The University of New South Wales

School of Computer Science and Engineering

2004 rUNSWift Thesis – Low Level Vision

Jing Xu ID: 3020495

Bachelor of Computer Science (Honours)

Submission Date: September 2004

Supervisor: William Uther

Assessor: Claude Sammut

Contents

1 Introduction

The Robocup four-legged league is an annual competition entered by the UNSW rUNSWift team, where Sony AIBO robots are used to play four on four soccer. The robots provide a standard platform, where only the software differ between each team.

One of the robot's main sensors is the camera in its nose, images are captured at 30 frames per second, from which, most of the information regarding its environment are derived and acted upon. Due to the low position of the vision module on the data flow hierarchy, its accuracy and speed is crucial to the functionality of all other modules, and the performance of the robot in general.

The rUNSWift vision module is divided into three parts, low-level vision – colour classification and segmentation, mid-level vision – clustering, and high-level vision – object recognition. This paper deals primarily with low-level vision.

Section 2 explores an interesting problem introduced by the ERS-7 model are chromatic distortions around the edge of images. Section 3 explores the current method of colour classification and segmentation in the rUNSWift vision module. Section 4 explores the effect of histogram equalisation on non-linear shifts in colourspace caused by lighting changes. Finally Section 5 explorers extracting additional visual information through edge detection.

2 Ring Correction

2.1 Introduction

A significant problem introduced by the Sony AIBO ERS-7 robots are chromatic distortions near the edge of images. The distortions are roughly cyclic in shape and usually add a blue tint to the affected area. The severity of the distortions is dependent on the colour and position of the affected area. Since the rUNSWift vision system is largely based on colour, these distortions lead to the misclassification of object colour, which results in an increase of noise, false identification (false positives) and mis-identification (true negatives) of objects.

Since this is the first time the ERS-7 model is adopted, although many other teams have also noticed this problem [5], there has been no research conducted in previous years, and no known solutions proposed. Although the ERS-210 also has some chromatic distortions, it has not been severe enough to cause concern. A comparison between the chromatic distortions of the ERS-7 and ERS-210 can be found on Roefer's website [6].

2.2 Concept

Fig 2.1 YUV plane of white border

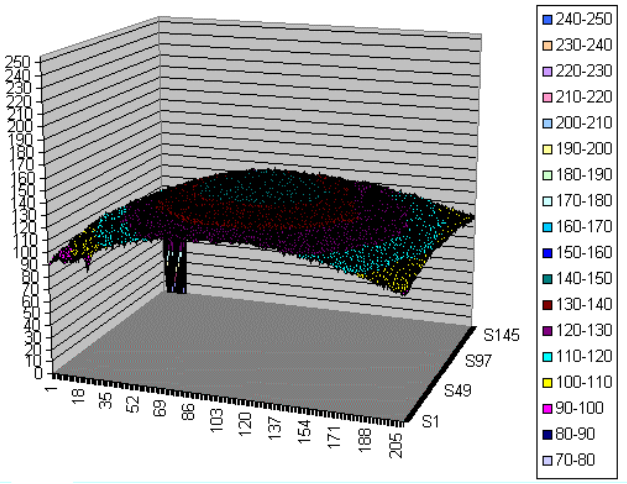


Fig 2.2 Y plane of white border

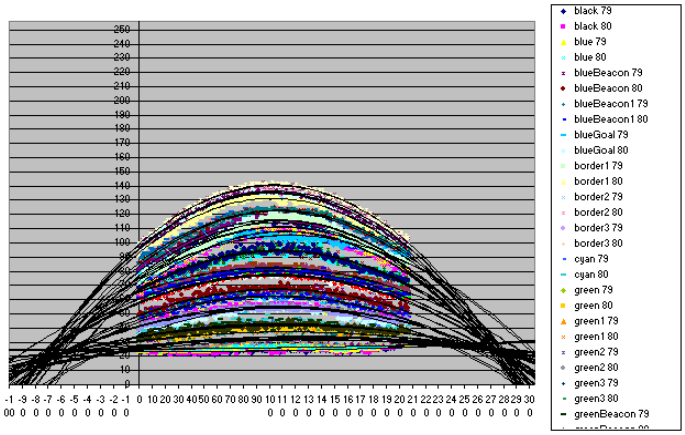


Fig 2.3 Y plane of uniform colours at Y=79 & Y=80 during daytime - unprocessed

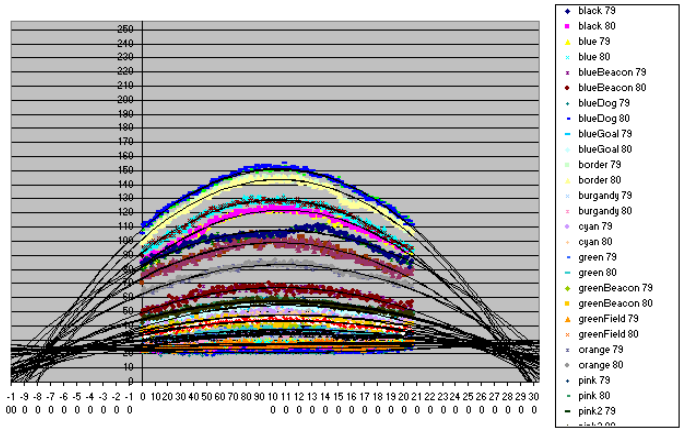


Fig 2.4 Y plane of uniform colours at Y=79 & Y=80 at night time - unprocessed

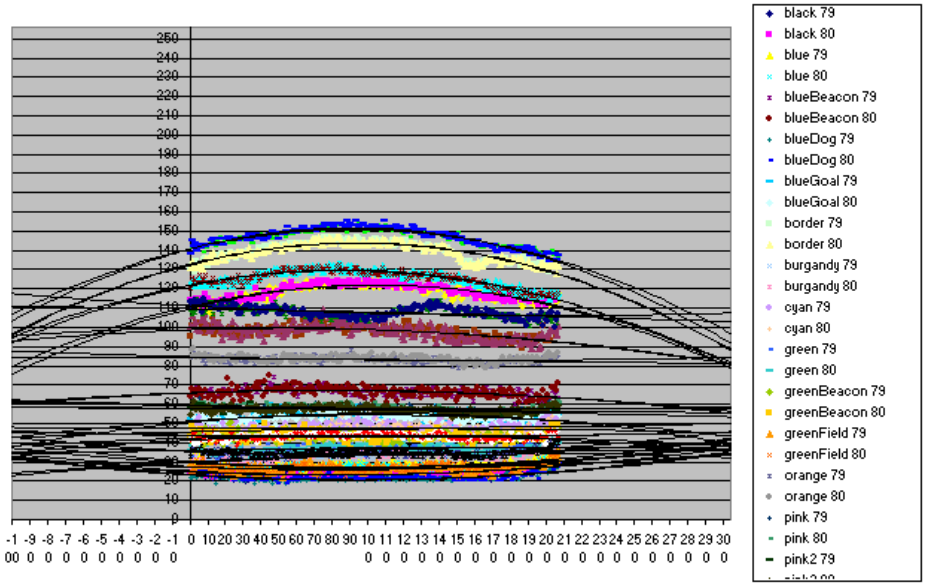


Fig 2.5 Using correction calibrated for daytime images on night time images

2.2.1 Initial Investigation into the Nature of the Problem

Due to the chromatic nature of the distortions, images of uniform colour were taken under controlled lighting conditions to investigate the nature of this phenomenon. YUV values of this set of data were analysed separately and it can be observed that the distortions are:

(a) symmetric about its centre, which is roughly, although not exactly, the centre of the image (*Fig 2.1*)

(b) parabolic in terms of its relationship between chromaticity (z) and pixel position (x, y) (*Fig 2.2*)

(c) different for each distinct colour (*Fig 2.3*)

(d) consistent for different colours with the same camera settings (*Fig 2.3, Fig 2.4*)

(e) independent of lighting conditions (*Fig 2.5*)

Due to the nature of uniform colours, without the distortions, the same colour should have the same YUV intensity (z), despite its pixel position on the image (x, y). But the images of uniform colours captured by the ERS-7 camera show consistently parabolic planes (*Fig 2.3*) rather than perfectly straight horizontal planes. It seems that a transformation which can straighten these planes would be the solution to this problem.

2.2.2 Geometric Distortion

Geometric distortions are a common problem in cameras because of the parabolic nature of lenses. Section 5.11 of Gonzalez [3] offers a good explanation of the theory of geometric distortions and transformations.

Although geometric distortions are usually two dimensional, given the roughly symmetric nature of the "blue ring", a cross section through the centre would be a fair representation in 2D which can be reconstructed into 3D when necessary. If a vertical slice (x, z plane) at y=79 or y=80 is taken, the chromatic distortions can be modelled by geometric distortions

2.2.3 Geometric Transformation

The conventional procedure to correct geometric distortions is geometric transformations in the form of tie-points and grey interpolation, which modify the spatial relationships between pixels in an image [3].

Unfortunately, the spatial transformations described in [3] require sample points with known solutions from the entire image. In the case of conventional geometric distortions this is not a problem, but in this case it is difficult to acquire samples of every colour in the visible spectrum and next to impossible to predict their chromaticity under lighting conditions which are not sufficiently static. Therefore the transformed solutions to tie points of certain chromaticity cannot be known, hence the general equations of geometric transformations for those colours cannot be calculated, rendering this method unusable.

2.2.4 Modelling

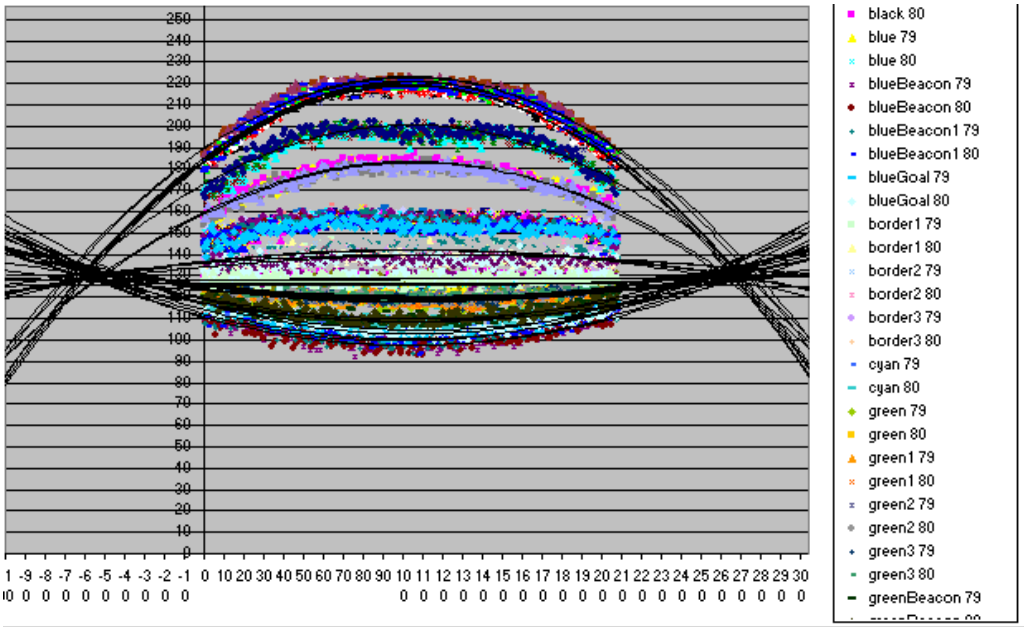


Fig 2.5 U plane of uniform colours at Y=79 & Y=80

It seems that if the distortions can be modelled, then it can be solved. After the application of regression analysis to vertical slices (in *xz* plane) of uniform colours, assuming the distortions are quadratic, and forecast ±100 units on the x-axis, it can be seen that the regression lines converge at roughly the same two points (p_0, p_1), one on either side of the image. These two points remain constant for the same camera settings independent of lighting.

