THE UNIVERSITY OF NEW SOUTH WALES
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

The 6th Sense:

I see red people

(As balls)

Andrew Owen

Thesis submitted as a requirement for the degree

Bachelor of Science (Computer Science) Honours

Submitted: September 7, 2006

Supervisor: Dr William Uther

Assessor: Professor Claude Sammut

Acknowledgements

Thanks to Mum for proof reading[1], Dad for getting me home, Lyn for keeping me fed and Rod for helping with results.

Thanks especially to Will for your great guidance and advice[2] and for not going too insane, Brad for making things happen and Claude for your valuable insights. To Eric, Michael, Oleg and Ryan – of all the robotic sporting teams I've been in, I can't think of one I've enjoyed more than this one. Thanks for the good times.

Thanks also to anyone who reads this through, it took a fair bit of time to do, so it is nice to know that at least someone will make use of it.

[1] And anyone else who did it while I was asleep.

[2] I followed your advice about making this about 100 pages too.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The ultimate goal of the RoboCup 4-Legged League is to produce robot dogs that can play a variant of soccer which is as close to human soccer as possible. For realistic soccer to take place, we need to have robots which can use their camera just like a person can use their eyes. All teams are faced with this same problem, so many different approaches have been taken. One common thread throughout all of these approaches is the use of colour to detect and recognise objects. This paper will look at the changes to the rUNSWift vision system since the beginning of the year.

The rUNSWift team has existed in some form since 1999. In 2006 there were 5 undergraduates actively working on the rUNSWift codebase under a single supervisor. Each person would be responsible for shifting the blame when their particular area of the code appeared to fail. This was typically done by correcting the problem, though other approaches were not unknown. So progress is made by continually improving the weakest section of the system, thereby exposing another weakness. The vision system at the beginning of the year was in no way a weakness, however it was noted that the time taken for calibrating the vision system for a new field was longer than necessary. By streamlining this process, we were able to significantly improve the calibration time leading to an increase in the time available for other tasks which required a functioning vision system.

1.1 Competitions

There were two major events which took place this year for the rUNSWift team. The Australian Open, held in the rUNSWift lab in April and the World Championship, held in Bremen in June. The majority of the work described in this report was done in the time between the Australian Open and the end of the World Championship.

1.2 Notes on Language

The term 'colour' is used frequently in this paper to mean both symbolic colour (e.g. 'red', 'orange', etc) or YUV colour (e.g. a set of 3 integers, representing the colour of a pixel as reported by the camera). When not explicitly specified, there should be only a single interpretation which makes sense.

The terms 'dog' and 'robot' are used interchangeably, and often when we speak about a robot, we are mainly speaking about the camera in the particular robot. The mixture of words provides some variety, and shouldn't be taken to have any significance.[1]

1.3 Overview

This paper seeks to give a description of the changes made to the rUNSWift vision system in 2006. The main area of focus was colour classification and the related tools. The colour classifier is described in detail in Section 2.2, and details regarding the strategies used for handling difficult colour boundaries are given in Section 2.3. A detailed evaluation of the vision system used in Bremen is given in Section 2.4.

Prior to the Australian Open, some work was done on primitive visual robot recognition for the passing challenge. Although the method developed was never reliable enough for being used in competition, it is described in Chapter 4.

[1] This also allows us to avoid rhyming 'dog' with 'log'.

1.4 Vision Systems

Although effort was concentrated on smaller parts of a whole system, this paper will still primarily be concerned with the performance of the vision system as a whole – the system taking raw input from a camera and outputting the objects recognised. We take a holistic approach to evaluation because the interaction between the different parts of the vision system is too complicated to be able to effectively isolate the performance of one component, and different parts of the system will tolerate different kinds of errors.

When comparisons are made between different components, we will compare the components in the context of the whole rUNSWift vision system, thus each variation is treated as a system of its own. This has the advantage of giving a good measurement for real-world performance, but results when integrated into different systems may vary depending on the behaviour of other components in the system.

1.5 The rUNSWift Vision System

Fortunately, the rUNSWift vision system was already strong before work began on it this year. The different components are described in detail in papers by previous members of the rUNSWift team. The low level vision processing is based on components described in [6] and [14], while the high level processing is described fully in [9]. A brief description of the entire rUNSWift vision system is given in this section describing which components exist and how they fit together. This is shown diagrammatically in Figure 1.1.

1.5.1 Ring Correction

Due to the construction of the cameras in the ERS-7 robots, there is ring shaped distortion present in every image. This work is described in [14] however this year the calculation of the correction function was modified slightly. The ring correction is implemented using 3 look-up tables, indexed by distance from the centre of the image and a Y, U or V component. The resulting image is obviously improved, however evidence of the ring is still present.

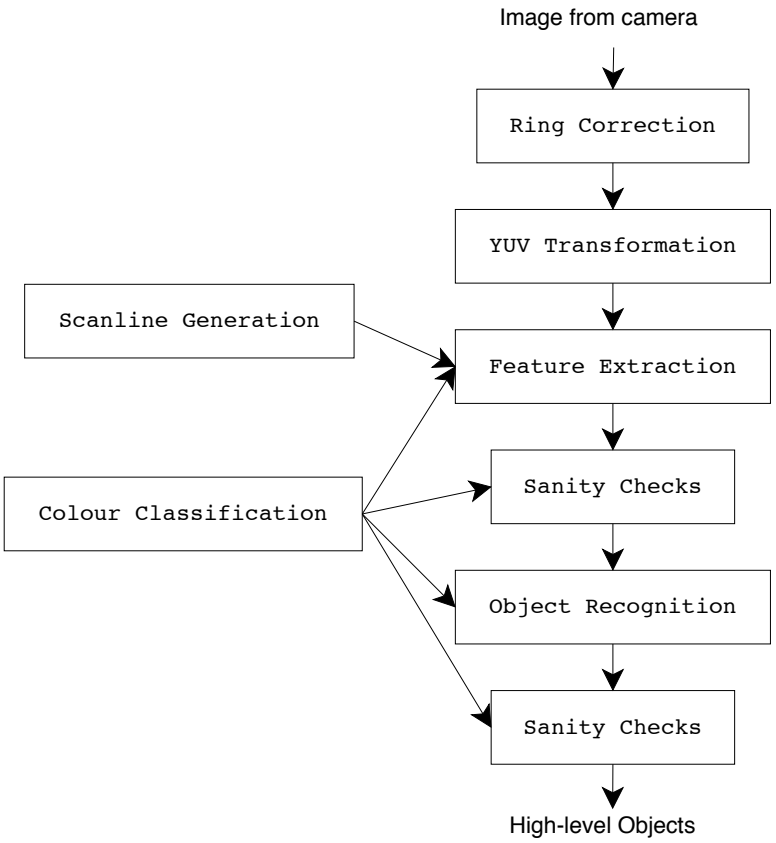

Figure 1.1: An overview of the components in the rUNSWift vision system.

