the pan angle values provides the best indicator. Using the velocity the amount of skew can be computed.

To do this first the effective camera distance (d) is calculated using the image width (W) and the cameras field of view (FOV) using the relationship shown in Figure 3.8. This value is also used to calculate the bearings of objects. The values given for the image width is 208 pixels and the field of view 56.9 Degrees [3]:

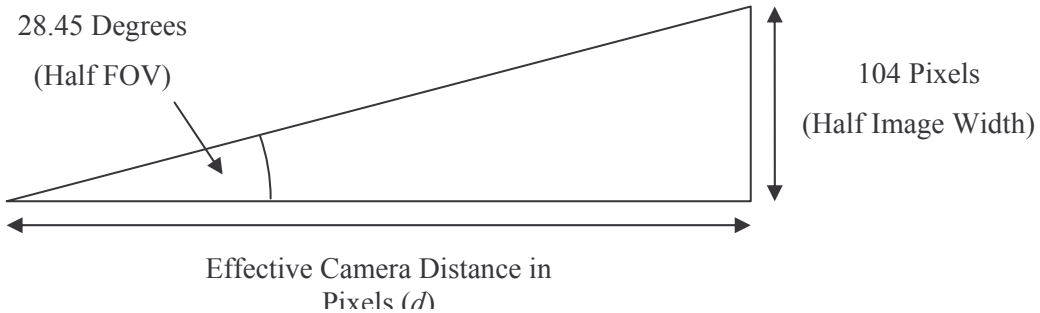


Figure 3.8 - Calculation of effective camera distance (*d*)

$$d = \frac{1/2 \times W}{\tan(1/2 \times FOV)} = \frac{104}{\tan(28.45)} = 191.9 \quad Pixels$$

The effective camera distance can then be used to convert between an angle (θ) and a pixel (x) as shown in Figure 3.9

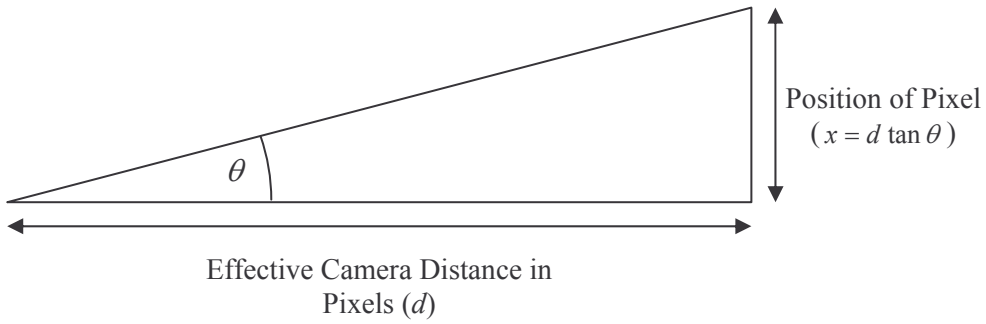


Figure 3.9 - Pixel Calculation using angle

To calculate the movement in pixels, firstly the position of a relative pixel is calculated from the pan of the frame before:

$$x_p = d\tan(\theta_P) \qquad \textbf{Where } \theta_P \textbf{ = Previous pan angle}$$

The position of the current relative pixel is then calculated from the current pan:

$$x_c = d\tan(\theta_c) \qquad \textbf{Where } \theta_c \textbf{ = Current pan angle}$$

The difference between these two pixels is the difference over the entire frame.

$$\Delta x = x_c - x_p$$

The next step was to find the time it took for the frame to be taken so that the distance that the pixel had moved during the capture of the image could be found. Using images of a fluorescent light which flickers at a known frequency (100Hz) the number of light dark cycles appearing on the image can be used to find the amount of time that has passed during the capturing of this image as seen in Figure 3.10. Using the fluorescent lights period of flicker (T_{FL}) equal to $10ms$ and the number of cycles per image (3) an estimate for the period of the camera (T_{cam}) can be made.

$$T_{cam} \approx 3 \times T_{FL} \approx 30ms \approx 33.33ms$$

This suggests that the since the camera operates at 30 frames per second, or a period of $33.33ms$, that the full time between images is used to read the next image. Therefore the time taken between scanning the top right pixel and the bottom left pixel is one full frame.

Figure 3.10 - Image of the fluorescent light shows three distinct cycles

Therefore the Δx value is used as is to calculate the skew amount per line.

$$SkewPerLine = \frac{\Delta x}{numberOfLines}$$

These equations were implemented, and worked reasonably well correcting the skewing of the image of a majority of the images correctly. The major problem found was that other factors would affect the skew on the ball such as movement of the ball. Using this de-skewing technique the correction generally improves the distances returned by the circle fitting when the ball is stationary. One such result can be seen below in Figure 3.11.

This method corrects the skew in the horizontal direction of the frame, however is much more difficult to implement in the vertical direction. This is because the vertical skew is caused mostly by the dropping of the robots body rather than movements of the head and involves the movement of many more joints which can mean increased noise and is also affected by external influences, eg if the robot is kicking the ball it will move at a different vertical speed compared to if there is no ball under it.

The velocity of the ball also causes skew in the image that is not corrected using this method. This feature could possibly be used in conjunction with ellipse fitting to determine the velocity of the ball minus the velocity of the robots head.

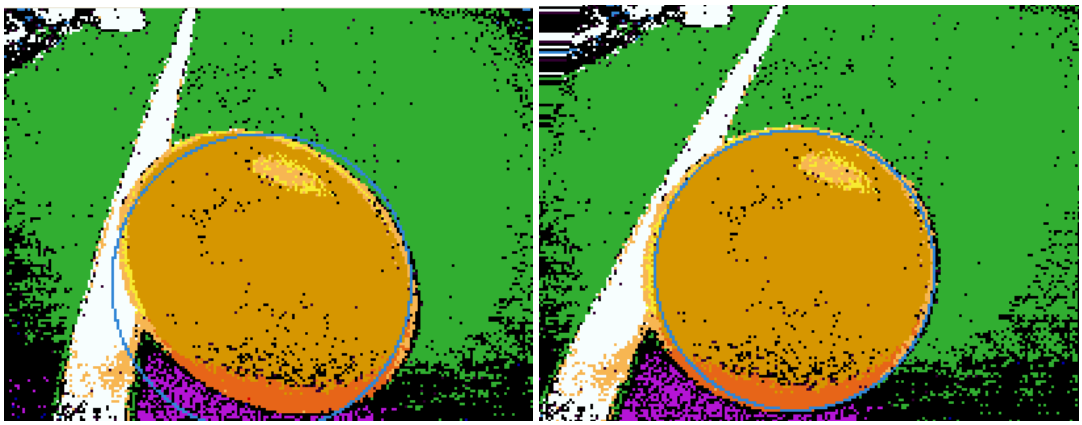


Figure 3.11 – Classified image of ball with resulting circle fit before and after skew correction

3.6. Results

The ball recognition code worked well after a large amount of testing and tuning and was used in the 2005 robocup competition. The modified circle fitting also performed well with the circle fitting greatly improving the calculated values of the balls location. The image skew correction was achieved in the horizontal direction although does not compensate for the movement of the ball, or vertical movements. This area could be improved upon. The investigation into ellipse fitting was relatively short for the reasons given previously however may in the future prove useful in velocity calculations.