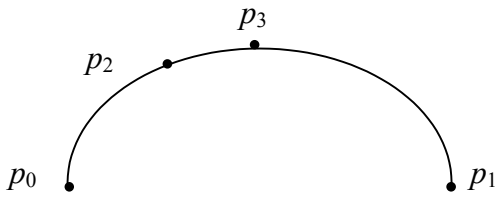


Fig 2.6 A parabola which begins at p_0, ends at p_1, passing through points p_2 and p_3

To calculate the equation of the distortion for any point p_2, so that given its position on the image (x,y) and chromaticity (z), then the equivalent z for an area with no distortion, ie. the centre of the image p_3, can be predicted (*Fig 2.6*).The problem then becomes, given 3 points, p_0=(x_0,z_0), p_1=(x_1,z_1), p_2=(x_2,z_2) and x_3, the x value of a fourth point p_3 on a parabola, find the z value (z_3) of p_3. Assuming the parabola is quadratic, the equation of the parabola would be:

$$z = \left[\frac{z_0}{{x_0}^2} - \frac{1}{x_0} \left(\left(\frac{z_1 - \frac{{x_1}^2 z_0}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right) - \left(\frac{1 - \frac{{x_1}^2}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right) \left(\frac{\left(z_2 - \frac{{x_2}^2 z_0}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{z_1 - \frac{{x_1}^2 z_0}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)}{\left(1 - \frac{{x_2}^2}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{1 - \frac{{x_1}^2}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)} \right) \right) \right]$$

$$- \frac{1}{{x_0}^2} \left[\frac{\left(z_2 - \frac{{x_2}^2 z_0}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{z_1 - \frac{{x_1}^2 z_0}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)}{\left(1 - \frac{{x_2}^2}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{1 - \frac{{x_1}^2}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)} \right] x^2$$

$$+ \left[\left(\frac{z_1 - \frac{{x_1}^2 z_0}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right) - \left(\frac{1 - \frac{{x_1}^2}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right) \left(\frac{\left(z_2 - \frac{{x_2}^2 z_0}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{z_1 - \frac{{x_1}^2 z_0}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)}{\left(1 - \frac{{x_2}^2}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{1 - \frac{{x_1}^2}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)} \right) \right] x$$

$$+ \frac{\left(z_2 - \frac{{x_2}^2 z_0}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{z_1 - \frac{{x_1}^2 z_0}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)}{\left(1 - \frac{{x_2}^2}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{1 - \frac{{x_1}^2}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)}$$

Equation 1

(Proof see Appendix A)

The complexity of the equation is high, which could easily introduce implementation errors, and calculating the solution for every pixel many be too computational expensive.

2.2.5 Electric Fields

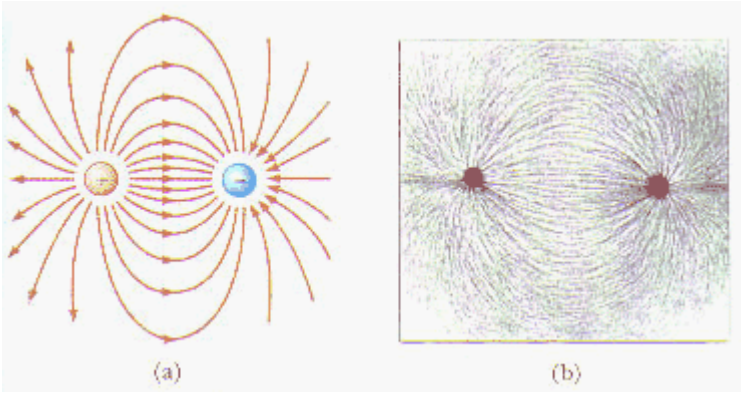


Fig 2.7 Electric Field Lines (a) Diagram (b) Iron fillings aligned by magnet

Another way to model the distortions is through the concept of electric fields. Between two oppositely charged point objects, the lines of force are concentrated at each point charge. A positive test charge would follow a curved path to the negative point charge, these lines of force are much like the chromatic distortions experienced by the images (compare *Fig 2.5* with *Fig 2.7*).

Glenbrook [7] is good reference on the concept of electric fields, point charges and lines of force, and also the quantisation of electric fields and electric potential. Where The Electric Field Java Program[8] is an applet which simulates the electric field generated between point charges.

The concept of electric fields is not only used to explain electromagnetic forces, but is also used to model the shape of cells during mitosis (cell division) [9] and electrophoresis, a sedimentation technique used in DNA analysis to determine the molecular weight of proteins [10].

Assuming the two points on either end of the image, p_0 and p_1, are equal and opposite point charges, then the distortions can be modelled by the electric field hence generated, and the equation for each colour would be equivalent to the equation of the corresponding electric field line.

$$E = \frac{kq_0}{r_0^2}\hat{r}_0 + \frac{kq_1}{r_1^2}\hat{r}_1$$

Equation 2

E	=	Electric force	
k	=	Constant of proportionality	$= \frac{1}{4\pi\varepsilon_0}$
ε_0	=	Absolute permittivity of free space	$= 8.85 \times 10^{-12}$
q_0	=	Charge 1	
q_1	=	Charge 2	
r_i	=	Distance between test charge and q_i	
$\hat{r}_i$	=	Unit vector of r_i	

However, given the magnitude and polarity of the point charges p_0 and p_1 (assume +1 and –1), and a third small positive test charge p_2, Equation 1 only derives the immediate direction of p_2, and not the equation of its path. But this is still useful because we can use numerical integration to find each point on the path.

Such methods may be computationally expensive to calculate dynamically, but given the nature of the problem, where the distortions are static, and the interval is discrete (integral) and finite (between 0 and 255), the corrections can be calculated offline in the form of a look-up table. The electric field model was chosen for its simplicity.

2.3 Implementation

A look-up table of acceptable size is usually preferred to dynamic calculations because of speed.

2.3.1 Method 1 – Naïve Implementation

Initially the concept is complex; there are 5 dimensions, x, y, Y, U, V, and 3 values to be looked up, Y_z, U_z, V_z. There are 208 possible values for x, 160 for y, and 256 for each of Y, U and V.

The look-up table would contain:

$$x \times y \times Y \times U \times V = 208 \times 160 \times 256 \times 256 \times 256 = 5.52 \times 10^{11}\ cells$$

If each of Yz, Uz and Vz require one byte of storage, then the look-up table would require:

$$5.52 \times 10^{11} \times 3 = 1.66 \times 10^{12}\ Bytes = 1560GB \qquad \textit{Look-up Table (LUT) 1}$$

One memory stick can only contain a maximum of 16 MB, therefore Method 1 is not feasible.

2.3.2 Method 2 – Assume YUV independence

Knowing that YUV are independent of each other, LUT 1 can be broken down into 3 smaller tables, one for each of Y, U and V.

$$x \times y \times (Y + U + V) = 208 \times 160 \times 256 \times 3 = 24.38MB \qquad \textit{LUT 2 (3 tables)}$$

LUT2 still does not fit onto one memory stick, therefore Method 2 is not feasible.

2.3.3 Method 3 – Assume x,y Symmetry

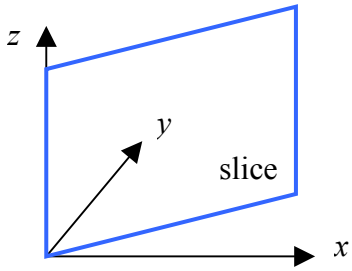


Fig 2.8 Vertical slice in the xz plane

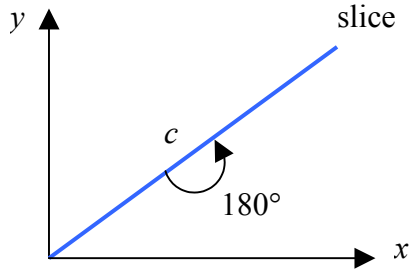


Fig 2.9 Vertical slice in the xy plane

If we exploit the symmetry of the problem, then we can take one slice in the *xz* plane along a diagonal of the rectangular image in the *xy* plane through the centre of distortion *c* (*Fig 2.8*), and rotate between 0-180 degrees to recreate y (*Fig 2.9*).

This would require:

$$\text{length of a diagonal*(Y+U+V)} = \sqrt{x^2 + y^2} \times 3 = 263 \times 256 \times 3 = 197.25KB$$

LUT 3 (3 tables)

Or cut the above slice in half along the y axis and rotate 0-360 degrees to recreate y.

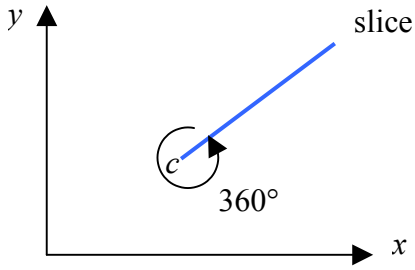


Fig 2.9 Vertical slice in the xy plane

This would require:

length of half a diagonal*(Y+U+V)

$$= \frac{1}{2}\sqrt{x^2 + y^2} \times 3 = \frac{1}{2} \times 263 \times 256 \times 3 = 98.62KB$$ *LUT 4 (3 tables)*

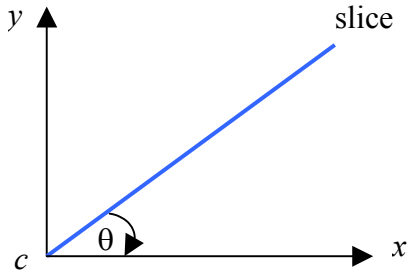


Fig 2.10 Map slice in the xy plane to its equivalent on along the x axis

To map *x*,*y* values to *x* values, LUT 4 requires the uses of trigonometry functions for each pixel, which reduces CPlanes processed per second down to approx. 11-12. Hence although this table is small enough, the overhead produced by the extra trigonometry transformations are too great to deem this method feasible.

2.3.4 Method 4 – LUT 4 + Replacement for Trigonometry Calculations

One way to compensate is to implement another table which maps x,y values to a corresponding x value in LUT 4.

This requires an extra:

x*y*(number of bytes per int) $= 160 \times 108 \times 4 = 130KB$ *LUT 5*

LUT 4 + LUT 5 require 98.63+130=228.63 *KB*

Method 4 is both fast enough and small enough to be feasible.

2.3.5 Generate Tables

LUT4 can be generated by the following code:

```
protected static void drawLine(int slice[][], Point q1,
Point q2, double k)
    {
     double rsq1, rsq2, x, y, sumX, sumY, dx1, dy1, dx2,
dy2, d;
     int CX = avg(q1.x,q2.x);

     for(int i=0; i<slice.length; i++) {
        x=CX; y=i;
        sumX=0; sumY=0;

        do {
            slice[(int) Math.round(y)][(int) Math.round(x)-
CX] = i;
            dx1 = q1.x - x; dy1 = q1.y - y;
            dx2 = q2.x - x; dy2 = q2.y - y;

            rsq1 = dx1*dx1 + dy1*dy1;
            rsq2 = dx2*dx2 + dy2*dy2;

            sumX += dx2/rsq2 - dx1/rsq1;
            sumY += dy2/rsq2 - dy1/rsq1;

            d = Math.sqrt(sumX*sumX + sumY*sumY);
            x += sumX/d;
            y += k*sumY/d;
        }  while(y>=0  &&  Math.round(y)<slice.length  &&
Math.round(x)-CX<slice[i].length);
    }
}
```

2.3.6 Table Access

The tables can be accessed via:

```
Yz = LUT4Y[Y][(LUT5[y][abs(x-cyx)])];
Uz = LUT4U[U][(LUT5[y][abs(x-cux)])];
Vz = LUT4V[V][(LUT5[y][abs(x-cvx)])];
```

Where `cyx`, `cux`, `cvx` are the *x* values of the centres of distortion in Y, U, V respectively.