1.5.2 YUV Transformation

It has been found that different cameras behave differently to the same light. This changes how colours will be classified near the boundaries between symbolic colours. We transform each Y, U and V component according to a linear function, with values traditionally calculated by choosing a "base dog" with no transformation applied, then comparing images of block colours taken with this dog to other dogs and fitting a line between the points in colour space. This process is tedious and error prone,[2] however the results can improve the quality of the colour classification if done correctly.

1.5.3 Scanline Generation

The vision system only processes a fraction of the pixels in the image. This component of vision generates a selection of lines to process in the feature extraction component, it is not based at all on the information from the camera but uses the joint sensors and some high-level behaviour information (for example when the ball is grabbed, less vertical scan-lines are needed but more horizontal scan-lines are needed to see the goal). The scanline generation algorithm tries to make lines which are likely to pass through interesting areas, without wasting too much time on areas that won't give much information gain. When the horizon calculation (based on joint angles) is correct then this works very well.

1.5.4 Colour Classification

The colour classifier takes a YUV triple and converts it to a symbolic colour. The symbolic colours used in the rUNSWift vision system are described in Section 2.3.2. The classifier is implemented as a look-up table, indexing YUV triplets to symbolic colours. Some support for using different classification functions for different purposes is in the code but unused at present.

[2]The images need to be taken with constant lighting, so we would only take logs at night. The dogs needed to be placed in the exact same positions, and people walking past would change the lighting.

1.5.5 Feature Extraction

By looking at pixels along the scanlines, we can pick up different features in the image. We detect features by edges in YUV space, and also by runs of certain blocks of classified colours. Different scanlines are treated differently, since detecting ball features above the horizon is likely to introduce false ball readings, and likewise for detecting beacons below the horizon. Full descriptions of the detection of these features can be found in [9]. The features which we detect are:

Ball A point on an edge of the ball. Detected by either a characteristic jump in YUV space or by the presence of the ball orange symbolic colour in certain situations (care is taken here to avoid seeing a red robot as a ball).

Beacon A beacon feature is a line of pink, the other coloured part of a beacon is not detected as a beacon feature. These are detected purely though runs of the pink symbolic colour in the image.

Goal A goal feature is like a beacon feature, but for different colours.

Field A patch of green field. These are detected by the symbolic green colour. These features are used for simple checks to see whether we are on the field or not, so only a few locations are sampled for field features.

Field line The edge of a field-line. These are detected by jumps in YUV space.

Field edge The point where the field meets the background. These are detected by jumps in YUV space.

Obstacle A shadow on the field, typically cast by another robot or the referee. These are detected by jumps in YUV space.

1.5.6 Object Recognition

The list of features found in feature extraction is then processed to find objects which fit the features found. The features are firstly grouped by location, to hopefully group all the features

for a particular object together. The rules for grouping features depend on the type of feature (for example, goal features will be grouped together if they appear at a similar height, even if their heading is quite different). Once the features are grouped, the groups are processed to see what object they represent and to get distance and heading information for that object.

Beacon Recognition

Since a beacon is only detected by the presence of pink, we then check above and below the pink for the other coloured section of the beacon to identify which beacon was found. The distance to the beacon is estimated using a combination of the area and height of the beacon. The direction to the beacon is estimated by the position of the beacon in the image.

Ball Recognition

The ball features are fit to a ball by considering every possible triplet of features (up to a certain maximum) and fitting a circle to the features. Medians for each circle parameter are taken and the distance to the ball is calculated by the radius of the circle which is fit.

Goal Recognition

A goal is recognised as just the bounding box containing all of its features. However, some extra processing is done to find a gap in the goal, so when shooting on goal, we can aim to space. This processing is done by scanning a line along the bottom of the goal, and recording where it is interrupted.

1.5.7 Sanity Checks

At various stages in the processing, we apply simple checks to ensure that the scene being recognised is consistent with what we know about the environment. For example, we don't allow the ball to be seen above a goal or a beacon and certain combinations of objects are impossible to see in a frame (such as beacons from opposite corners of the field). All of these checks are designed manually and added only when problems are evident – a more general approach to writing the higher level sanity checks is described in [1], however the rUNSWift

checks also include lower level checks, such as the presence of various symbolic colours in the vicinity of an object (for example, a certain amount of green below a goal is required for the goal to be detected).

1.6 Areas Changed

The time spent on the vision system was spent improving many different parts. In this paper we will describe the changes made in the following areas:

1.6.1 Tools

The vision calibration tools were changed extensively. The tool used for training the colour classifier was merged with the offline simulation of vision. This was done to speed up the calibration process, and give a better method for evaluating the vision system as a whole when building the classification function.

1.6.2 Colour Classification

The learning algorithm for colour classification was modified slightly. This was done to allow modification of smaller parts of the classification function without having the change affect other colours nearby in YUV space.

1.6.3 Beacon Detection

Beacon recognition was modified somewhat in Bremen in response to the different shade of pink. The symbolic colours for red and pink were merged together, and some shape recognition was added to the beacon feature grouping code, to ensure that any beacon features grouped together were shaped roughly like a rectangle.

The calculation of beacon distances was modified to use different constants for beacons on one side of the field. The exact reasons for this being needed are unknown, but it enabled much more accurate positioning of the goalie and other players.

1.6.4 Camera Specific Transformations

It has been known that different cameras respond differently to the same light, so we have used YUV transformations on the images to attempt to calibrate all cameras to a single reference. The offline vision tool was modified to allow fast evaluation of different transformations, which allowed these transformations to be done by hand (which was much faster and less error prone than the previous automatic methods).

Chapter 2

Colour Classification

The camera tells us the Y, U and V components for a single pixel, however this is often too much information for us to process. To simplify calculations, we often want to know what a particular triplet of Y, U and V actually represents. We therefore write a colour classifier which is a function that takes the YUV triple for a single pixel (possibly after some preprocessing) and returns a symbolic colour. Often, a look-up table (often called a 'colour table' in this context) is used to perform the classification, allowing complicated classification algorithms to be used without a performance hit when being used on a dog.[1] Since there is an easy and efficient method for representing the function, the main problem is then the actual construction of this function. It is infeasible to manually assign a symbolic colour to every possible YUV value, so some form of generalising classifier must be used.