4. Goal Recognition

The recognition of the goal is required as the objective of the game is to get the ball into the goals. The goals and its posts are also used for the localisation of the robot. Like the ball it is important that the goal is identified properly when it is seen in the image and it is also important that other objects are not identified as the goal.

Since the introduction of soft colours and due to obstructions on the field, the goal is often not seen as a single blob, but a cluster of blobs. Because of this the blobs must go through a reconstruction process. Once the goal has been reconstructed a series of tests are performed and if the goal has been identified then goalposts can be found.

Goal recognition uses the confidence system previously described in section 2.3. That is as the goal candidate performs well in tests it will increase in confidence and when it performs poorly it will decrease. At the end of the tests if the confidence is greater than zero the candidate is accepted as a goal.

The following chapter will describe the object recognition software developed for use on the goals.

4.1. Goal Candidate Construction

The soft coloured blobs are linked to basic coloured blobs in a structure called a goal object. These blobs are all combined to get the total size of the object and stored as a goal candidate. These objects are then compared and those that are within a specified range, or those that line up horizontally are merged together to create one object as can be seen in Figure 4.2.

One problem that occurs is that when there is a goalkeeper inside the goals it can cause the blob to break into two with a gap at the location of the goalkeeper. To reconstruct this part the goal objects are rotated so that they are parallel to the ground to nullify tilts caused by the cameras orientation. This uses functions described in [7]. Any blobs whose top and bottom co-ordinates are approximately equal are merged into a single goal candidate. The results of this can be seen in Figure 4.3.

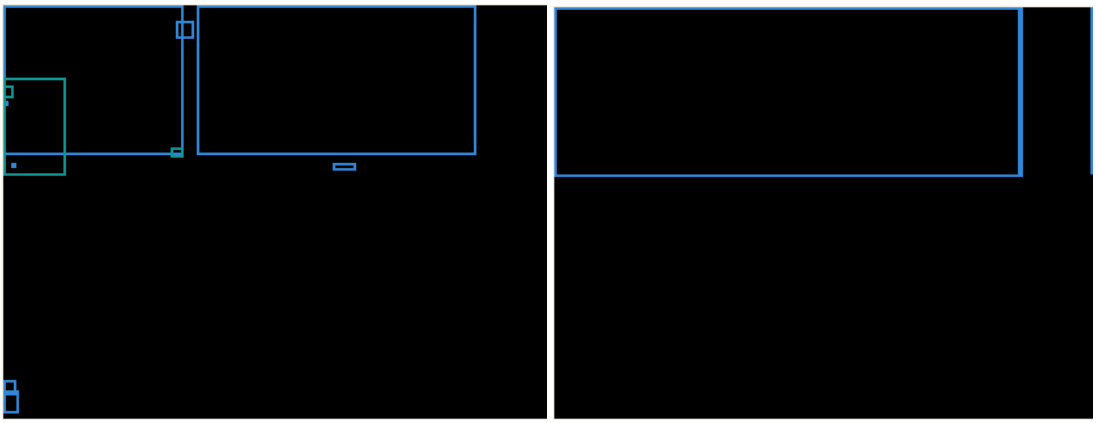


Figure 4.1 - *Left*: Original blue blobs. *Right*: Resulting blue goal blob.

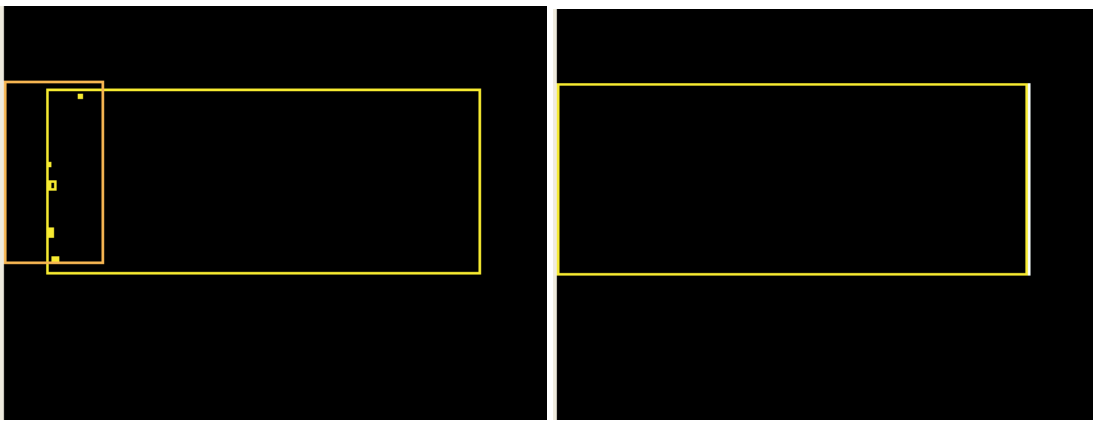


Figure 4.2 - *Left:* Original yellow blobs. *Right:* Resulting yellow goal blob.

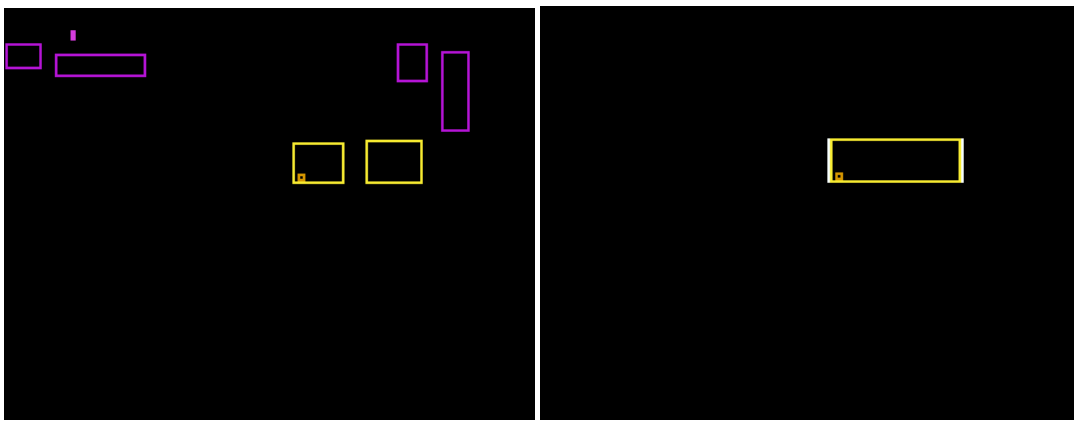


Figure 4.3 – *Left:* Blobs of goal broken into two by goalkeeper. *Right:* Resulting goal blob.