2.4 Results

2.4.1 Effectiveness

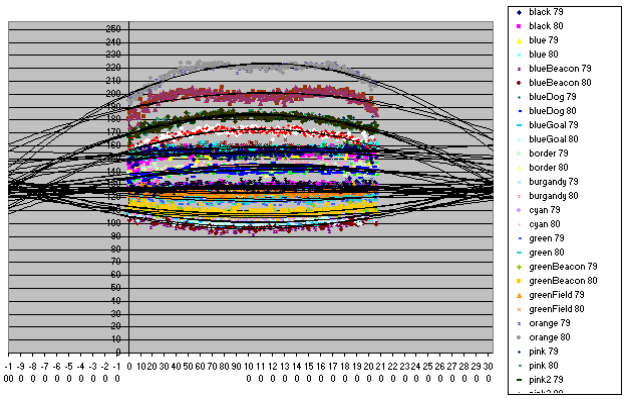


Fig 2.8 UnprocessedU plane of uniform colours at Y=79 & Y=80

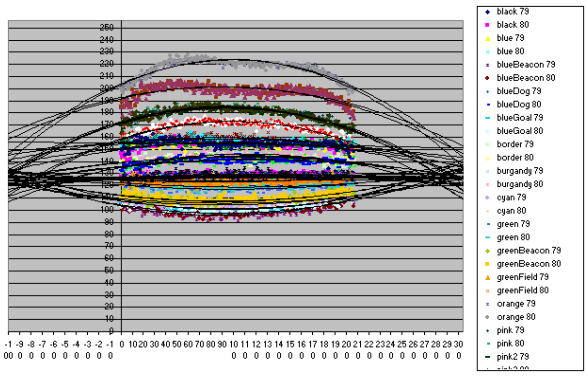


Fig 2.9 Preliminarily Processed U plane of uniform colours at Y=79 & Y=80

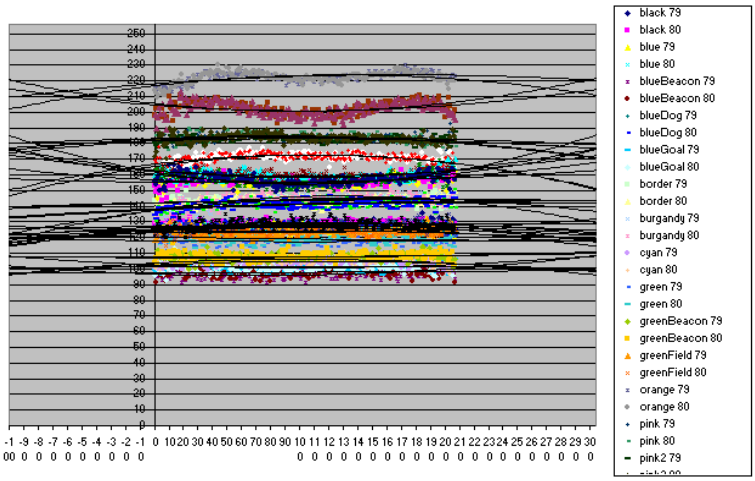


Fig 2.10 Processed U plane of uniform colours at Y=79 & Y=80 with concavity coefficient

Initial testing verifies that this method reduces distortions (*Fig 2.9*), but these lines are not as straight as predicted. There are still consistent curvatures to be seen. To compensate for this, the concavity of the curves was increased by a constant, which was determined by trial and error. After this modification, there are no consistent curves to be observed (*Fig 2.10*). For a more comprehensive comparison, see Appendix B.

However, there are certain abnormalities in the lines furthest away from the centre. It can be seen that these areas do not conform to the same concavity as the rest of the curves in the original images. These parts of the curve cannot be modelled by the general equation.

There are also some level of noise, which is expected.

2.4.2 Speed

Although ring correction introduces some overhead in speed, it outweighs the amount of time the blobber spends processing the extra noise caused by the ring.

2.4.3 Robustness

A ring correction file generated under one lighting condition is equally effective on images captured under another lighting with the same camera setting (*Fig 2.5*). Although, one calibration does not work for different camera settings, but after recalibration using the same method, similar results were achieved.

2.5 Discussion/Conclusion

Ring correction using the electric fields model has generally demonstrated to be a good method in terms of speed and effectiveness. It can be seen that there is still noise, and certain colours which are further away from the centre of distortion have segments which are not uniform. These cannot be avoided since they are present in the original data. If these inconsistencies in concavity can be modelled, then they can be corrected.

Currently the process is rather time consuming, a more methodical approach to the approximation of end points and automation of the entire process is desirable.

3 Colour Classification

3.1 Introduction

The aim of the rUNSWift vision system is to extract and process as much visual information as accurately as possible in real-time. Although inter-related, the priorities can be roughly categorised into speed, accuracy and completeness of information, in order of importance.

To aid in the development of such a system in its early stages, the Robocup field is set up in such a way, where lighting conditions are controlled, and every object class of interest (*Table 2.1*) can be characterised by a distinct colour.

Value	Colour	Objects
0	Orange	Ball
1	Blue	Blue beacon, blue goal
2	Green	Green Beacon
3	Yellow	Yellow beacon, yellow goal
4	Pink	Pink beacon
5	Robot Blue	Blue robot
6	Robot Red	Red robot
8	Field Green	Green field
7	Grey	Grey robot
9	White	Borders, field lines, white robot, light background
10	Black	Referee pants, black robot, dark background

Table 3.1 Objects of interest

To take advantage of this situation, a fast method of colour image segmentation based on a YUV colour look-up table was developed[11] and improved upon over the years [14][15][17][1][2]. The 2002 thesis[1] contains a detailed and structured description of the colour classification process, but not much of the concept was discussed, and many details were amended in 2003 and 2004. The 2003 thesis[2] only emphasises the changes made in that particular year (eg. learning algorithm and list of interesting

objects), whereas the overall description of the process itself is brief. Small changes made in 2004 remain undocumented. This section aims to document a coherent synthesis of concepts and 2002-2004 rUNSWift practices in colour classification adopted by the 2004 team.

3.2 Concept

In an ideal environment, where each object of interest belongs to a unique class in colour space, each YUV coordinate is associated with only one symbolic colour. Such a mapping is linear, and can be looked up in a static table which only needs to be generated once.

It is near impossible to find samples of every possible colour in the environment, and since there are 256*256*256 ~= 17 million combinations of YUV coordinates, it is cumbersome to manually associate each coordinate with a colour. Hence it is much more efficient in terms of time and effort to generate tables based on a smaller but representative set of training data.

Since adjacent YUV coordinates usually represent the same colour, it is feasible to generalise and compress groups of YUV coordinates into one class of symbolic colour without losing too much information.

There have been two approaches to segment colour space, (a) divide the native camera colour space (YUV)[11] or transformed colourspace (eg. HSV)[12] into rectangular boxes whether it be done in hardware[13] or software[11]; (b) use a machine learning algorithm to generate classes of arbitrary shape in colourspace[1]. The 1999[14] and 2000[15] rUNSWift teams used a combination of both, rectangular slices in Y plane, and learnt polygons of best fit generated by a Polygon Growing Algorithm were used in the UV plane.

Usually colour classes do not fit into rectangular boxes, so although method (a) is faster offline and dynamically than (b) especially when done in hardware, (b) is preferred because it models the concave nature of actual colour classes more closely, and therefore is more accurate. The extra overhead in speed of (b) both in offline generation and dynamic application is also acceptable.

Two machine learning algorithms were used by rUNSWift in 2004, C4.5 and LMT. C4.5[16] has demonstrated in 2001[17] and 2003[2] to be more robust under lab conditions than algorithms experimented with in previous years, such as polygon growing and nearest neighbour, it is also very fast to run (usually 1-2 min). LMT models the concavity of colour classes better than C4.5, it also allows a confidence factor to be associated with each labelled colour class, but is computationally more expensive (usually 20 min – 1 hour) and its effectiveness requires more rigorous testing. Both algorithms are heavily dependent on the representativeness of the training data.

3.3 Implementation

To generate a look-up table, the 2004 rUNSWift team has 7 steps:

3.3.1 Colour Test

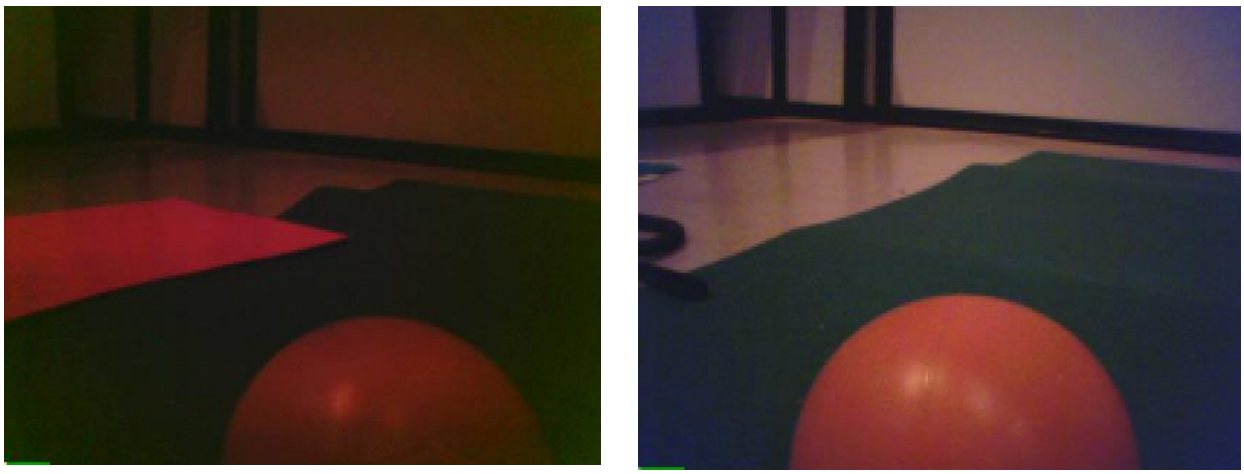


Fig 3.1 Colour Test – images of similar objects taken with different camera settings

(a) White balance: indoor, gain: mid, shutterspeed: high (b)White balace: outdoor, gain: mid, shutterspeed: slow

Several images of reference colour (eg. same segment of white border and beacon under same lighting, *Fig* 3.1) are taken with different camera settings, ie. different combinations of shutter speed, white balance and gain. The images are compared, and the camera setting which yields the clearest distinction between colours is chosen.

Note: usually the shutter speed is set to fast to reduce motion blur, so there are only two settings to be changed: white balance and gain.

3.3.2 Collecting Training Data

Images of every interesting object are taken from different distances at different angles on different parts of the field to product as many possible colours of the same object as possible. This may potentially result in over 100 pictures which would be time consuming to label. A set of pictures has been provided as a reference (*Table 3.2*), there is no need to take every single picture in the table, a representative sample would suffice.