2.1 Previous Work

All teams in RoboCup make use of colour classification in some way. Although some teams (including rUNSWift as described in [9]) make use of techniques where features are recognised based on edges in YUV space, there is still some reliance on classifying colours. Because this is a common problem for all teams, many approaches have been taken, some are automatic, some are interactive and others are manual with less interactivity. This section seeks to give an overview of some of the other techniques used by other teams.

[1] The rUNSWift colour classification code (using a 2MB look-up table) takes approximately 2ms per frame.

Some of the methods used here could theoretically be able to build similar colour tables, given the correct input. However the work needed to actually create the function which represents the correct[2] classification can vary. This means that the main differentiating features of these methods are how they generalise, what methods are used for training and in some cases, what functions are possible to build.

2.1.1 Automated Colour Clustering

Members of the Dutch Aibo Team have researched an automated method for building a colour table, described in [3]. Briefly, the system uses images taken by the robot and attempts to find clusters of similar colours. This is done by transforming the image into a different colour space (Hue Saturation Intensity or HSI) then choosing a set of characteristic colours from the image and then using a dimension reduction algorithm to form groups of similar colours. These groups correspond to symbolic colours. The colour table can then be constructed by running each YUV triple through the dimensionality reduction algorithm to find the symbolic colour. By doing processing in HSI space, colours were separated reasonably well by just the hue component.

This method has the advantage of being automated – building the colour table is a matter of taking a training image and running the algorithm over it. However, there are drawbacks in the method – the main problem was that the results as described could only distinguish between 5 of the 'easier' colours (white, yellow, beacon-blue, green and pink). This method in its current form is therefore unusable for a game situation.

2.1.2 Dynamic Thresholds on Hue Channel

[5] describes a method which aims to be robust to lighting changes by dynamically updating the classification function. The colour is transformed into HSI space and the hue component is extracted. If the saturation or intensity of the colour falls outside a certain range (for colours like black or white), then the hue is set to specific hard-coded values. The hue component is then transformed by a function called τ, then this transformed hue is classified according to

[2] As discussed in 2.3, there are YUV triples which should belong to multiple symbolic colours. However there is still a "target" classification which is sought after when building the function.

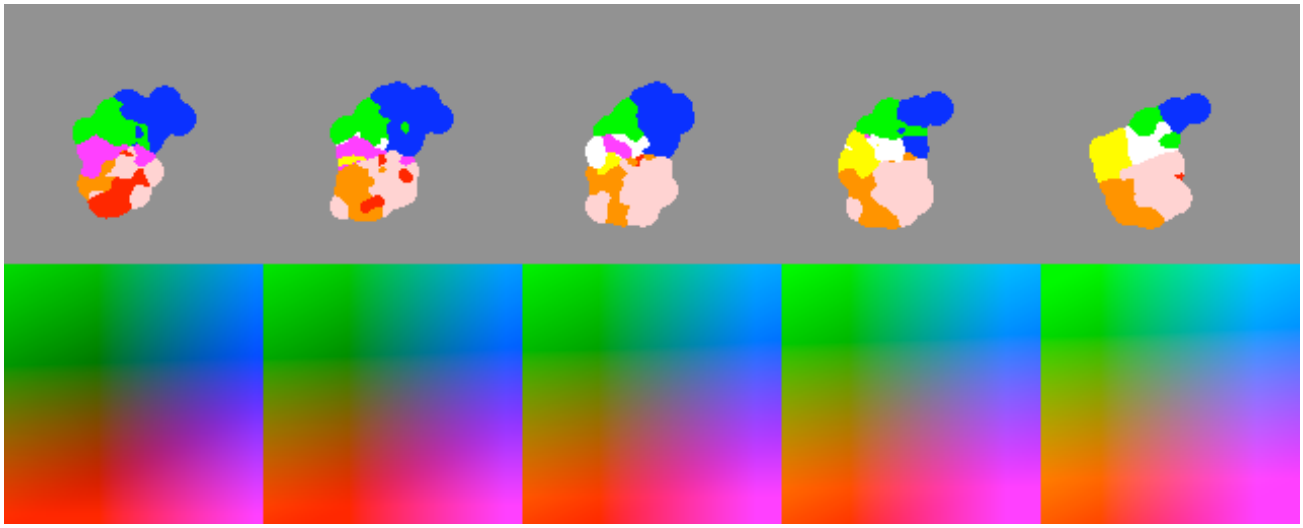

Figure 2.1: Top row: Cross sections of the classification function used in the rUNSWift lab with Y = 28 (left), 34, 40, 46, 52 (right). Note the disconnected areas classifying to the same symbolic colour.
Bottom row: Cross sections of the YUV colour space at the same Y values. These are included to give an idea about what parts of the colour space are being classified, they do not correspond to the colours seen by the robots – this would require exact specifications of lighting conditions and a printing process that allowed these colours to be represented correctly (these diagrams were converted to RGB space and then to CMYK space for printing).

fixed thresholds. The τ function is recomputed periodically (every 25 frames) to handle changes in lighting conditions.

There is a fairly high performance overhead of 100ms every time the τ function is recalculated (though in a game situation, this is never done when attempting time critical tasks like ball approach), though it might be possible to spread this calculation over multiple frames or just fix the τ function for a given lighting condition. Another weakness of this method is that although the classification function can change dynamically, it won't necessarily fit the data, due to its reliance on only one dimension of colour – for example, the colour table used by rUNSWift in the lab has disjointed regions in YUV space which classify as the same symbolic colour (see Figure 2.1). This is not possible to represent with the system described here.

The obvious benefit of this method is that calibration time is minimal, with only a few

parameters which need to be set manually and there is mention of using machine learning algorithms to automate the setting of these parameters also, creating a completely automated colour classification algorithm.

2.1.3 Median Filter Over Look-up Tables

The method used by the UChile1 team, briefly described in [15] is to create a look-up table mapping YUV triples to one of 8 symbolic colours, with 6-bits per colour dimension. The table is created manually, assigning colour classes to individual YUV triples. The number of samples used is said to be approximately 5000. This table is then median-filtered to generalise the function to handle colours outside of the sample space and to attempt to clean up the colour boundaries. Details of how the median is defined for the symbolic colour domain are not given, though it seems reasonable to assume that it is similar to finding the mode.