4.2. Goal Candidate Cut-Off Detection

The sides of the goals that are cut-off can give a lot of information that is used in deciding what checks should be performed on the goals as well as in finding the goal posts. To find the edges that have been cut-off by the edge of the screen the blob is first rotated to remove the effects of the camera orientation. This allows 8 points to be selected. These points are the top, left, right, bottom, top left, top right, bottom left and bottom right these can be seen in Figure 4.4. These points are then un-rotated back into the original image to determine if they are close to, or outside of the edges of the screen. A side is seen as cut-off if two or more points on a side are located near the edge, or outside of it. For example if the top point and the top left point is outside or close to the edges of the screen then the top is assumed to be cut-off. This information on the suspected cut-off areas of the objects is used to tell the accuracy of the measurements obtained as well as the presence of goalposts in the image of the goal.

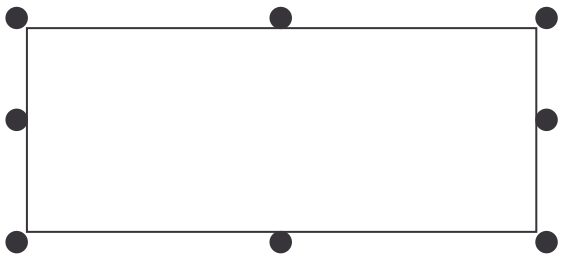


Figure 4.4 – Points of blob used for object cut-off detection

4.3. Goal Distance, Bearing and Elevation

The raw distance of the goal is found by using the height in pixels. This value is the only factor that will remain relatively constant when viewed from different parts of the field. The problem with this, however is that the robot must see the entire goal. When the robot is close to the goals this can be a problem. Because of this the infra-red distance sensors are used to gain a more accurate distance to the goal when the goal is close. The raw bearing and elevation is found using the centre of the goal blob or in the case of the goal posts their centre points. As with the ball these raw distances are translated back so they are relative to the centre of the robot. If the object cannot be properly seen and the cut-off detection has determined that part of the object has been cut-off that is required for correct measurements to be made, then the software sets a flag so that the localisation software knows to give these values less weighting.

4.3.1. Goal Distance Calculation

The distance of the goal is calculated using the height of the blob formed around the goal. To do this an approximate linear relationship was used requiring an offset and a multiplier. This value was recorded for a large range of distances and these were plotted against the actual distance using Excel. Excel was then used to apply the line of best fit to these two sets of data and these equations were used to approximate the linear relationship between the two. Because the height of the goal is used, the distance of the goal becomes inaccurate when either the top or bottom of the goal is cut-off by the edge of the image. In this case a flag is set for the localisation component signalling that the distance given should be taken as a maximum limit on the distance, rather than the exact distance.

$$dis\tan ce = \frac{multiplier}{goal\ blob\ height} + offset$$

When using this method it was found that as the robot moved closer to the goals distance does not remain linear. This occurs because as the robot moves towards the goals the sides of the goals artificially increase the height of the goal. It was attempted to approximate the non-linearity by create two linear relationships, one near and one far, however it became difficult to determine when to use each one and it was found that there was a better way. Due to the introduction of soft colours, it was found that in many cases the sides of the goals were classified as soft colours. Therefore the height is taken only from the yellow blob if the difference in heights is not too great.

4.4. Goal Identification

Once the goals candidates are reconstructed they are then tested for identification. The following tests are only applied to goals that do not cover a large majority of the screen since it can be very difficult to get readings in this situation. For this reason if a goal candidate appears to be a very close it is assumed to be a goal and these tests are skipped since it is very likely to be the goal as there are not usually other objects large enough and of that colour to fill the cameras FOV.

The first test that is performed is a colour test similar to that used in the ball recognition. The test works in the same way as that described in Section 3.3. This test determines that the candidate is on the field.

4.4.1. Elevation and Z-Value test

Because the goals are stationary, like the ball, their height should remain relatively constant. For this reason the z-value is calculated as described in section 3.2.2; this is then compared to a range of acceptable values. If it is not in this range then the candidate is penalised through a reduced confidence. Elevation checks are then performed; this is because the range of z-values for an acceptable goal needs to be quite large to account for incorrect distances and elevations. Therefore the elevation of the objects is used to help remove the other false goals.

4.5. Goal Post Detection

Using the information gathered during the cut-off detection the image is scanned for goal posts. This is done by performing a least squares line fit on the edge of the goals. Firstly the points to be used in the fit are gathered. This is done by performing scans parallel to the horizon at regular intervals along the side of the goal. The least squares line fitting method is then performed by first calculating the sum of squares as this method appeared the simplest to implement as an algorithm. The values used with the sum of squares method are:

$$ss_{xx} = \left(\sum_{i=1}^{n} x_i^2\right) - n\bar{x}^2 \qquad ss_{xy} = \left(\sum_{i=1}^{n} x_i y_i\right) - n\bar{x}\bar{y} \qquad ss_{yy} = \left(\sum_{i=1}^{n} y_i^2\right) - n\bar{y}^2 \qquad [11]$$

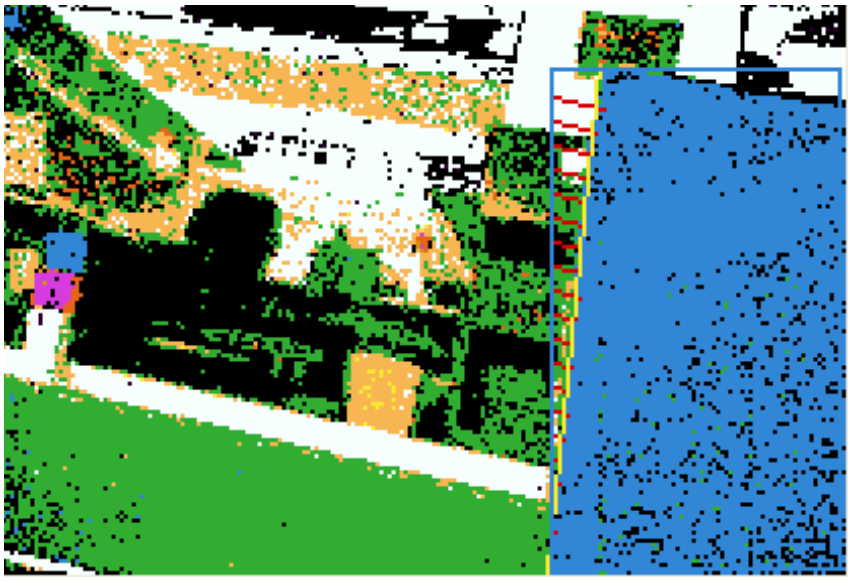


Figure 4.5 - The scanning path for the line fit points are drawn in red, while the least squares line fit is drawn in yellow