Num	Setup	Purpose
0	One corner beacon + white border	Colour test
1-4	Close-up of each beacon	Large samples of beacon colours
5-8	Mid-range beacons, 1/3 of the field away from beacons	Sample different reflectance of beacons
9-12	Close-up of shadowed beacons	Simulate effect of crowd leaning over beacons, variance in colour
13-14	Close-up ball on field white in the middle and a corner of the field	Large samples of ball, border and field in different parts of the field
15-16	Close-up and mid-range of ball in yellow goal	Large samples of yellow goal and ball
17-18	Close-up and mid-range of ball in blue goal	Large samples of blue goal and ball
19	15 with shadows	Samples of shadowed yellow goal and ball
20	17 with shadows	Samples of shadowed blue goal and ball
21-22	Front and side of red robot in yellow goal	Samples of red goalie
23-24	Close-up and mid-range of red robot in quadrant 1 of field from the front	Large samples of red robot in different parts of the field
25-26	Close-up and mid-range of red robot in quadrant 1 of field from the side	
27-28	Close-up and mid-range of red robot in quadrant 4 of field from the front	
29-30	Close-up and mid-range of red robot in quadrant 4 of field from the side	
21-22	Front and side of blue robot in blue goal	Samples of blue goalie
23-24	Close-up and mid-range of blue robot in quadrant 2 of field from the front	Large samples of blue robot in different parts of the field

25-26	Close-up and mid-range of blue robot in quadrant 2 of field from the side	
27-28	Close-up and mid-range of blue robot in quadrant 3 of field from the front	
29-30	Close-up and mid-range of blue robot in quadrant 3 of field from the side	
67-69	Scrums	Samples of shadowed robots, balls and field
70+	Freestyle – beacons and robots at different angles	Large samples of beacons and robots in different parts of the image (especially ring area)

Table 3.2 List of images to take as training data

3.3.3 Labelling Training Data

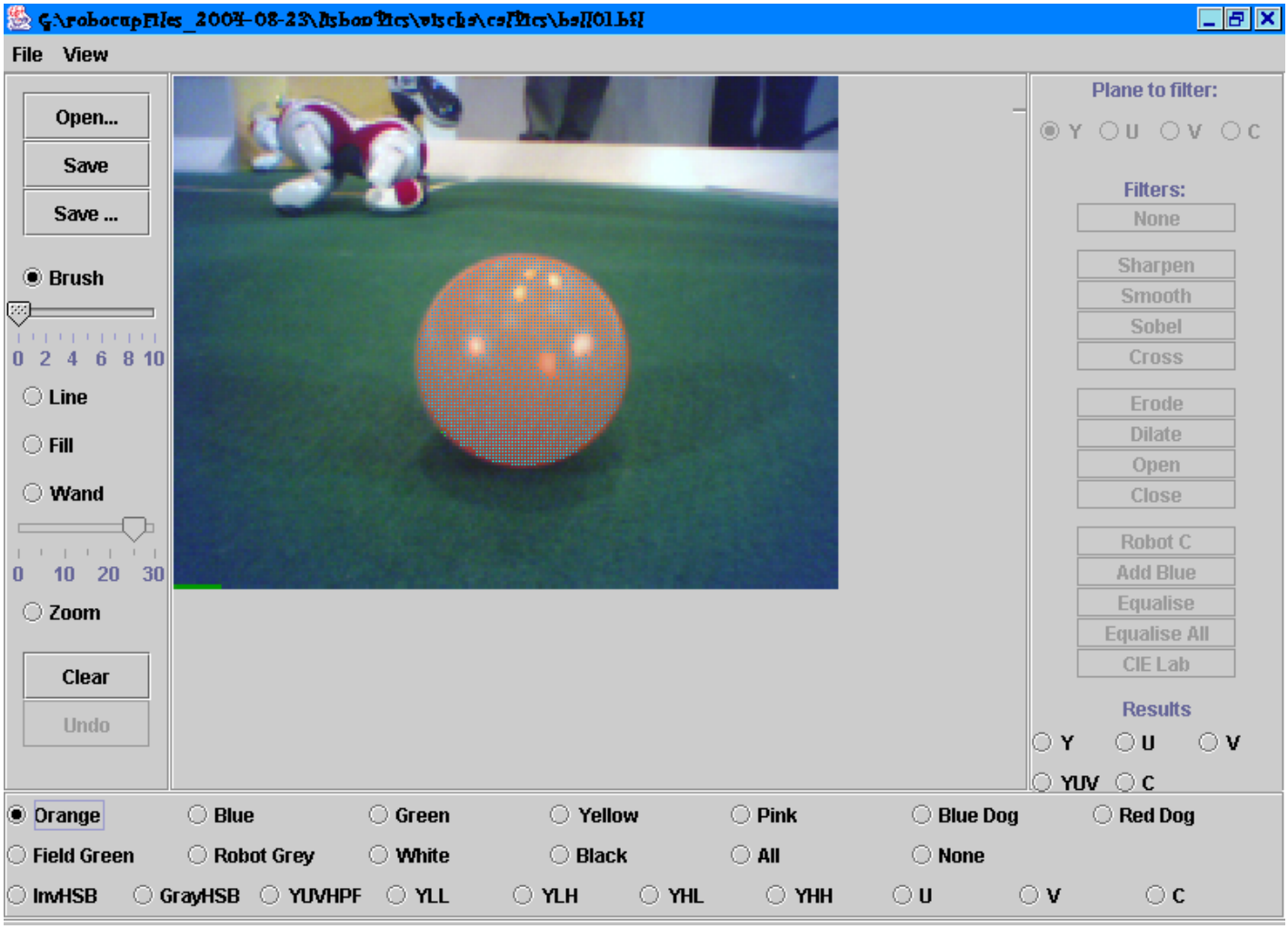


Fig 3.2 Labelling images using ic

Images of interesting objects are labelled with symbolic colours using a graphics editor, *ic*. Sections of images can be "painted" in a symbolic colour using tools such as paint, fill and wand (*Fig 3.2*). How much of which images to label is an art that can only be mastered through experience.

3.3.4 Learning Algorithm

The labelled images are then fed into a learning algorithm, where a decision tree is generated and a look-up table derived.

3.3.5 Test 1

Usually the look-up table yields reasonable colour when tested on the training data, so it is important to test the table on novel images. To test a large quantity of images in as little time as possible, a robot is loaded with the new colour table, and pointed towards objects of interest, while the dynamic CPlanes are noted.

Only in the case where there are significant errors (eg. large areas of red robot is consistently labelled orange) are additional images taken and relabelled, and the learning algorithm re-run (repeat steps 2-4). This is because usually steps 2-4 are the most time consuming.

3.3.6 Manual Classification

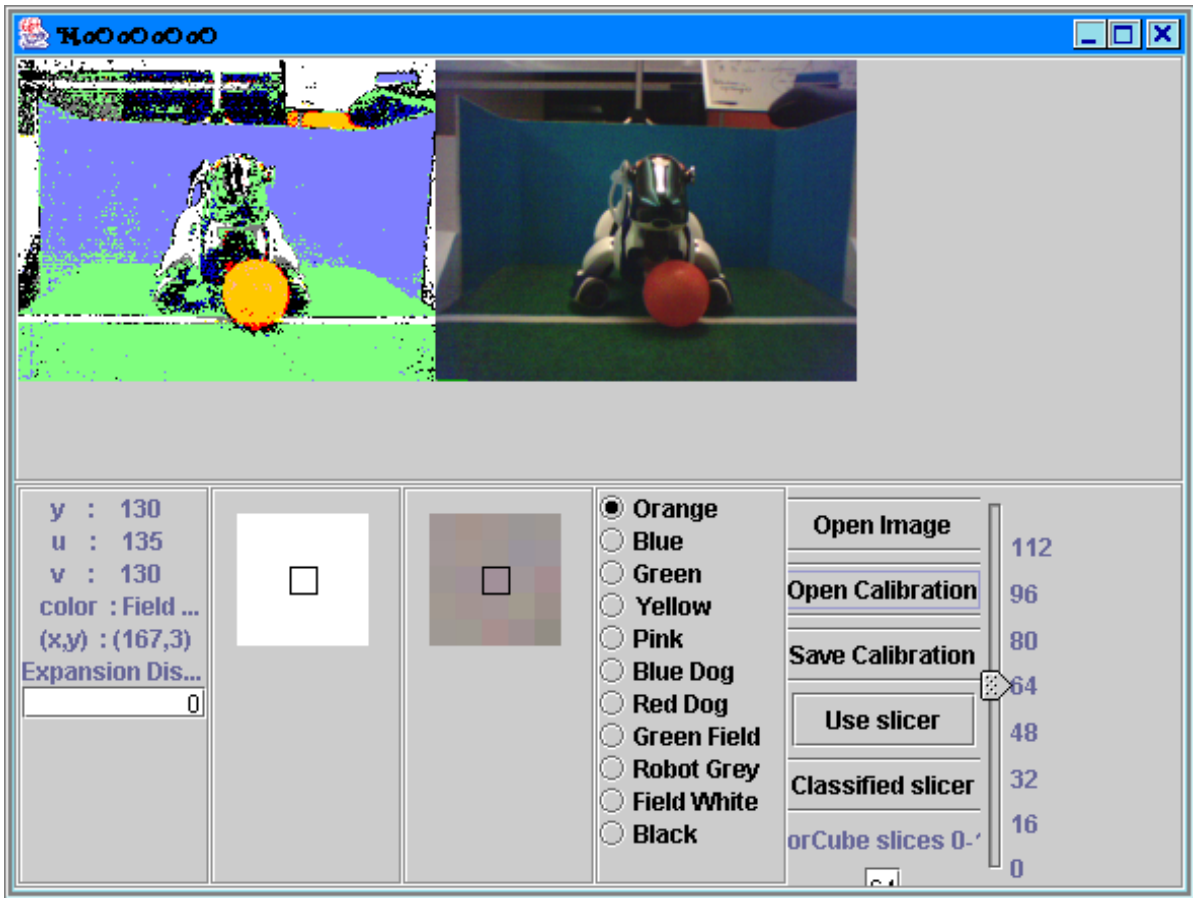


Fig 3.3 Overriding the calibration file using mc

Small errors such as a few scattered pixels consistently being labelled the wrong colour (eg. yellow in white border) can be corrected using *mc*. It lets the user edit values in the look-up table directly (*Fig 3.3*), so it should be used only on a small scale and with caution.

3.3.7 Test 2

The new colour table is tested in the same way as Test 1. Take additional images and redo step 6 until the colour table is acceptable.

3.4 Discussion/Conclusion

The YUV look-up tables are an efficient way of segmenting images into objects, and is currently the only way to process all image frames in real-time. Although it does not allow distinct YUV values to be mapped to more than one colour, which is often the case in nature, it has demonstrated to be reasonably accurate in the idealised Robocup environment until 2004.

When compared to the ERS-210 model, the new ERS-7 model has a camera of significantly poorer quality, where the images are noisier and the colours harder to distinguish. Methods to reduce noise are explored in Section 4.

Local changes in lighting such as shadows and reflections have always been a problem on the Robocup field, where there is a non-linear shift of colour classes in colourspace. This results in more than one value for each YUV coordinate, which is unsupported by the colour table. With the ERS-7 model, the mappings of symbolic colours are further degraded such that the level of accuracy is no longer sufficient for progress in higher leveled modules. It also defeats Robocup's purpose of the eventual transition to a natural environment with dynamic lighting.

There are two approaches to solving this problem, (a) correct the non-linear shift in colour space, explored in Section 4, or (b) extract additional visual information such as shape, explored in Section 5.

In conclusion, although the current method of colour classification is fast and reasonably accurate for an ideal environment, the perceived environment is less than ideal than previous years, hence colour is no longer reliable enough to be the only source of information.

4 Histogram Equalisation

4.1 Introduction

The rUNSWift system is highly sensitive to changes in lighting, due to its dependency on colour. Even though lighting conditions in the Robocup environment is currently controlled, local changes in lighting such as shadows and reflections are unavoidable. This introduces such inaccuracies in visual information that it is difficult for higher leveled modules to function correctly.