2.1.4 Exponential Generalisation of a Colour Table

A similar method has been used by the GermanTeam [8], though the generalisation process is different, and will be summarised here. Every sample given has an influence which covers the entire colour space. The influence is highest at the YUV coordinates of the sample, and decreases exponentially with Manhattan distance from the YUV coordinates. Formally, the influence I, of point p_2 being classified as i on point p_1 is given by:

$$I_i(p_1, p_2) = \begin{cases} \lambda^{|p_1 - p_2|} & i = c(p_2) \\ 0 & \forall i \neq c(p_2) \end{cases}$$

Where $c(p_2)$ is the colour class of p_2, and λ is some constant less than 1.

Each point in the colour table then has the total influence calculated based on all samples given, here each colour class is given some bias B_i which the influence is multiplied by to give a parameter to adjust how much the colour class will expand after generalisation. The symbolic colour with the highest influence on a particular point is the classification of that colour with the condition that the influence of this symbolic colour is significantly larger than the influence for other colours, with parameters to adjust confidence and background colour influence.

The process is not real-time, as it takes from 20 seconds up to 7 minutes on current hardware, depending on the size of the colour table and other factors. An advantage of the method is the extra parameters available for controlling how the table is generalised – without having to provide more training data, a table can be modified if a particular colour is over or under classified.

2.1.5 Ripple Down Rules

In the past, the rUNSWift team has used ripple down rules for colour classification [10]. The process builds the classification by adding and subtracting hyper-rectangles of symbolic colours inside the YUV space. The method was interactive, with the operator able to see the classification function updated as more samples are given to the system. The exact details of how the classification function was implemented are likely of little interest to the reader, so they will not be given here. The resulting classifier was then applied to the entire YUV space at 7-bits per channel of precision to build the colour table before being loaded onto a robot.

We moved away from using ripple down rules because of the biases in the classifier [9]. Since the hyper-rectangles spanning the colour classes were always axis-aligned, the classification function would require many levels of refinement to represent most boundaries. These extra levels of refinement would slow down the process of classifying, reducing the benefit of a real-time training program. The great advantage of using ripple down rules was that arbitrary functions could be represented.

2.2 Parzen Window Classifier

A Parzen Window is a simple method for estimating complex probability density functions. It approximates a distribution by taking a sum of multiple kernel functions:

$$p(\bar{x}) = \frac{1}{N} \sum_{i=1}^{N} W(\bar{x} - \bar{x_i})$$

In our case, the kernel function W is a Gaussian in YUV space:

$$W(\bar{x}) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-(\bar{x}\cdot\bar{x}/2\sigma^2)}$$

To use a Parzen Window as a classifier we work out probability density functions for each symbolic colour. For any YUV triple, we can check to see which symbolic colour has the greatest likelihood and provided that this is above a minimum threshold, it is declared as the classified colour.

The rUNSWift teams in 2005 and 2006 have used Parzen Window Classifiers, however to make them more suited to this specific task, some modifications were made. These modifications were made to make the algorithm easier to implement, faster to run and faster to learn in this domain. A description of each modification follows.

- It is easier to interactively classify colours if the probability density function for each channel is not a true probability density function, but an arbitrary function:

$$f_{colour} : (Y \times U \times V) \rightarrow weightf(\bar{x}) = \sum_{i=1}^{N} W(\bar{x} - \bar{x_i})$$

  This means that as kernels are added to the function, the function does not need to be re-normalised, so the effect is local in YUV space. This is important for interactive classification as it means that the border between neighbouring colours will be changing only in the area near where samples are being given, leaving other borders unchanged. This is demonstrated by example using Figures 2.2–2.4.

- The coefficient for the Gaussian ($\frac{1}{\sigma\sqrt{2\pi}}$) was dropped, since we aren't trying to maintain a normalised distribution. This makes implementation simpler and faster.

- The kernel function used can be modified slightly to have tighter variances in some dimensions, or tighter variances in all dimensions when $\sum_{c \in C} f_c(x)$ is high. We typically have a tighter variance in the Y dimension as this channel is less noisy than others (see Figure 2.5 and Table 2.1) and requires fine details in the final classification function. Setting the variances correctly simply speeds up the calibration process, if variances are too large, then more negative samples are needed, if variances are too small then more positive samples are needed.

- Similarly, the Y dimension can be further tightened by changing the distance calculation in the exponent so the change in the Y dimension is raised to the 4th power, rather than

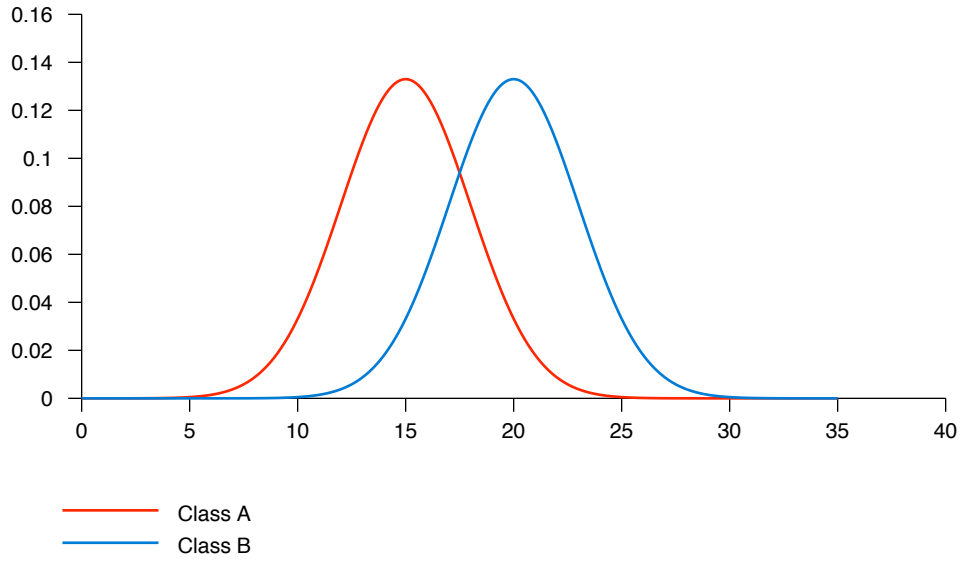

Figure 2.2: A Parzen window classifier in 1 dimension classifying between Class A and Class B, just before adding another sample for Class A at $x = 3$. The classification function currently represented is:
$x < 17.5 \rightarrow Class = A$, otherwise $Class = B$