These are used to calculate the equation of the line $y = a + bx$. Firstly the regression coefficient b, the correlation coefficient r^2, and the residual variance s^2 are calculated as follows:

$$b = \frac{ss_{xy}}{ss_{xx}} \qquad r^2 = \frac{ss_{xy}^2}{ss_{xx}ss_{yy}} \qquad s^2 = \frac{ss_{yy} - \frac{ss_{xy}^2}{ss_{xx}}}{n-2} \qquad [11]$$

The value of a is then found: $a = \bar{y} - b\bar{x}$ [11]

Where $\bar{y} = \sum_{i=1}^{n} y_i$, $\bar{x} = \sum_{i=1}^{n} x_i$, $\overline{xy} = \sum_{i=1}^{n} x_i y_i$

Once this line has been fit, the correlation co-efficient $\left(r^2\right)$ and the sample variance $\left(s^2\right)$ are used to determine the edges suitability as a goal post. One change that had to be considered was that in the most likely and most ideal case of a goal post being a straight vertical line, the gradient of the fit line would be infinite. To correct this problem the x and y axes are swapped in the line fitting mathematics and a check is done in the case of a completely horizontal line to prevent a divide by zero error that would cause the robot to crash.

The raw distance for the goal is used and new raw elevations and bearings are calculated for the individual posts using their centre points. The detected posts can be seen in Figure 4.2 and Figure 4.3 they are shown as the vertical white lines on the edges of the goals.

4.6. Goal Gap Detection

4.6.1. Motivation

The locations of the goals play a large part in the NUbots kicking behaviour in that before a robot takes a shot at goal it will first try to visually line up the goals. However the centre of the goal (given by the goal detection) is not always the best place to kick the ball. The best place to kick the ball in a majority of cases is at the largest “gap”, the goals largest unobscured area. This allows the shot to go past the obstacles and therefore increase the chance of scoring. To help improve the NUbots goal kicking abilities when faced with a goal keeper or other obstacles, goal gap detection was developed.

4.6.2. Implementation

The gap detection consists of an algorithm that searches from the left hand side of the goal to the right hand side of the goal in the image designated by the current goal blob. The search runs parallel to the horizon and hence roughly parallel to the goals. The height of the scan is calculated depending on the height of the goals in an effort to scan at approximately the shoulder height of another robot, thereby at the height of the largest portion of the robot. Runs of pixels that are of the correct goal colours are counted and the longest run found is given as the gap. This gap is given as a field object and therefore the robot can use this to line up shots.

Figure 4.6 - The scan path for the gap detection is show in yellow

4.7. Results

The goal rebuilding code works well, the major concern was blobs that were not part of the goals being merged with the goal objects; however this did not prove much of a problem. The line fitting greatly improved the accuracy of the post identification, removing many false positives. This was a large improvement over the initial method of only looking for the cut-off parts at the edges of the image. The new method also removes many posts that have objects in front of the posts such as robots.

5. The Almost SLAM Challenge

The following chapter will describe the almost SLAM challenge, and the work done for it during this project involving object recognition.

5.1. About The Almost SLAM Challenge

This challenge is designed to help the league move away from using fixed beacons for localisation and use more general landmarks such as those naturally found in a soccer stadium. In the challenge there are a number of artificial landmarks placed around the field. The landmarks are not known before the challenge. The challenge begins by allowing the robot some time to explore the field with the traditional beacons and goals in place. These are then removed and the robot must then move to set points on the field that are written on its memory stick. It must do this using the information that it has gathered during its exploration. There are a number of constraints on the landmarks placed around the field [12]:

- They shall all be outside the playing area but on the green field.
- They shall be of varying size and color.
- They are guaranteed to be unique when color and orientation are taken into account.
- There shall be at least three of these landmarks containing a patch of pink at least 10cm across.
- There shall be at least six landmarks.
- They shall be at least 15cm apart.
- They shall contain no white or black, although they may have a stand that is black or white.
- Each dimension shall be between 10cm and 50cm.
- They shall be no more than 50cm above the field.

The Almost Slam Challenge requires a large amount of collaboration between both the vision module and the localisation module. The vision module must be able to recognise the new objects that are present and re-recognise them as the same object. While the localisation module must keep track of the locations of the objects and then re-use them as the basis for localisation.

5.2. Coloured Pattern Recognition

The solution developed for this challenge involved a form of coloured pattern recognition. The method used creates a selection of regions of the image centred on the pink blobs, taking advantage of the guaranteed minimum of three objects containing pink. Nine regions in total are created, equal to the size of the pink blob to form a 3x3 cross. Each section of the grid is scanned for the two most common colours in that region. Once these have been

found the colours of each region of the grid is compared to the equivalent region of each of the available patterns for each object that has been previously seen. If a similar pattern with eight or more sections of the grid having at least the most common colour, then the object is identified.

During the initial exploration stage of the challenge the original beacons are used for localisation and so the location of the objects that are seen can be worked out as the robot has an accurate estimation of its current position on the field. In this stage if a pattern is not matched to an object the calculated location of the pattern is then compared with that of the previously seen objects. If it is within an acceptable range to the previously seen object it is assumed to be that object and the patterns are added as an additional pattern for that object. Otherwise the pattern is set as a new object. At the end of this phase the most commonly seen patterns are set as the land marks.

During the localisation stage of the challenge objects are only identified, not created. This means that if a new pattern is found it is ignored. This occurs because the location of the object cannot be accurately calculated as it cannot be guaranteed that the robot can accurately predict its location as was the case in the previous section.

For the module to calculate the distance, bearing and elevation to the objects in the second part of the challenge, the size of the blobs is tracked by the localisation module. During the initial part of the challenge the localisation module computes a linear relationship between the size of the blobs and the distance of the objects. This is then used by the object recognition code when finding objects in the second part of the challenge.

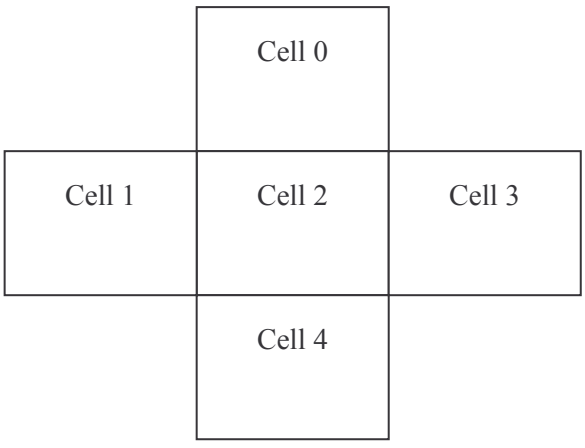


Figure 5.1 - Scanning grid for almost SLAM Challenge

5.3. Results

The results obtained offline using streams of test images featuring the normal beacon objects showed promising results with four different objects repeatedly recognised a large number of times. These being the four distinctively coloured beacons. The system was also tested on the field with the beacon recognition code disabled.