The colour of surfaces is the result of a combination of components, such as the spectral radiance of the illuminant, specular reflection, and the spectral reflectance of the surface. Forsyth[4] Section 6.1 is a good reference on the physics of colour.

Changes in illumination cause a non-linear shift of colour classes in the otherwise linear colourspaces. This creates more than one mappings of colour classes to colourspace coordinates during certain intervals of time, which is unsupported by the static colour look-up tables used the rUNSWift vision module.

A solution proposed by Cameron and Barnes[20], is to dynamically update the colour table based on prior knowledge of the environment, such as the shape of objects. This method is computationally feasible, and yields good results for object of known shapes such as beacons and goals, but it is ineffective towards objects of unknown or inconstant shape such as robots and partial balls. Since the landmarks are in positions which are rarely affected by local changes in lighting, and the robots and balls are affected most, this method is inappropriate for solving this problem.

This section presents Histogram Equalisation, a technique which compensates for changes in lighting in an image, without prior knowledge of visual information other than chromaticity, which is also computationally feasible to implement.

4.2 Concept

Histogram equalisation seeks to automatically determine a transformation which produces an image with a uniform histogram of intensity values. Basically, histogram equalisation consists of three steps:

(a) collect an array of cumulative histograms ΣH_i

(b) equalise histogram by multiplying ΣH_i by the maximum intensity level/number of pixels

(c) map original intensity level to equalised intensity level and reapply transformations to image

Section 3.3.1 of Gonzalez[3] and the ImageJ Documentation[19] are good references on the theory of histogram equalisation, whereas the ImageJ source code [19] shows a good implementation.

4.3 Implementation

4.3.1 Collect histogram

An image is parsed where the frequency of intensities are added to an array of 256 levels of intensity.

```
for (i = 0; i < HEIGHT; i++) {
    for (int j = 0; j < WIDTH; j++) {
        histogramY[Y[i][j])]++;
        histogramU[U[i][j]]++;
        histogramV[V[i][j]]++;
    }
}
```

4.3.2 Equalise histogram

The histogram is then equalised, and a mapping between old intensity and equalised intensity is formed. This is stored in a look-up table.

```
public void getLUT(int[] histogram, byte[] lut)
{
     int levels = 256;
     int max = 255;
     range = 255;
     double sum;
     sum = getWeightedValue(histogram, 0);
     for (int i=1; i<max; i++)
          sum += 2 * getWeightedValue(histogram, i);
     sum += getWeightedValue(histogram, max);

     double scale = range/sum;
     double delta;

     lut[0] = icConstant.uInt2Byte(0);
     sum = getWeightedValue(histogram, 0);
     for (int i=1; i<max; i++) {
          delta = getWeightedValue(histogram, i);
          sum += delta;
          lut[i]        =        icConstant.uInt2Byte((int)
     Math.round(sum*scale));
          sum += delta;
     }
     lut[max] = icConstant.uInt2Byte(max);
}
```

Notice that each value in the original histogram is weighted. It has been shown that using the square root of histograms reduces the tendency for Histogram Equalisation to enhance meaningless detail and hide important but small high-contrast features (*Fig 4.1*).

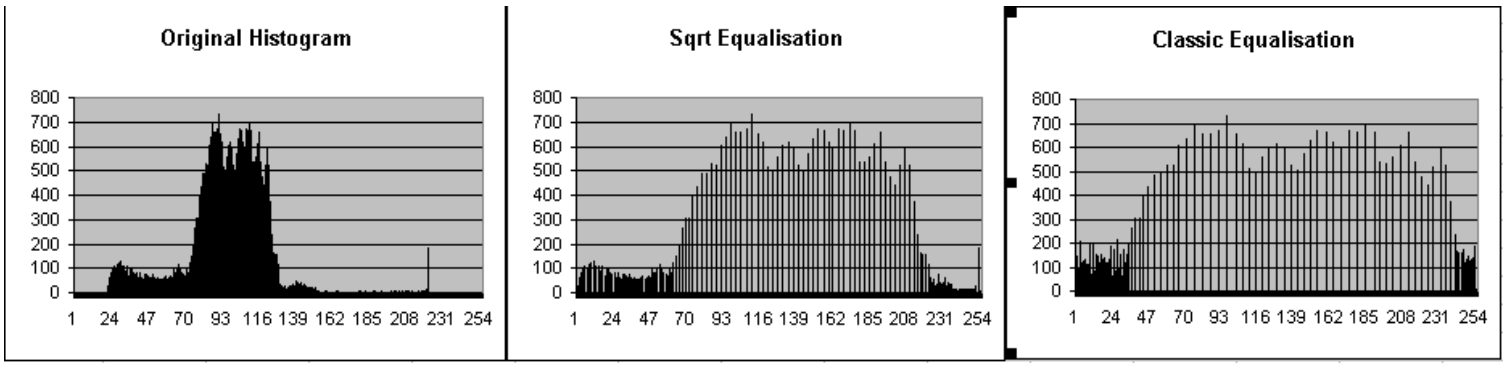


Fig 4.1 Comparison between classic equalisation and square root equalisation

4.3.3 Apply transformations

The look-up table of mappings are then applied to the image.

```
for (i = 0; i < HEIGHT; i++) {
    for (int j = 0; j < WIDTH; j++) {
        Y[j] = lutY[Y[j]];
        U[j] = lutU[U[j]];
        V[j] = lutV[V[j]];
    }
}
```

4.3.4 Assume temporal locality

It would be time consuming to parse an image twice to reapply transformations to the same image. If we assume temporal locality, that the difference between frames is negligible, then histograms collected in the past can be applied to future images, and this would make sure each image to only be parsed once.

Ten arrays are set up such that histograms over the past ten seconds are collected, and equalised once per second.

```
static const int MAX_FRAME = 30;
static const int MAX_SEC = 10;
static const int NUM_VAL = 256;

int frameCount = 0;
int secCount = 0;

long histY[MAX_SEC][NUM_VAL];
long histU[MAX_SEC][NUM_VAL];
long histV[MAX_SEC][NUM_VAL];

long* curHistY = histY[0];
long* curHistU = histU[0];
long* curHistV = histV[0];

uchar lutY[NUM_VAL];
uchar lutU[NUM_VAL];
uchar lutV[NUM_VAL];
```

```
frameCount = (frameCount + 1) % MAX_FRAME;
//stats for this frame
if(frameCount == 0) {
secCount = (secCount + 1)%MAX_SEC;
getLUT(histY, lutY);
getLUT(histU, lutU);
getLUT(histV, lutV);
curHistY = histY[secCount];
curHistU = histU[secCount];
curHistV = histV[secCount];
//clear last histogram
clearHist(curHistY);                    clearHist(curHistU);
clearHist(curHistV);
```

4.4 Results

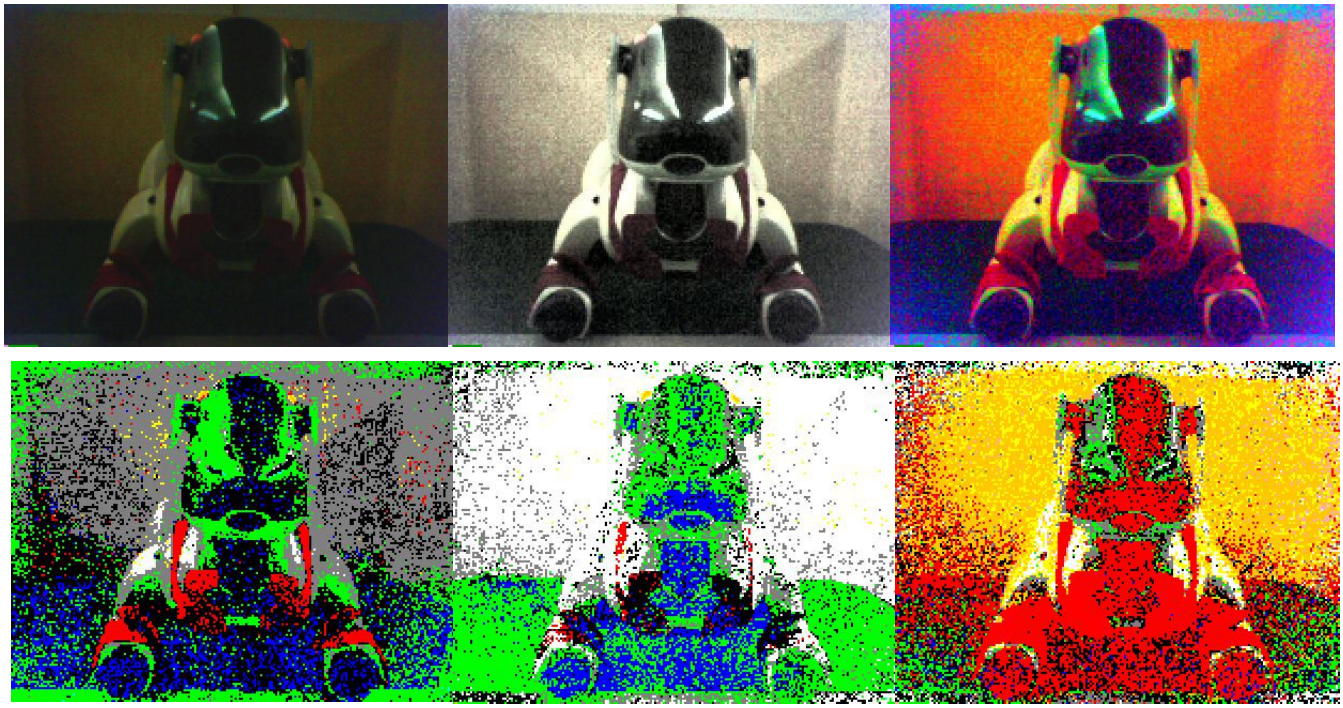


Figure 4.2 Histogram Equalisation

(a)Original dark image of red robot in yellow goal (b) Equalised Y plane only (c) Equalised all YUV planes (d)C plane of original image (e) C plane of equalised Y plane only (f) C plane of equalised all YUV planes

One way to combat problems associated with lighting changes, is to adjust the contrast of each image. Histogram equalisation was tested, and reliable contrast could only be generated for the Y plane. When histogram equalisation was only applied to the Y plane of a dark image (Figures 4.2a & 4.2d), objects in that image would be more recognisable to the human eye only (Figures 4.2b & 4.2e), they still lacked enough colour (UV) information to be recognised correctly by a robot. However, when all three YUV planes were equalised (Figures 4.2c & 4.2f), there was too much colour for a robot to recognise objects correctly.

4.5 Discussion/Conclusion

Although Histogram Equalisation is fast enough to be applied dynamically, and it can enhance the relative appearance of images, the colour component is hard to control. It is not enough to transform pixels under different lighting to the exact same colour. However, it might prove useful in a hybrid vision system where other information such as shape is used in conjunction with colour.

5 Edge Detection

5.1 Introduction

Due to manual mis-classifications of the look-up table and local changes in lighting, there is an increase in noise which makes colour no longer reliable enough to be the only means of segmentation and clustering. This section explores a shape/colour hybrid system, which uses shape as a basis for segmentation, and both shape and colour as bases for clustering.

To determine the shape of an object, one must determine its edges first. There are three approaches to edge detection.

(a) convolution based gradient operators, which use convolution matrices to detect dramatic changes in intensity (otherwise known as edges)

(b) techniques such as the Canny Edge Detector[23] which build statistical models of gradients to determine edges

(c) region-filling techniques such as Region Growing and Watershed Segmentation, which extract boundaries of adjoining regions.