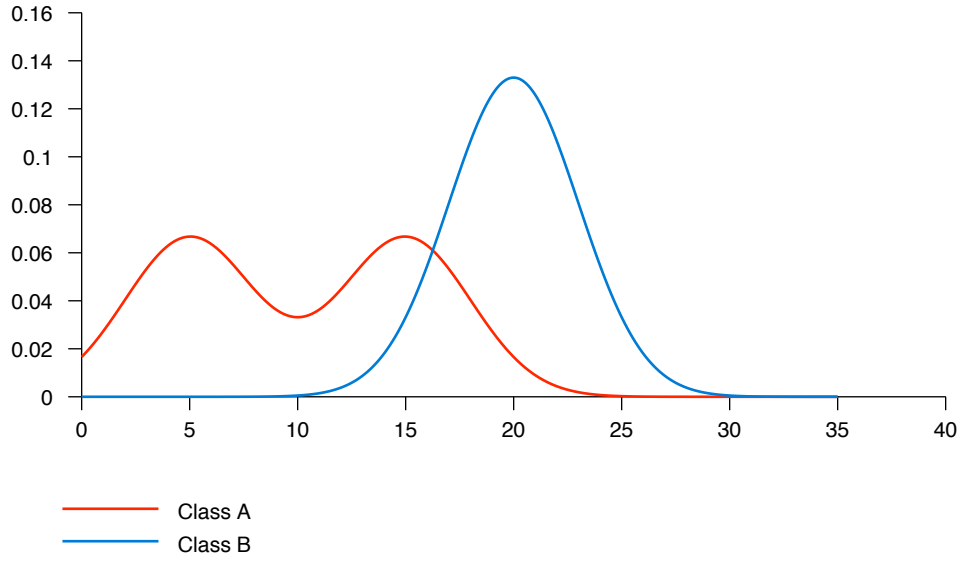

Figure 2.3: The Parzen window classifier representation from Figure 2.2 after adding a sample for Class A and re-normalising the distribution. Note that adding this sample has affected the border between the two classes, the classifier is now given by:
$x < 16.25 \rightarrow Class = A$, otherwise $Class = B$

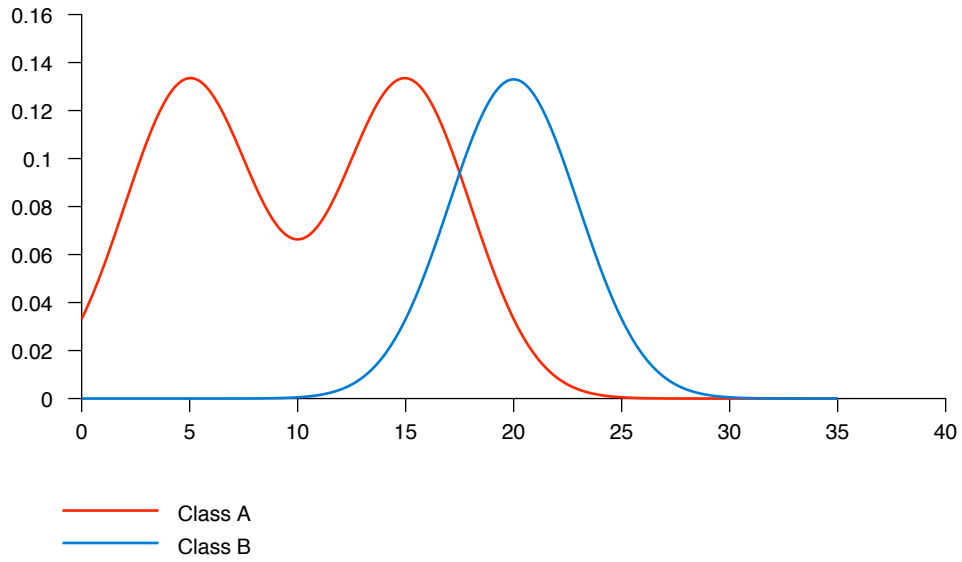

Figure 2.4: The Parzen window classifier representation from Figure 2.2 after adding a sample for Class A but *without* re-normalising. The boundary between the two classes has remained in the same location from Figure 2.2.

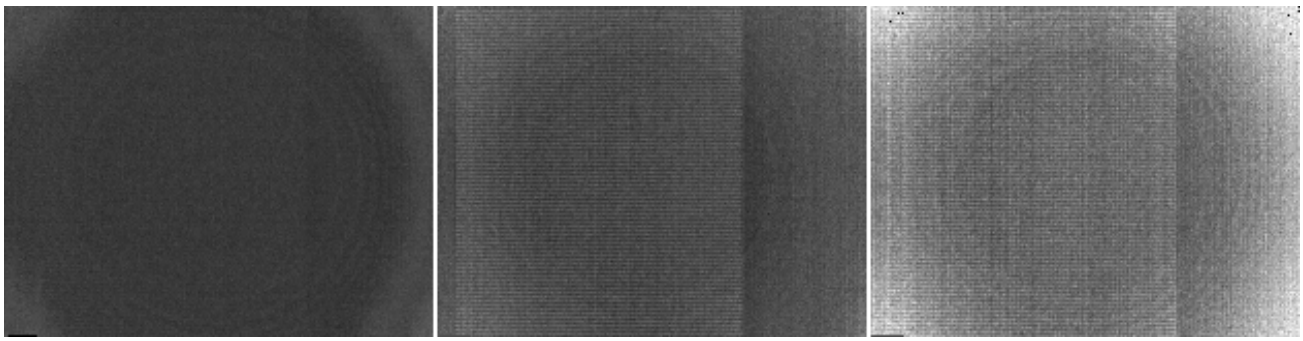

Figure 2.5: Standard deviation for the Y, U and V channels respectively. Lighter colours represent a higher standard deviation. Diagram was produced using 80 frames of a log taken with the robot in a dark cupboard and approximating the standard deviation for each pixel.

	Standard Deviation		
	Y channel	U channel	V channel
Median	1.12	1.56	2.51
Mean	1.15	1.57	2.71
Min	0.79	0.76	1.17
Max	1.99	3.40	5.00

Table 2.1: Some properties of the camera found in an ERS-7 robot. Based on 80 frames of a log taken with the robot in a dark cupboard. Each statistic was calculated discarding the 16 pixels in the bottom left corner which represent non-image data.

the second, leading to W being defined as:

$$W(\bar{x}) = exp(\frac{\bar{x}_y^4 + \bar{x}_u^2 + \bar{x}_v^2}{-2\sigma^2})$$

This form of the function was able to be toggled in the interface.