Because the system must work closely with the localisation module a large number of changes were also made to the localisation portion of the code by other members of the NUbots team. Unfortunately during The Almost SLAM Challenge event one of these changes encountered a math error causing the robot to crash, and hence failing the event.

Due to the removal of this event from the challenges program for next years competition and the lack of relevance to real problems faced in this area, no further time was spent on this aspect of the project following robocup.

6. World Model Debugger Overhead Camera

The following chapter will discuss the work done on the selection and implementation of an overhead camera for use in the development of the NUbots Robocup soccer team.

6.1. Overview

This part of the project involves the use of an external camera to view the field and recognise the objects. It can be used to report the "real" location of objects on the field so that they can be compared to what the robot has reportedly seen.

There are many advantages to knowing the real locations of the robots and the ball. It can be used to improve the vision system of the robots, providing easier measurements of the distances of the objects from the robot; these were previously done with a tape measure. It can provide better tracking of speed and direction when developing walks and kicks allowing more flexibility when using machine learning methods. Finally it will allow a direct comparison between the Robots world model data and the real world data (from the overhead camera) allowing the world model data's accuracy to be assessed.

The World Model Debugger application was developed with Kenny Hong, who worked on the applications user interface as well as other elements such as networking, data logging and object prediction. For details on the work done on these elements see [7].

6.2. Aims

The aims of this part of the project were to develop a system that can be used to track the locations of objects on the soccer field via an overhead camera.

The system should be able to:

- Track the location of the ball and robots on the field in terms of the coordinate system used in the robots and display them in the World Model Debugger application.
- Run in real time and be able to store the data collected for later analysis.
- Provide an interface usable by other members of the NUbots team and have its' data on object locations accessible from other areas of the WMD World Model Debugger application.
- Provide an accuracy of around 10cm.

6.3. Camera Selection

The selection for the camera was an important part of this section of the project. There were a number of desired Features:

- Resolution of at least 640x480 for the desired measurement accuracy.
- Able to be permanently fixed to the ceiling.
- Field of View (FOV) large enough to capture full field in a single frame from ceiling (may require attached lens).
- Digital (USB 2.0 or IEEE1394) connection (should not require a frame grabber).
- Should have good picture quality so that objects are easily seen.
- Should be compatible with windows.

During the initial stages there were many different cameras considered. These included two that were currently available to the NUbots team. These were a Philips Vesta Pro web-cam, and a JVC GR-DV3000EA Camcorder. Other cameras that were considered that would have to be purchased included an Apple iSight Web-cam, a Sony HDRHC1 High-Definition Camcorder, and a Basler A601fc Digital video camera. The selection criteria for the cameras are shown in Table 1.

Figure 6.1 - Balser A601fc

The camera eventually chosen for use was the Basler A601fc. This camera is designed for use in machine vision and was recommended for use in this application by a representative of Total Turnkey solutions who specialise in imaging and machine vision equipment. Some of the advantages with this camera were Windows drivers for the camera, a large selection of compatible lenses, a large amount of control over the cameras operation from the pc, and also a compact design. The camera also came with a recommended lens that covered the full field as per the required specifications. This makes it ideal for mounting on the roof as it is light and can be controlled and powered remotely via the IEEE 1394 firewire connection used for the data connection. The interfacing was simplified with the windows drivers available.

Camera	Phillips Vesta Pro	JVC GR-DV3000EA	Apple iSight	Sony HDRHC1	Basler A601fc
Resolution 640x480 or higher	Yes	Yes	Yes	Yes	Yes
Permanent Fixture	Yes	No	Yes	No	Yes
FOV Obtainable	No	Yes	Yes	Yes	Yes
Windows Support	Yes	Basic	No	Basic	Yes
Digital connection	Yes	Yes	Yes	Yes	Yes
Picture Quality	Fair	High	Good	High	High
Other Features Considered	Owned by NUbots	Owned by CDSC	Inexpensive	High Definition	

Table 1 - Camera Features used for selection

6.4. The small-sized league

An application similar to that of the overhead camera can be found in other robocup leagues, one in particular is the small-sized league. This league consists of 5 individual custom built robots. In this league all 5 of the robots share the same vision system which consists of an overhead camera connected to a central PC. This camera is used to see all of the objects on the field, including the other robots and the ball. This application is very similar to that required in this project. For this reason the small sized league was looked at as a base for ideas concerning the equipment and software to be used in this project.

After reviewing a number of reports and publications including [13-15] concerning small-sized league vision systems it was found that by far the most common and effective way of processing the image is through the colour classification method. This is currently the same method that is used effectively by the NUbots in the four-legged league. Due to the extensive use of colour classification in the small-sized league, as well as my familiarity with the colour classification process from working with the NUbots code, it became clear that colour classification would be the most effective and simplest method to implement for this project.

6.5. Colour Classification

The colour classification method of image processing relies on the colours of objects as their most prevalent feature. This means that each object should have a unique colour or pattern of colours that can be used to identify the object from others.

The colours in the image have a large number of different shades and hence individually distinguishable colours. Colour classification is used to group all of these shades into more meaningful classified colours such as red, blue, orange and green. These classified colours simplify the image for the proceeding algorithms by giving the image in terms of more general colours allowing similar colours of varying shades to be grouped and seen as equal classified colours. In the case of this project where a YUV image is the data source, the classification procedure is done by mapping all of the possible YUV values to their respective classified colour groups. This is done via a 3 dimensional lookup table that stores a classified colour for the YUV values. The lookup table must be manually produced using the users' eyes and own sense of colour recognition to determine the colours that belong to the classified groups.