Statistical models and region filling techniques are usually too computationally expensive to be applied dynamically, hence only convolution based edge detectors are further explored in this section.

Chapter 10 of Gonzalez [23] is a good reference on gradient operators and region-filling techniques.

5.2 Concept

Edges can be detected by using gradient operators such as the Roberts operator or the Sobel operator on an image. A pair of operators (*a,b*) are masked over sections of an image such that:

$$gradient = |a| + |b|$$

Gonzalez[3] Section 10.1 is a good reference on edge detection using convolution based operators.

−1	0	0	−1
0	1	1	0

Fig 6.1 Roberts Operator

−1	−2	−1	−1	0	1
0	0	0	−2	0	2
1	2	1	−1	0	1

Fig 6.2 Sobel Operator

If the gradient of the intensity of a pixel is higher than a certain threshold (determined through trials and error using *ic*), then it is labelled as an edge pixel. Then regions enclosed by edge pixels can be clustered and identified based on average colour. This determines a more accurate boundary of objects, and compensates for minor noise caused by misclassifications of the colour table.

5.3 Implementation

5.3.1 Edge pixels

Apply gradient operators to determine edge pixels:

Roberts:

```
If   (abs(pixel[w][h] - (pixel[w+1][h+1]) +
      abs(pixel[w+1][h] - pixel[w][h+1])
      >= threshold)
      pixel[w][h].isEdge = true
```

Sobel:

```
If   (abs(pixel[w-1][h+1]     +     2*pixel[w][h+1])     +
pixel[w+1][h+1] -
         pixel[w-1][h-1] – 2*pixel[w][h-1] - pixel[w+1][h-
1])) +
      abs(pixel[w+1][h-1]     +     2*pixel[w+1][h])     +
pixel[w+1][h+1] -
         pixel[w-1][h-1]   –   2*pixel[w-1][h]   -   pixel[w-
1][h+1]))
      >= threshold)
      pixel[w][h].isEdge = true
```

5.3.2 Edge runs

To find edge runs, edge pixels need to be run length encoded. The code below demonstrates a simple implementation based on the state of the previous and current pixel in a row.

```
for (y = 0; y < HEIGHT; y++) {
     start = 0;
     for (x = 0; x < WIDTH; x++) {
if(!prevEdge && (curEdge || x==end))
               saveSegment(y, start, x-1)
else if(prevEdge && (!curEdge || x==end))
               start = x //reset start
     }
}
```

For actual implementation, see `trunk/robot7/vision/BlobEdge.cc`

5.3.3 Edge blobs

Edge runs are clustered into edge blobs using the same disjoint blob algorithm used to cluster colour blobs (for details see 2004 rUNSWift Thesis A Report[22] and `trunk/robot7/vision/Blob.cc`), the only difference is in the overlap condition.

```
bool overlaps(eRunLengthInfo *segment) {//segment is in previous row
    if(this->isEdge)
       return this->color == segment->color &&
             this->startIndex <= segment->endIndex &&
             this->endIndex >= segment->startIndex;
    else if(!segment->isEdge) //&& !this->isEdge
       return this->startIndex <= segment->endIndex-MIN_OVERLAP_PIXELS &&
             this->endIndex >= segment->startIndex+MIN_OVERLAP_PIXELS;
    else //!this->isEdge && segment->isEdge
       return false;
}
```

5.4 Results

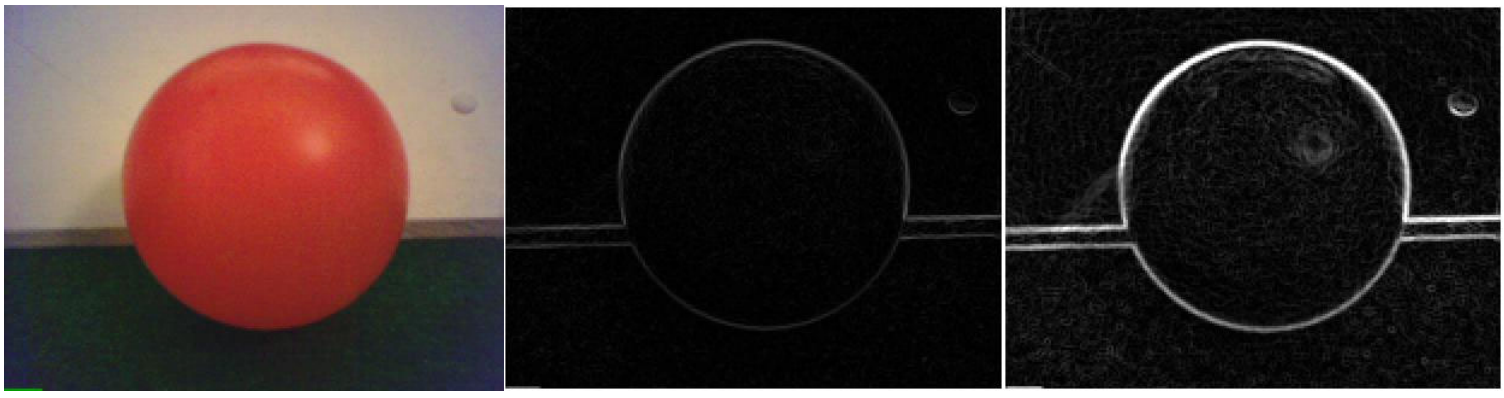


Fig 5.3 (a) original image (b) applied Roberts operator in Y plane (c) applied Sobel operator in Y plane

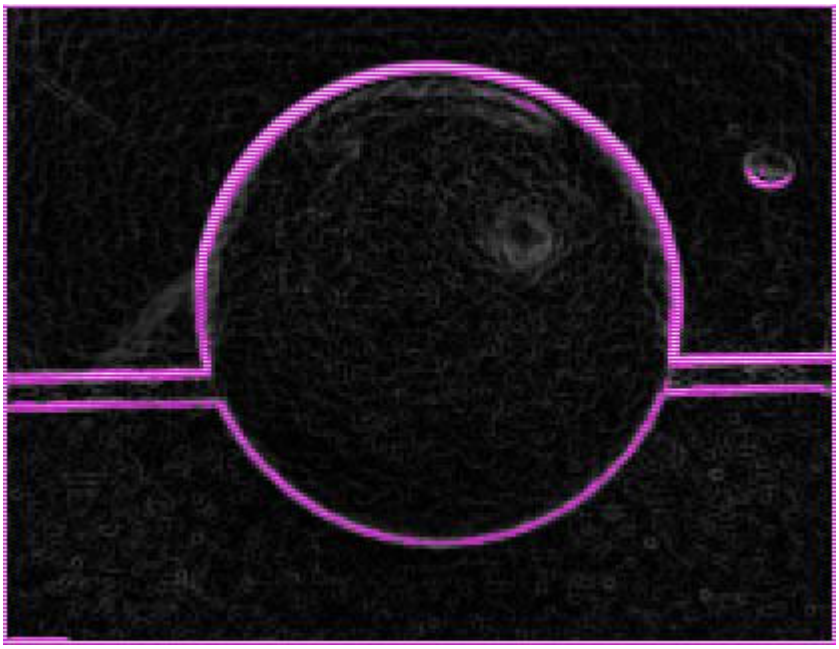


Fig 5.4 image thresholded at Y>=75 after application of Sobel operator in Y plane

Due to time constraints, section 5.3.3 has not been fully tested. Intermediate results of edge runs show that most of the edges of distinct objects of a certain size, which stand out from the background, are classified correctly most of the time. But small discontinuities in the boundaries of regions are frequent, which would cause the blobbing algorithm to join two regions together incorrectly.

5.5 Discussion/Conclusion

Edge detection using gradient operators are good for finding most edges around objects, but small discontinuities cause regions to not be fully enclosed. Algorithms which join broken lines can be expensive, and do not guarantee all gaps would be joined. The current clustering algorithm would not leave clusters intact unless it is fully enclosed, therefore this method is not fit for purpose.

6 Discussion/Conclusion

rUNSWift's current vision system is highly dependent on colour, its advantages are that its fast, and it has been reasonably accurate in the past. Its disadvantages are that it is sensitive to manual misclassifications, noise, and changes in lighting. These problems are enhanced by the poor quality of the ERS-7 model.

Investigations into linear filters and histogram equalisation show that additional information is required to compensate for the unreliability of colour. Investigations into edge detection show that simple gradient edge detectors are not good enough to determine the shape of an object when tested in non-ideal environments.

More effective methods such as region-filling techniques are too computationally expensive to be implemented without loss of information. It might be worthwhile to attempt partial region filling where the seeds can be based on colours of high importance (eg. orange).

It might also be beneficial to profile all modules in detail to see exactly how much more time can be spent on vision.

If the vision system can yield more accuracy, then perhaps not all information need to be processed, which perhaps would leave enough time to use the more expensive but accurate techniques occasionally. A fast method to determine the difference between two consecutive frames may be useful in this case so that only new information processed extensively.

The rUNSWift vision system still has a long way to go to fulfil its purpose of speed, accuracy and completeness.

7 References

[1] Olave A., Wang D., Wong J., Tam T., Leung B., Kim M.S., Brooks J., Chang A., Huben N.V., Sammut C. Hengst, B. *The UNSW RoboCup 2002 Legged League Team* (UNSW 2002 RoboCup Team Thesis), University of New South Wales, 2002

[2] Chen J., Chung E., Edwards R., Wong N., Hengst B ,.Sammut C., Uther W. *Rise of the AIBOs III – AIBO Revolutions* (UNSW 2003 RoboCup Team Thesis), University of New South Wales, 2003.

[3] Gonzalez R.C., Woods R.E., *Digital image processing* 2nd ed, Prentice Hall, 2002.

[4] Forsyth D.A., Ponce J., *Computer vision : a modern approach,* Prentice Hall, 2003

[5] *AIBO forum*
http://openr.aibo.com/openr/end/index.php4

[6] Roefer, T., *Quality of ERS-7 Camera Images*
http://www.informatik.uni-bremen.de/~roefer/ers7/

[7] *Electric Fields*
http://www.glenbrook.k12.il.us/gbssci/phys/Class/estatics/u814c.html

[8] Robb, G.R.M., *Electric Field Java Demo Program*
http://local.phys.strath.ac.uk/12-157/lab8/Efield.java

[9] Koelher, *Electric Potential*
http://www.rwc.uc.edu/koehler/biophys/4b.html

[10] *Study of the effect of electric fields on E. coli*
http://chemcases.com/cisplat/cisplat01.htm

[11] Bruce J., Balch T., Veloso M., *Fast and inexpensive color image segmentation for interactive robots*, Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2000.

[12] Dahm I., Deutsch S., Hebbel M., Osterhues A., *Robust Color Classification for Robot Soccer*, University of Dortmund, 2003.

[13] Veloso, Uther, Fujita, Asada, Hitano, *Playing soccer with legged robots*, Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1998.

[14] Dalgliesh J., Lawther M., *Playing Soccer with Quadruped Robots*, Computer Engineering Thesis, University of New South Wales, 1999.

[15] Hengst B., Ibbotson D., Pham S.B., Sammut C., *The UNSW United 2000 Sony Legged Robot Software System*, University of New South Wales, 2000.

[16] Quilan, J.R., *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[17] Chen S., Siu M., Vogelgesang T., Yik T.F., Hengst B., Pham S.B., Sammut C., *The UNSW RoboCup 2001 Sony Legged League Team*, University of New South Wales, 2001.