- Although a Gaussian distribution is non-zero everywhere, it is near zero in most places. To speed up computation, we clip the function outside a certain range of the mode. The range where the Gaussian is non-zero is given by $D(\bar{x}, \sigma) \leq 1$ where:

$$D(\bar{x}, \sigma) = \left(\frac{\bar{x}_y}{\sigma_y}\right)^p + \left(\frac{\bar{x}_u}{\sigma_u}\right)^2 + \left(\frac{\bar{x}_v}{\sigma_v}\right)^2$$

$p = 2$ or 4, depending on which variation on the function is being used.

With the modifications as described above, the classification only bears a passing resemblance to the original Parzen Window Classifier as described. The modifications made were made either with the purpose of simplifying the algorithm, or allowing the classifier to be trained faster. Any of these modifications can be left out with the algorithm still working, just the time spent classifying might be higher. Even though the classifier is so heavily modified, we will still refer to it as a Parzen Window Classifier for simplicity. It should be noted that some literature refers to this as a kernel classifier because of the kernel function which the distributions are built up from, we have avoided this term because this term is also used to refer to support vector machines.

2.2.1 rUNSWift 2005

The 2005 rUNSWift team used ripple down rules in the lab, but for the competition in Osaka, they switched to a Parzen Window Classifier. This was done for performance reasons as the learning algorithm would slow down significantly once the rules were sufficiently complicated. Because a Parzen Window Classifier applies the same constant time algorithm for each sample, the performance remains consistently fast, even for a complicated classification function.

2.2.2 rUNSWift 2006

The 2006 rUNSWift team continued to use Parzen Window Classifiers, but modified the tools used somewhat. The previous method of building a colour table was:

1. Boot a dog and log its camera images over wireless.
2. Strip out the images from the log file.
3. Load those images into the classification program and classify the image.
4. Save the final colour table and test it with the offline vision tool.

Since it was known that we would be playing on multiple fields in Bremen, we wanted to simplify the process to speed it up. This was done by putting the functionality of the classification tool into the offline vision tool. This improved the process of building a colour table considerably through the addition of several new features, and general simplification.

Live Classification

Since the offline vision tool is able to run the full vision pipeline, it is possible to view the end result of processing an image using the colour table you are currently editing, every time the colour table changes, the image is reprocessed with the new classification. This feature made it much easier to work out what is causing vision errors, and also to know what sort of tolerances the vision system has to poorly classified colours.

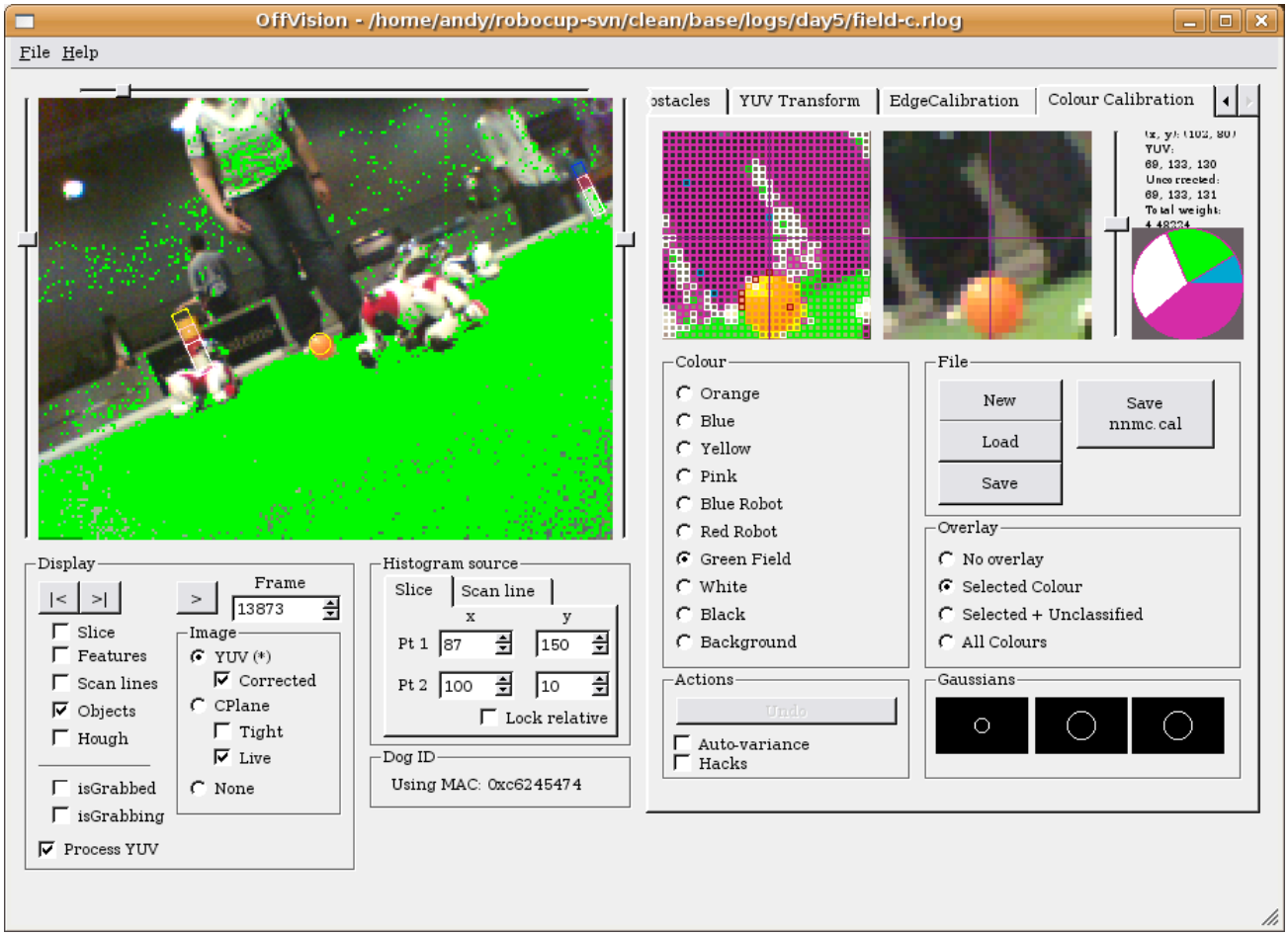

Figure 2.6: The new colour classification tool, built in to the existing offline vision simulator.