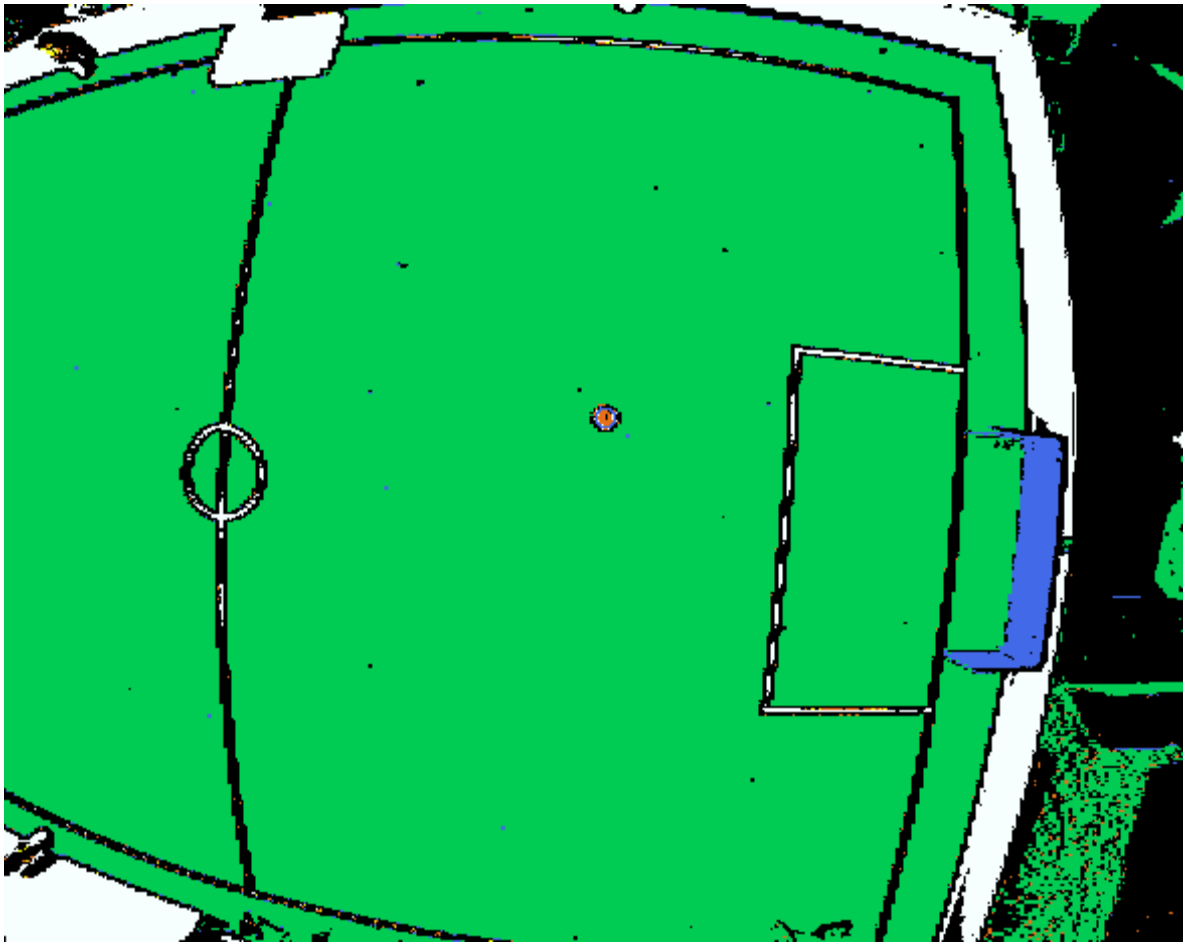


Figure 6.2 - A classified image from the above field camera

6.6. Blob Formation

Blob formation is the process of identifying areas of the same classified colour and storing these areas for later use. The blob formation process involves moving through the image pixel by pixel and comparing each pixel to its neighbours. Pixels of the same classified colour found together are formed into a blob. These blobs are then extended or merged as new pixels are found that belong to it. The algorithm used for the blob formation has two major requirements, accuracy and speed, since it is performed on every pixel.

Each blob has a number of properties attached to it. These can be seen in Table 2. These were taken from those used in the NUbots classification system on the robots. However for this application some of the values used by the NUbots were not required. These were values that tracked the points at which the pixels intersected the edges of the blobs.

Property	**Definition**
Colour	Defines the colour of the pixels that were used to form the blob.
Blob Number	Is the number of the blob given at creation, used to identify blob.
Width	The width of the blob in pixels.
Height	The height of the blob in pixels.
Area	The area of the blob in pixels (Width × Height).
Minx	The smallest X coordinate in the blob (left edge).
Maxx	The largest X coordinate in the blob (right edge).
Miny	The smallest Y coordinate in the blob (top edge).
Maxy	The largest Y coordinate in the blob (bottom edge).
Pixels	The number of pixels present in the blob.
Trueblob	Used to keep track of the current blob in use when blobs have been merged.
Ignore	Used to indicate that the blob should no longer be used.

Table 2 - Blob Properties

6.7. User Interface

The user interface was created to utilise menus rather than buttons. The primary reason for this was that the images have a resolution of 640x480 and therefore take up a large portion of the screen. Once multiple images are added this becomes an even greater problem. Using menus allows more room on the screen to display images rather than buttons or other controls.

The file menu, seen in Figure 6.3, is used to open and store files. The files that can be accessed are lookup tables and image streams. The lookup tables store the table to map the YUV values to their classified values. The image stream files store the raw data stream from the camera, this can be used to later review images.

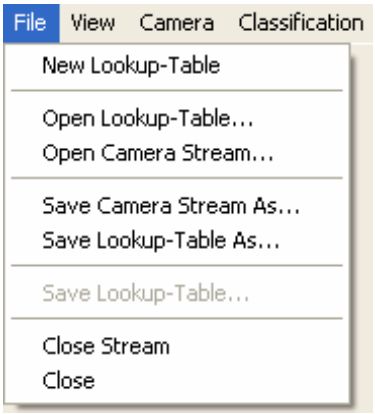


Figure 6.3 - File Menu

The camera menu, seen in Figure 6.4, is used to connect and disconnect the camera driver. The menu also allows access to the drivers configuration dialog for access to the camera settings. The begin streaming and stop streaming options begin and stop the streaming of images from the camera once itr has been connected. To stream image a file must be specified for writing. The camera menu also allows the grabbing of individual images.

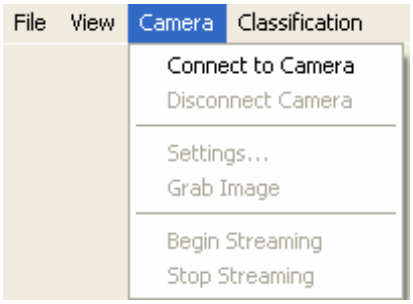


Figure 6.4 - Camera Menu

6.7.1. Classification

The classification menu allows the user to select the classified colour they are currently defining. They can then select pixels from the colour image by left clicking on the required pixel. Pixels that are of the selected colour are highlighted in the currently selected classified colour. To write this association to the lookup table, the user clicks the right mouse button.

To speed up this process the colour sensitivity control seen in Figure 6.6, is used. This control enables the user to extend the range of the colours properties to be selected when the user selects a pixel. A sensitivity of zero indicates that only colours whose values for the lookup table are equal to that clicked are selected. A value of one indicates that the values one below and one above are also selected. A value of two allows values within a range of two from the selected value in that component, and so on. This is true for each of the Y, U and V components.

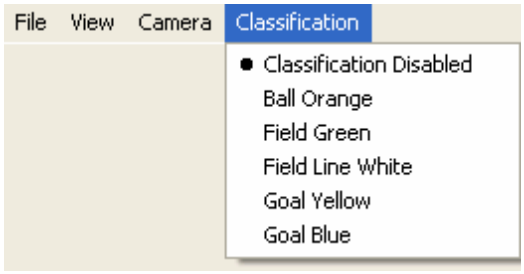


Figure 6.5 - Classification menu

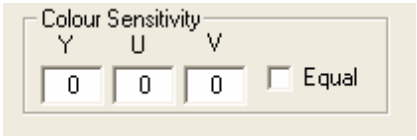


Figure 6.6 - Colour sensitivity control

6.7.2. Images

The program displays one enlarged image at full resolution as well as several smaller images. This interface is used during the manual classification procedure and testing. The smaller images can be enlarged by clicking on it with the left mouse button. This causes the selected image to be displayed in the enlarged image section and the previously enlarged image to be displayed in the place of the smaller image.