[18] *Hypermedia Image Processing Resources*
http://homepages.inf.ed.ac.uk/rbf/HIPR2/

[19] *ImageJ*
http://rsb.info.nih.gov/ij/docs/menus/process.html

[20] Cameron D., Barnes N., *Knowledge-based Autonomous Dynamic Colour Calibration*, University of Melbourne, 2003.

[21] Manduchi R., *Learning Outdoor Color Classification from Just One Training Image*, University of California, Santa Cruz, 2004.

[22] Chan K., Lam D., Pham K., Whaite D., Wong T., Xu J., *2004 UNSW Robocup Team Thesis A Report*, University of New South Wales, 2004.

[23] *Canny Edge Detector*
http://www.cee.hw.ac.uk/hipr/html/canny.html

8 Appendix A – Proof of Quadratic Equation of Chromatic Distortion Model

$z=ax^2 + bx + c$ [1]

Subtitute (x_0,z_0), (x_1,z_1), and (x_2,z_2) into [1]:

$z_0=ax_0^2 + bx_0 + c$ [2]

$z_1=ax_1^2 + bx_1 + c$ [3]

$z_2=ax_2^2 + bx_2 + c$ [4]

Combine [2], [3] and [4]:

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \end{pmatrix} = a\begin{pmatrix} x_0^2 \\ x_1^2 \\ x_2^2 \end{pmatrix} + b\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} + c\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\left(\begin{array}{ccc|c} x_0^2 & x_0 & 1 & z_0 \\ x_1^2 & x_1 & 1 & z_1 \\ x_2^2 & x_2 & 1 & z_2 \end{array}\right) \Rightarrow \left(\begin{array}{cccc} 1 & \frac{1}{x_0} & \frac{1}{x_0^2} & \frac{z_0}{x_0^2} \\ x_1^2 & x_1 & 1 & z_1 \\ x_2^2 & x_2 & 1 & z_2 \end{array}\right) R1 \div x_0^2$$

$$\Rightarrow \left(\begin{array}{cccc} 1 & \frac{1}{x_0} & \frac{1}{x_0^2} & \frac{z_0}{x_0^2} \\ 0 & x_1 - \frac{x_1^2}{x_0} & 1 - \frac{x_1^2}{x_0^2} & z_1 - \frac{x_1^2 z_0}{x_0^2} \\ 0 & x_2 - \frac{x_2^2}{x_0} & 1 - \frac{x_2^2}{x_0^2} & z_2 - \frac{x_2^2 z_0}{x_0^2} \end{array}\right) \begin{array}{l} R2 - x_1^2 R1 \\ R3 - x_2^2 R1 \end{array}$$

$$\Rightarrow \left(\begin{array}{cccc} 1 & \frac{1}{x_0} & \frac{1}{x_0^2} & \frac{z_0}{x_0^2} \\ 0 & 1 & \frac{1 - \frac{x_1^2}{x_0^2}}{x_1 - \frac{x_1^2}{x_0}} & \frac{z_1 - \frac{x_1^2 z_0}{x_0^2}}{x_1 - \frac{x_1^2}{x_0}} \\ 0 & x_2 - \frac{x_2^2}{x_0} & 1 - \frac{x_2^2}{x_0^2} & z_2 - \frac{x_2^2 z_0}{x_0^2} \end{array}\right) R2 \div \left(x_1 - \frac{x_1^2}{x_0}\right)$$

$$\Rightarrow \left(\begin{array}{cccc} 1 & \frac{1}{x_0} & \frac{1}{{x_0}^2} & \frac{z_0}{{x_0}^2} \\ 0 & 1 & \frac{1-\frac{{x_1}^2}{{x_0}^2}}{x_1-\frac{{x_1}^2}{x_0}} & \frac{z_1-\frac{{x_1}^2 z_0}{{x_0}^2}}{x_1-\frac{{x_1}^2}{x_0}} \\ 0 & 0 & \left(1-\frac{{x_2}^2}{{x_0}^2}\right)-\left(x_2-\frac{{x_2}^2}{x_0}\right)\left(\frac{1-\frac{{x_1}^2}{{x_0}^2}}{x_1-\frac{{x_1}^2}{x_0}}\right) & \left(z_2-\frac{{x_2}^2 z_0}{{x_0}^2}\right)-\left(x_2-\frac{{x_2}^2}{x_0}\right)\left(\frac{z_1-\frac{{x_1}^2 z_0}{{x_0}^2}}{x_1-\frac{{x_1}^2}{x_0}}\right) \end{array} \right) R3-\left(x_2-\frac{{x_2}^2}{x_0}\right)R2$$

$$c\left(1-\frac{{x_2}^2}{{x_0}^2}\right)-\left(x_2-\frac{{x_2}^2}{x_0}\right)\left(\frac{1-\frac{{x_1}^2}{{x_0}^2}}{x_1-\frac{{x_1}^2}{x_0}}\right)=z\left(y_2-\frac{{x_2}^2 y_0}{{x_0}^2}\right)-\left(x_2-\frac{{x_2}^2}{x_0}\right)\left(\frac{z_1-\frac{{x_1}^2 z_0}{{x_0}^2}}{x_1-\frac{{x_1}^2}{x_0}}\right)$$

$$c=\frac{\left(z_2-\frac{{x_2}^2 z_0}{{x_0}^2}\right)-\left(x_2-\frac{{x_2}^2}{x_0}\right)\left(\frac{z_1-\frac{{x_1}^2 z_0}{{x_0}^2}}{x_1-\frac{{x_1}^2}{x_0}}\right)}{\left(1-\frac{{x_2}^2}{{x_0}^2}\right)-\left(x_2-\frac{{x_2}^2}{x_0}\right)\left(\frac{1-\frac{{x_1}^2}{{x_0}^2}}{x_1-\frac{{x_1}^2}{x_0}}\right)}$$

$$b + c\left(\frac{1-\frac{x_1^{\,2}}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right) = \left(\frac{z_1-\frac{x_1^{\,2}z_0}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right)$$

$$b = \left(\frac{z_1-\frac{x_1^{\,2}z_0}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right) - c\left(\frac{1-\frac{x_1^{\,2}}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right)$$

$$b = \left(\frac{z_1-\frac{x_1^{\,2}z_0}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right) - \left(\frac{1-\frac{x_1^{\,2}}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right)\left(\frac{\left(z_2-\frac{x_2^{\,2}z_0}{x_0^{\,2}}\right)-\left(x_2-\frac{x_2^{\,2}}{x_0}\right)\left(\frac{z_1-\frac{x_1^{\,2}z_0}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right)}{\left(1-\frac{x_2^{\,2}}{x_0^{\,2}}\right)-\left(x_2-\frac{x_2^{\,2}}{x_0}\right)\left(\frac{1-\frac{x_1^{\,2}}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right)}\right)$$

$$a + \frac{b}{x_0} + \frac{c}{x_0^{\,2}} = \frac{z_0}{x_0^{\,2}}$$

$$a = \frac{z_0}{x_0^{\,2}} - \frac{b}{x_0} - \frac{c}{x_0^{\,2}}$$

$$a = \frac{z_0}{x_0^{\,2}} - \frac{1}{x_0}\left(\left(\frac{z_1-\frac{x_1^{\,2}z_0}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right) - \left(\frac{1-\frac{x_1^{\,2}}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right)\left(\frac{\left(z_2-\frac{x_2^{\,2}z_0}{x_0^{\,2}}\right)-\left(x_2-\frac{x_2^{\,2}}{x_0}\right)\left(\frac{z_1-\frac{x_1^{\,2}z_0}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right)}{\left(1-\frac{x_2^{\,2}}{x_0^{\,2}}\right)-\left(x_2-\frac{x_2^{\,2}}{x_0}\right)\left(\frac{1-\frac{x_1^{\,2}}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right)}\right)\right)$$

$$-\frac{1}{x_0^{\,2}}\left(\frac{\left(z_2-\frac{x_2^{\,2}z_0}{x_0^{\,2}}\right)-\left(x_2-\frac{x_2^{\,2}}{x_0}\right)\left(\frac{z_1-\frac{x_1^{\,2}z_0}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right)}{\left(1-\frac{x_2^{\,2}}{x_0^{\,2}}\right)-\left(x_2-\frac{x_2^{\,2}}{x_0}\right)\left(\frac{1-\frac{x_1^{\,2}}{x_0^{\,2}}}{x_1-\frac{x_1^{\,2}}{x_0}}\right)}\right)$$

Substitute a, b, c into [1]:

$$\therefore z = \left[\frac{z_0}{{x_0}^2} - \frac{1}{x_0} \left(\frac{z_1 - \frac{{x_1}^2 z_0}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right) - \left(\frac{1 - \frac{{x_1}^2}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right) \left(\frac{\left(z_2 - \frac{{x_2}^2 z_0}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{z_1 - \frac{{x_1}^2 z_0}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)}{\left(1 - \frac{{x_2}^2}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{1 - \frac{{x_1}^2}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)} \right) \right]$$

$$- \frac{1}{{x_0}^2} \left[\frac{\left(z_2 - \frac{{x_2}^2 z_0}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{z_1 - \frac{{x_1}^2 z_0}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)}{\left(1 - \frac{{x_2}^2}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{1 - \frac{{x_1}^2}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)} \right] x^2$$

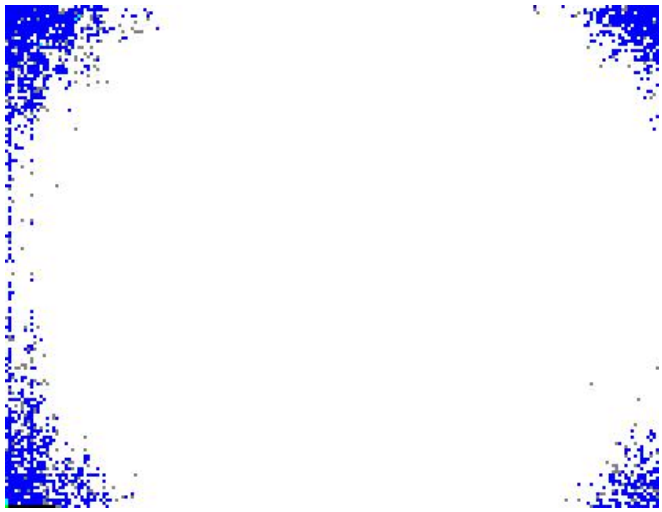
$$+ \left[\left(\frac{z_1 - \frac{{x_1}^2 z_0}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right) - \left(\frac{1 - \frac{{x_1}^2}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right) \left(\frac{\left(z_2 - \frac{{x_2}^2 z_0}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{z_1 - \frac{{x_1}^2 z_0}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)}{\left(1 - \frac{{x_2}^2}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{1 - \frac{{x_1}^2}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)} \right) \right] x$$

$$+ \frac{\left(z_2 - \frac{{x_2}^2 z_0}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{z_1 - \frac{{x_1}^2 z_0}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)}{\left(1 - \frac{{x_2}^2}{{x_0}^2} \right) - \left(x_2 - \frac{{x_2}^2}{x_0} \right) \left(\frac{1 - \frac{{x_1}^2}{{x_0}^2}}{x_1 - \frac{{x_1}^2}{x_0}} \right)}$$