Function Visualisation

When classifying colours between colour boundaries, extreme care must be taken. Often you will get two colours with the exact same YUV triple, but belonging to objects that should have different symbolic colours. By just looking at the image, it is difficult to tell which colours are going to be on these borders.

To aid in this, a small pie chart was added to the tool which displayed the relative weights of each symbolic colour for the YUV triple under the cursor. When classifying colours, this visualisation made it easier to know what adding an extra kernel to the function would do.

The background colour for this pie chart was the colour of the pixel below the mouse cursor. This allowed the operator to see the actual colour they were classifying without having the adjacent colours in the image distract them.

Online Calibration

It was possible to connect directly to a dog and build a colour table directly from the live streaming image. The vision tool was given a buffer to enable a live stream to be paused, rewound and played back without needing to save the log to a file.[3] This was a only a side effect of the merging of the offline vision tool and classification tool. Although this feature wasn't useful for generating competition quality colour tables, it enabled very fast calibrations to be made for non-competition situations (such as the 'walk-learning runway' in the rUNSWift lab which doesn't need excellent vision calibration, but does have different lighting to the field). When a second person is available to help with calibration, then one person can move the robot while the other calibrates, significantly reducing the time taken to build the colour table.

Kernel Modification

From the modifications to the Parzen Window Function described previously, some required changes to the user interface. Modification of the variance for individual samples was supported, but in order to make the function usable it was automated somewhat. A checkbox allowed

[3]Due to the design of the rUNSWift logging system, it is actually simpler to save the log to the file while viewing it in the vision tool, and having saved copies of logs is usually quite useful.

toggling automatic variances on the Gaussians being used, the variance for each dimension would be inversely proportional to the total weight for each symbolic colour at the particular point in YUV space. That is:

$$\sigma_d = \frac{\Upsilon_d}{\sum_{k \in C} W(\bar{x})}$$

Where C is the set of symbolic colours, $d \in \{y, u, v\}$ and Υ_d are constants.

This meant that for areas which had been classified before were given new samples, the samples had a tighter variance. The weight given to these samples was inversely proportional to the variance, so as the kernels became more specialised, they had increased weighting.

Calibration Comparisons

The improved vision tool also made it simpler to compare two different colour tables. Typically, we were interested in comparing the previous colour table with the new table being developed. The vision tool was able to select between using the colour table on the robot, the classified image present in a log file, or the colour table which is being worked on. This enabled very fast comparisons to be made between calibrations to see the errors present in each and whether the new table was actually an improvement.

2.2.3 Calibration Strategy

The final result of calibration is a function mapping a YUV triple to a classified colour. That is, a function from 3 continuous[4] dimensions to 1 discrete dimension. However, when building the colour table, also important are the individual weight-space functions for each classified colour since they are what we modify directly. These form a function from 3 continuous dimensions to several more continuous dimensions. Because of the high dimensionality of the functions being built, it is typically impossible to visualise the whole function while working on it. Instead of trying to visualise the whole function, we look mainly at the results of using the current function – so we will show the image after classification, and which objects were recognised.

Understanding how the higher-level vision responded to classification noise is essential. A

[4]Although our representation of a colour is discretised to 128^3 possible, this is only a limitation imposed by computers and makes no difference to the algorithm.

single pixel misclassified on its own will rarely make any difference to the final result of image processing, if a difference is made it is typically just a slight variation on the distance or heading estimate (which will be filtered before being used by behaviours anyway, reducing the impact of these variations). So, although noise is unavoidable, we can shape our colour table so the noise will be spread out in the image, rather than having sections in the image where many pixels are misclassified we aim to have misclassified pixels scattered around the image.

The technique used to try to achieve this starting from an empty colour table (i.e. no YUV triples are classified) was as follows:

- Classification was always done a single YUV triple at a time. Although there was support for variable brush-sizes (thus classifying a pixel and all its neighbours), this was never used for building match-playing colour tables. This prevents pixels near borders in the colour table from being classified unintentionally.

- A rough, conservative colour table is built first, with only 2 or 3 samples per symbolic colour used – at this stage, there will be a lot of unclassified colours in any image. This is done first so as to give obvious warning if a colour is over-classified – if a conservatively classified pixel changes its classification, then this is a clear sign that the last change to the colour table was bad.

- After the conservative colour table is built, individual colours are worked on. This means getting the border between a particular colour and all the neighbouring symbolic colours correct, so although it might be the green colour which is actively being worked on, pixels might be classified as background or blue to make sure green is not over-classified.

- When a large area of pixels is unclassified, attempt to classify the pixel which will classify as many of the other pixels in the image as possible. Actually doing this is a process of trial and error (the classification program has unlimited levels of undo), but it has the effect of expanding regions of colour in the colour table slowly, over-classifying as little as possible.

The result of building a colour table this way was that the borders of symbolic colour regions were close to correct. This means that rather than having an area in YUV space which tends

to be misclassified as C_1 and another area which tends to be misclassified as C_2, we have the whole border between C_1 and C_2 which tends to be misclassified in either direction. This has the consequence of spreading out the noise in the classified image, helping the high-level vision to filter it out.

This method is probably quite similar to the method described in [2] where only colours that were known for certain were given a classification with ambiguous colours left up to the learning algorithm. Although here due to the extra information given by the real-time processing, we can classify some of the less ambiguous colours with immediate feedback on what this does to the vision system as a whole, allowing the change to be undone if necessary.

2.3 Problem Colours

There are certain colours which are frequently confused (such as the colour of a shadowed ball and a red robot). Handling the misclassification of these colours is handled in several ways. Some of these techniques are quite domain specific, and will depend on priorities (for example, the rUNSWift team seeks to avoid false positives on the ball, before it seeks to see a shadowed ball) however other ways of minimising the problem are more general, using higher-level vision code to eliminate spurious features.

This section aims to give a detailed description of how the rUNSWift vision system was made to be robust in the areas typically found difficult. Some of the methods used were specific to a particular environment – although the vision system will perform adequately in different environments,[5] spending time to tune the vision system for a specific environment leads to much more reliable performance.

[5]On arriving in Bremen, a dog running code calibrated for our lab could grab the ball and score goals – it was by no means perfect, but it could have played a game.