There are a number of overlays available for the program to display the current classified pixels and the blobs formed by them. These overlays are drawn over the original colour image.

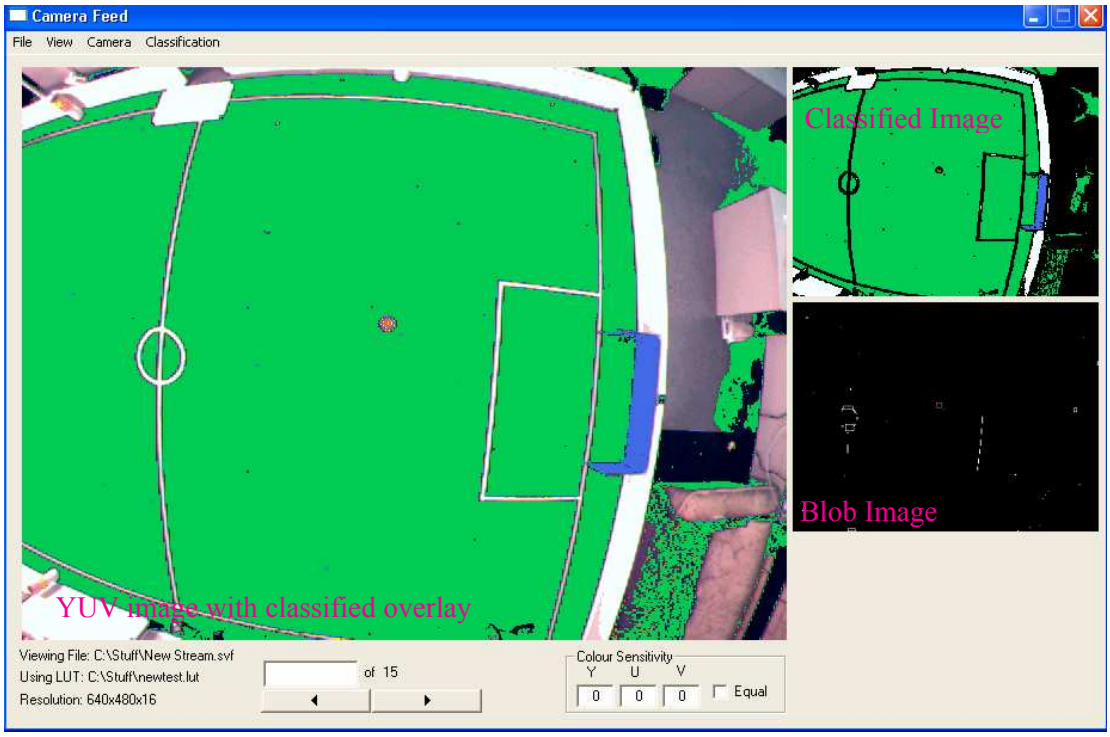


Figure 6.7 - Interface for camera dialog

6.7.3. Zooming

To aid in testing and classification a zoom feature was added. This allows the user to select a predefined level of zoom, or specify their preferred value. Zooming allows more accurate selection of pixels and also greater detail to be seen.

The zooming can also be controlled using the mouse. Moving the mouse wheel up increases the zoom on the image, while moving the mouse wheel down decreases the zoom. When zoomed in, holding down the middle mouse button allows the image to be panned. This is required as the display area of the image does not change and stays at 640 by 480. Double clicking the mouse button resets the zoom to the original level and re-centres the image.

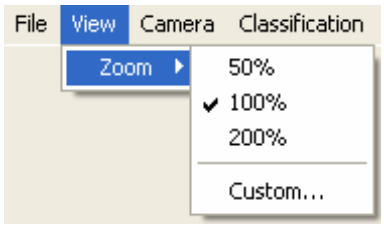


Figure 6.8 - View Menu

6.8. Camera Interfacing

The Basler A601fc camera was supplied with a dll file allowing access to the cameras drivers. The driver provided functions to connect and disconnect from the camera, a function to open the drivers' settings dialog, as well as functions to read and modify settings. To grab images from the camera there are two possible modes, individual frame grabs, or pass through mode. With individual frame grabs the function is passed a buffer and the current image is written to this buffer. This is useful for individual image captures.

Using pass through mode the driver takes images at the frame rate currently set in the drivers settings dialog. Upon calling the function to enable pass-through mode a pointer to an array of buffers is passed. The images are stored into one of these buffers. Each time a new image is available an event message is sent to the window specifying the pointer to the start of the current buffer, allowing the code to process the image to be run. This method was used as it easily allows a fixed frame rate to be used for the streaming.

6.9. Image Display and Conversion

The images obtained from the camera are in YUV 4:2:2 in the UYVY format [16]. This means that each Y value shares its U and V value with the neighbouring Y pixel. It is possible to perform interpolation to calculate an approximation for this missing data. However in an effort to quickly display an image the same U and V data was used for both of the Y values. This produced a clear image and so the interpolation was not implemented as it did not appear necessary.

U_1	Y_1	V_1	Y_2	U_3	Y_3	V_3	Y_4	U_5	Y_5	V_5	Y_6	•••

Figure 6.9 - UYVY Data Structure

To display the image, the YUV4:2:2 images are converted to RGB which is then written to a bitmap and displayed onto the screen. The RGB image is used to draw extra data onto the image before it is displayed. The conversion between YUV and RGB used is shown below.

$$R = clip[(298(\mathrm{Y}-16)+409(\mathrm{V}-128)+128)>>8]$$
$$G = clip[(298(\mathrm{Y}-16)-100(\mathrm{U}-128)-208(\mathrm{V}-128)+128)>>8]$$
$$G = clip[(298(\mathrm{Y}-16)+516(\mathrm{U}-128)+128)>>8]$$

where clip denotes clipping between 0 *and* 256

6.10. Processing an Image

When an image is initially read it is first classified. Blob formation is then run on the classified image. If the image needs to be displayed, the YUV image is then converted to RGB format and written to a bitmap. The classified image, blobs and objects are all drawn to RGB images, or over the coloured image as required and these are copied to bitmaps. These bitmaps are then accessed and displayed as required.

6.11. Object Recognition

Once objects are recognised their position in the image is used to calculate their location on the field. This requires that the camera be calibrated upon installation and each time the camera, or the field is moved.