9 Appendix B – Comparison Between Unprocessed Images and Ring Corrected Images

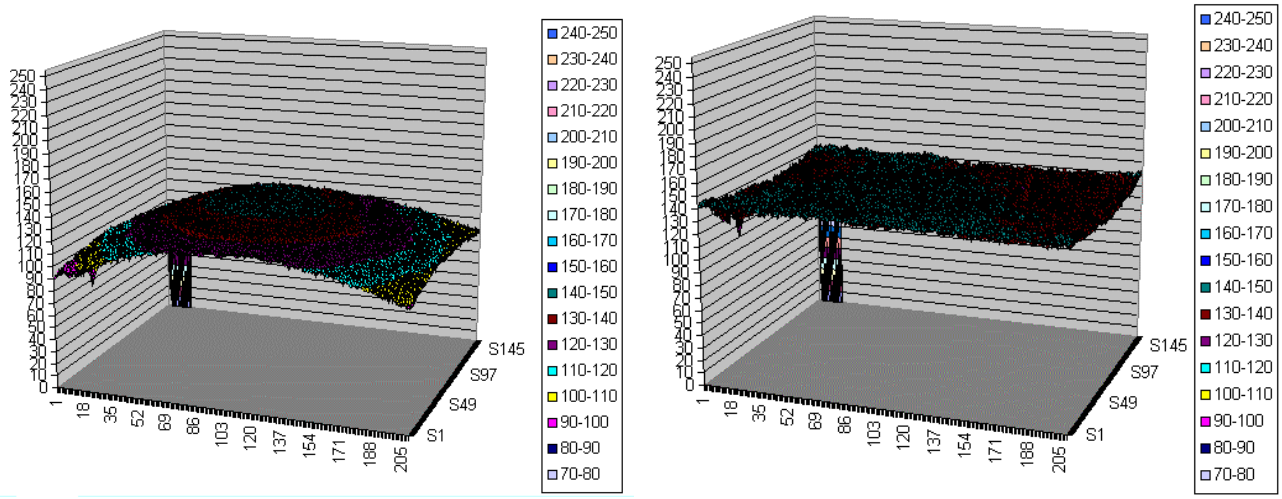
YUV plane of white border – unprocessed

YUV plane of white border – processed

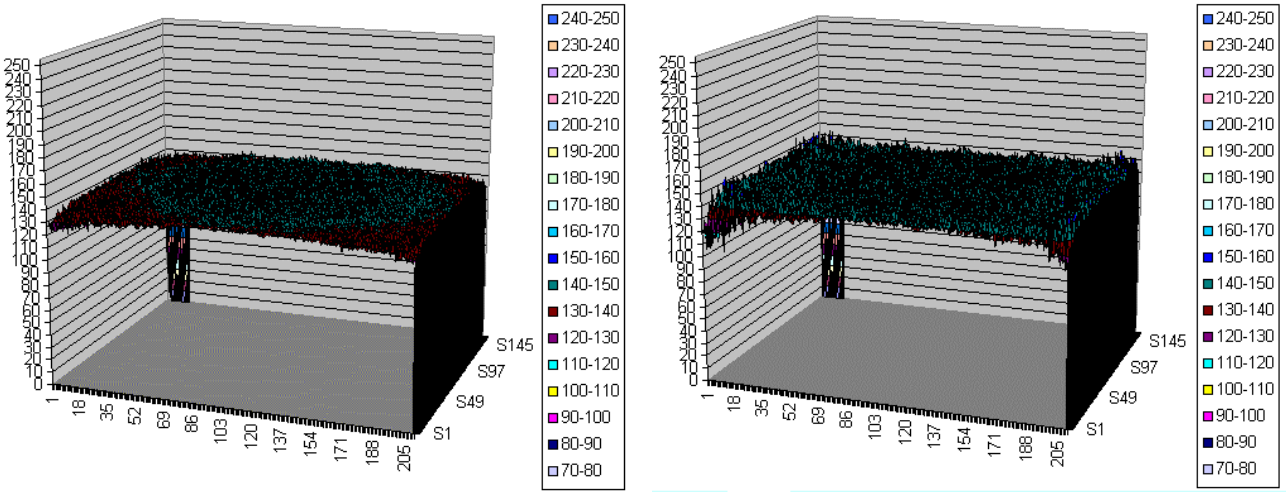


C plane of white border – unprocessed

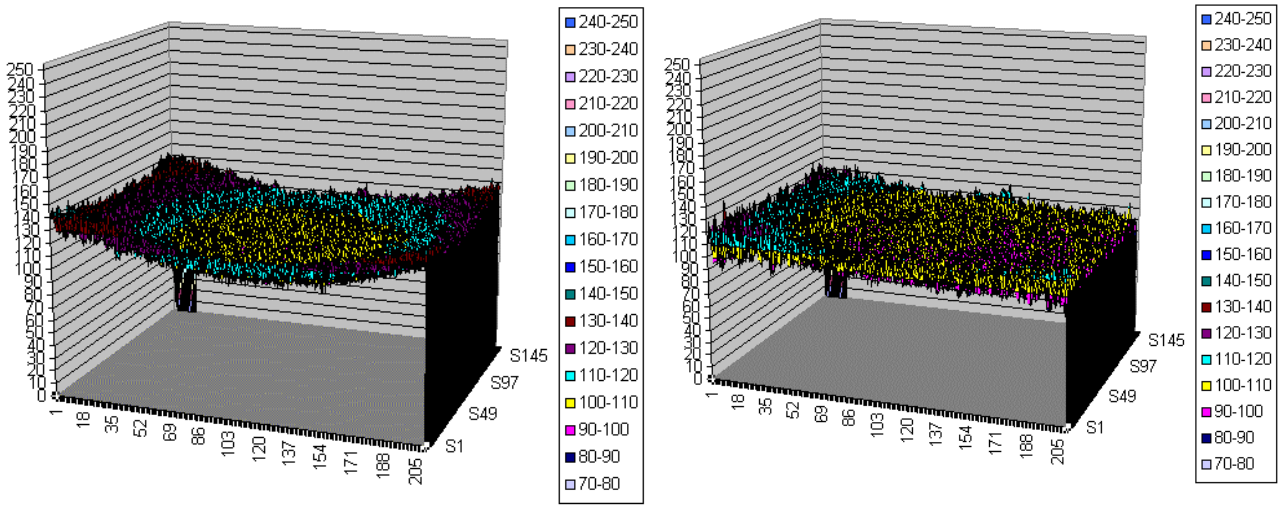
C plane of white border – processed



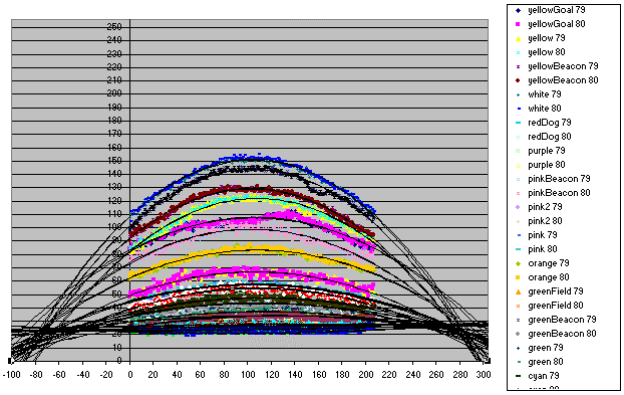
Y plane of white border – unprocessed **Y plane of white border – processed**



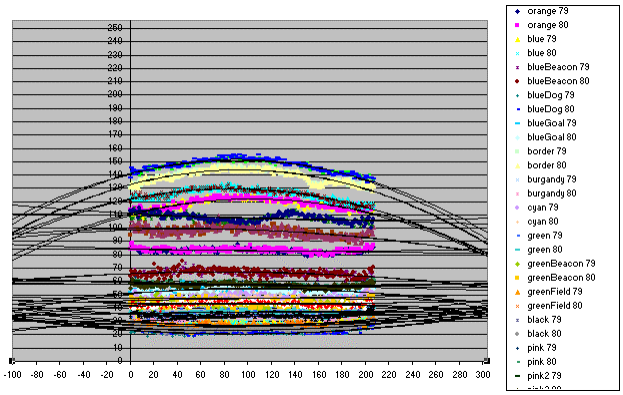
U plane of white border – unprocessed **U plane of white border – processed**



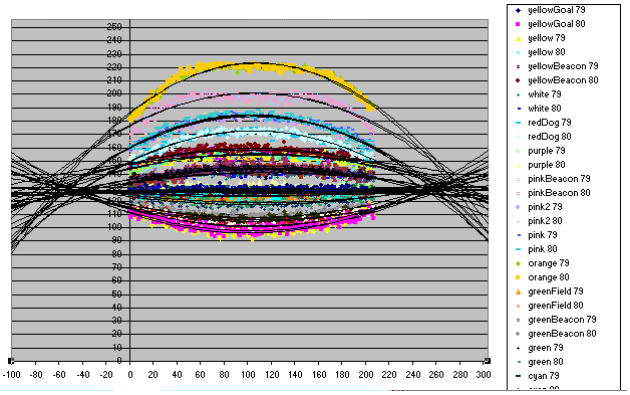
V plane of white border – unprocessed **V plane of white border – processed**



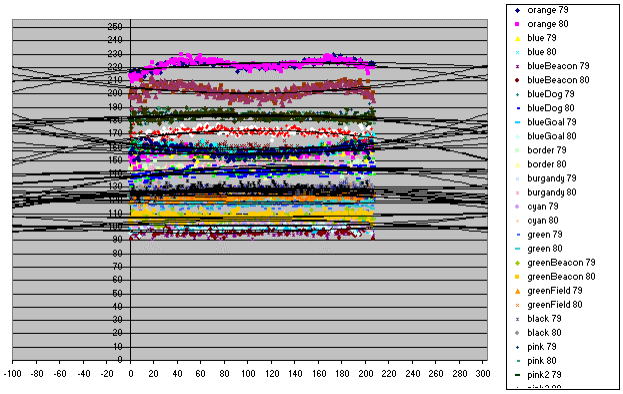
Y plane of uniform colours at Y=79 & Y=80 – unprocessed



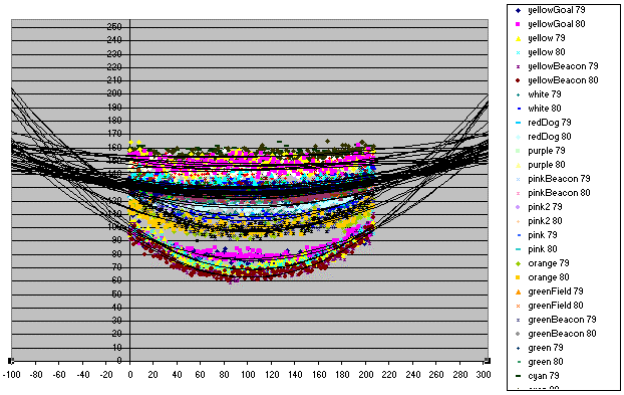
Y plane of uniform colours at Y=79 & – processed



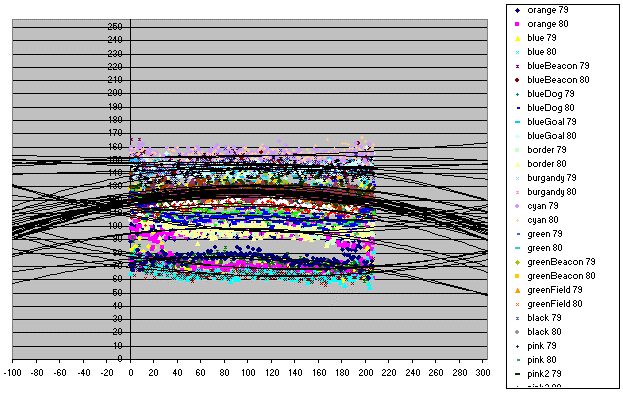
U plane of uniform colours at Y=79 & Y=80 – unprocessed



U plane of uniform colours at Y=79 & – processed



V plane of uniform colours at Y=79 & Y=80 – unprocessed



V plane of uniform colours at Y=79 & – processed