6.11.1. Ball Recognition

The ball recognition is much simpler to that implemented on the robots. The ball stays approximately the same size as it moves around the field as the distance does not change. This is taken advantage of in that only blobs of the appropriate size are checked as being the ball. The circle fitting code described in section 3.4 was modified to work with the images from the new camera and vision system and is used to more accurately recognise the ball and determine its centre, similar to when used on the robot.

6.11.2. Robot Recognition

Robot recognition requires a special marker attached to the top of each individual robot to uniquely identify them, so that the data collected from robots can be matched to each specific robot seen by the camera.

6.11.3. Image Distortion

The lens used on the camera causes some “fish-eye” distortion of the image; hence the field lines are not straight but curved. To accurately find the position of objects on the field this distortion must be either corrected or taken into account when calculating the position of the object.

Figure 6.10 - An image from the camera showing the 'fish eye' distortion

6.12. Results

The Interface is fully completed with all elements working. The file I/O is fully completed with the ability to save and open both lookup tables and image streams. The classification and blob formation system are both working and images can be streamed, displayed and classified.

7. Future Work

The object recognition software has all been completed and was used successfully at the 2005 robocup competition. The overhead camera is nearing completion with the basic vision system fully implemented.

7.1. Immediate Future

- ⇨ Implement robot recognition for the overhead camera.
- ⇨ Implement fish-eye distortion correction for overhead camera.

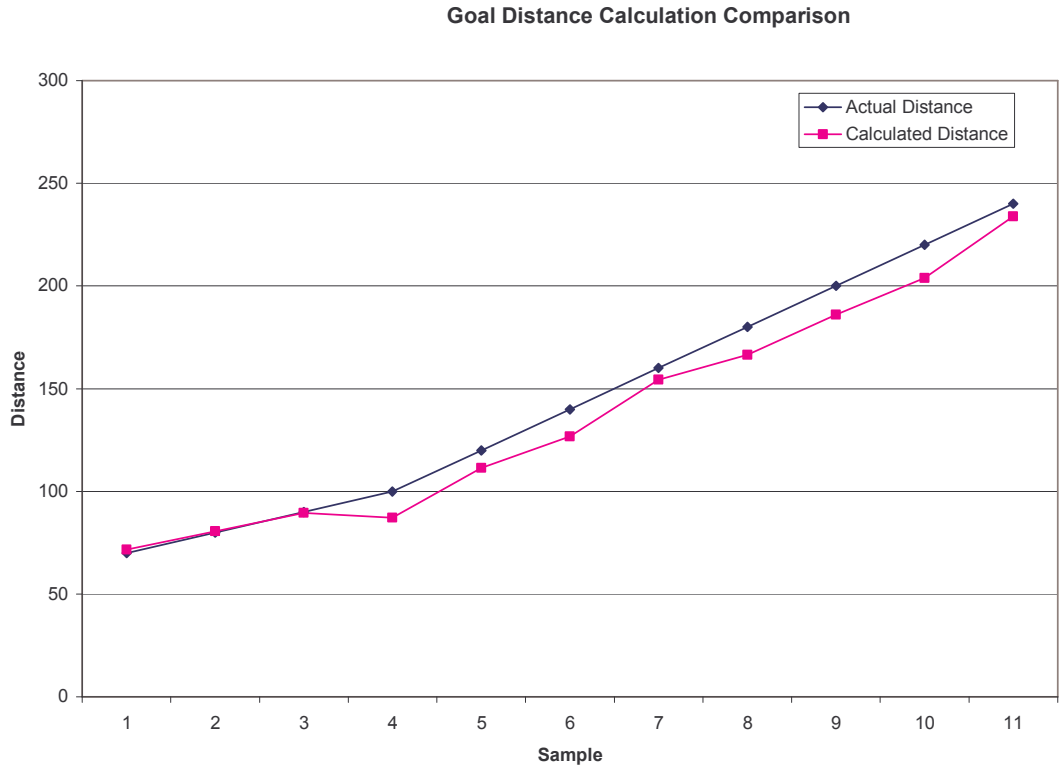
7.2. Possible Extensions

- ⇨ Ellipse fitting to ball to find velocity.
- ⇨ Determine skew for vertical camera movements.

8. References

1. *RoboCup Brief Introduction*. 2005, The RoboCup Federation, http://www.robocup.org/Intro.htm.
2. *Sony Four Legged Robot Football League Rule Book*. 2004, RoboCup Technical Committee, http://www.tzi.de/4legged/pub/Website/History/Rules2005.pdf.
3. *OPEN-R SDK: Model Information for ERS-7*. 2004, Sony Corporation
4. Henderson, N., *Colour Classification in Robotic Vision*. 2005, University of Newcastle
5. Seysener, C., *Vision Processing for RoboCup 2003*. 2003, University of Newcastle
6. Seysener, C.J., C.L. Murch, and R.H. Middleton. *Extensions to Object Recognition in the Four-Legged League*. in *Proceedings of the RoboCup 2004 Symposium*. 2004: Springer-Verlag
7. Hong, K., *NUbots: Enhancements to Vision Processing, and Debugging Software for Robocup Soccer*. 2005, University of Newcastle
8. Röfer, T., et al., *German Team Robocup 2004*. 2004, German Team, http://www.germanteam.org/GT2004.pdf.
9. Rossum, G.v., *Extending and Embedding the Python Interpreter Release 2.4*, J. Fred L. Drake, Editor. 2004, Python Software Foundation, http://www.python.org/doc/2.4.1/download.html.
10. Pilu, M., A. Ftzgibbon, and R.B. Fisher, *Direct Least Square Fitting of Ellipses*. 1996: Edinburgh, http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/PILU1/demo.html.
11. Weisstein, E.W., *"Least Squares Fitting."* in *MathWorld*, Wolfram Web, http://mathworld.wolfram.com/LeastSquaresFitting.html.
12. *Technical Challenges for the RoboCup 2005 Legged League Competition*. 2005, RoboCup Technical Committee, http://www.tzi.de/4legged/pub/Website/Downloads/Challenges2005.pdf.
13. Martinez-Gomez, L.A., et al. *Eagle Knights Small Size League Team Description Paper*. in *Proc. RoboCup 2005*. 2005. Osaka, Japan, http://robotica.itam.mx/espanol/archivos/EKTDP.pdf.
14. Martinez-Gomez, L.A. and A. Weitzenfeld. *Real Time Vision System for a Small Size League Team*. in *Proc. 1st IEEE-RAS Latin American Robotics Symposium*. 2004. Mexico City, http://robotica.itam.mx/espanol/archivos/Vision.pdf.
15. Simon, M., S. Behnke, and R. Rojas, *Robust Real Time Color Tracking*. Lecture Notes in Computer Science, 2001. **2019**: p. 239-248,http://robocup.mi.fu-berlin.de/docs/MelWS/MelWS.html.
16. Basler, *Basler A600f Users Manual*. 2005, http://www.baslerweb.com/popups/874/A600f_Users_Manual.pdf.

Appendix A – Goal Distance Comparison



Appendix B – Pan Sensor Values