2.3.1 Environments

The rUNSWift Lab

The rUNSWift lab has a full field, with dark green carpet. The field border is a small dark grey barrier and beyond that is a variety of objects such as computers, cables and filing cabinets. All of these sit on a dark grey carpet. Lighting is provided by 4 halogen lights and many general purpose fluorescent lights. The halogen lights are in line with the side-lines of the field and slightly closer to the centre of the field than the beacons. Sharp shadows are only cast by the halogens, but because of their positioning, a referee standing outside the field does not cast much of a shadow. Beacons are more strongly lit on the side closest to the field, with most of the side light coming from the fluorescent lights.

Bremen

While in Bremen, vision was calibrated for 3 different fields. Conditions were fairly similar for all three fields. Lighting was mostly provided by halogen lights - there were some fluorescent lights in the ceiling, but they were very weak. The halogens were mounted higher up than in the rUNSWift lab, and they were also located outside the rectangle of the field – this meant that a referee standing outside the field would cast a sharp shadow onto the field (often onto a beacon). The outside of the field was marked by a light grey carpet.

Although the 3 fields in Bremen were mostly the same, there were some key differences between the non-centre-court fields (field A and C) and the centre-court field. Fields A and C were elevated while the centre court field was not. The rUNSWift vision system will discard a lot of visual information when it is above the horizon. An elevated field will put more things below the horizon if a dog gets into a situation where it is looking off the edge of the field. Fields A and C also had a clear view of some windows at one end of the field. From a dog's perspective, these windows looked a similar colour to a blue goal and were approximately half a goal-height above the expected position of a goal. Other difficult background objects included a video screen, and the other fields (see Figure 2.7 for an example).

Figure 2.7: An example of difficult objects in the background. A video screen could display anything, and the window is a very similar shade of blue to a goal of a beacon. Both of these examples appear above the horizon, but with only a small error margin.

2.3.2 Colours Used by rUNSWift

In the 4-legged league, there are about 11 distinct colours appearing in the game – the green carpet and the white field-lines, the yellow and blue on beacons and in goals, the pink in beacons, the white on a robot, the white on the side of a goal and under the beacon, the robot uniform colours, the orange in the ball and the black in the referees socks and pants.[6] However, it is not necessary to distinguish exactly between all these colours (in some cases it is impossible to do this, just considering the YUV values), and some colours aren't as useful as others so they need not be classified. It is also useful to define a "background" colour to classify colours appearing off-field. Following is a list of the colours used by rUNSWift and their purposes in the higher level vision code:

Orange used to find some ball features, and also to verify that a detected ball is actually a ball. Under ideal conditions, approximately half of the ball features found are found using symbolic colours. In other circumstances (such as blurred balls or shadowed balls) this proportion can change in either direction.

[6]Here, we mention three different forms of white – one for beacons and goals, one for robots and one for field lines. Although the variation between these colours is small, it could be possible to classify them differently in some circumstances (for example, depending on the material used, the field lines may appear slightly green).

Pink is used to detect the presence of beacons, and also in the calculations to work out the distance from a beacon.

Beacon/Goal Blue and Yellow are used to work out which beacon was detected when a pink feature is found and also in calculating the distance to the beacon. The same colour is also used for detecting goals.

Robot Red we try to detect the colour of a red robot uniform to try and prevent seeing a red robot as a ball. The colour is not used to detect robots, only to cast doubt over the detection of a ball.

Green is used to detect when we are standing off the field, and to confirm other features (for example, a goal must have green below it to be considered 'sane').

White is used to confirm that a beacon is actually a beacon. The white in field lines is detected through YUV differences, not symbolic colour differences.

Background is a symbolic colour for the colours which commonly appear off field. This is separate to an unclassified colour which has too low a weight to be classified as anything, though it is treated similarly. While no code directly uses this colour, presence of background (or any other non-goal colour) in a goal will identify a gap to shoot at.

It is impossible to find a colour table for an environment which gives perfect classification. Shadowed balls will look red, blurred red uniforms will look orange or pink depending on the background, yellow goals will look white and noise in the image will often cross classification boundaries. This means that the higher level vision must be able to handle misclassified regions. The following section discusses how particular boundaries were handled.

2.3.3 Orange-Red-Pink Boundary

The orange-red-pink colour boundary is probably the most difficult boundary to get correct. With this boundary set poorly, there will be problems with seeing the ball in a red robot, not seeing grabbed balls, not seeing beacons or even seeing ball in beacon. The philosophy used by the rUNSWift 2006 team was to minimise false positives of the ball first, with any detection of

shadowed balls as a bonus. Similarly for beacons, we sought to minimise false positives first, and difficult-to-see beacons were a bonus.

Since this problem is common across all teams, we will give some background on other methods of handling this colour boundary before giving the method used by rUNSWift in 2006. There is likely to be some overlap between the different methods as some ideas can be combined together, and some can be viewed as generalisations of other methods.

Other Approaches

Soft Colours The NUbots' team report from 2005 [11] describes their approach to the problem. To handle the ambiguous colours, extra symbolic colours (called "soft colours" were added, shadow-blue, red-orange, yellow-orange, pink-orange, ball-orange and shadow-object. When detecting higher level features, rules are extended to combine blobs of these soft colours together to form objects.

Tight and Loose Colours Previous rUNSWift teams have used the concept of tight and loose classifications, where a YUV triple is given a symbolic colour as well as a boolean stating the confidence ("tight" or "loose") of the classification. Higher level vision then has rules to discard feature that do not have enough colour from a tight classification to agree with them. This method wasn't used by either the 2005 and 2006 rUNSWift teams as the complexity involved with using the extra information at a higher level was deemed too high for the small improvement gained.

Primary and Secondary Colours Similar to the soft colour approach by the NUbots, we experimented with more complicated colour classification – allowing each YUV to be classified as either:

- A single symbolic colour.
- A symbolic colour, and a possible other symbolic colour.
- A symbolic colour, and a flag stating that there is high uncertainty about this colour.

- A special case near one of the troublesome colour boundaries. (e.g. "orange, near red and near pink")

The distinction between the three classes was made automatically by looking at the relative weights of the symbolic colours for a YUV triple. Any symbolic colours with a weight greater than 80% of the weight of the classified colour were considered to be 'possible' colours. The encoding of as many of these colours into a byte was done with priority given to colours of a higher weight. (With less than $2^4 = 16$ colours, we could fit two colours into a byte, with room left over to encode special cases)

This method was never fully explored due to time constraints. Actually finding ways to use the extra information was the challenging part. Without the extra information, the vision system was adequate, though it would often miss shadowed balls. A more relaxed system for detecting balls would have fixed this, however the risk of seeing red robots as balls was deemed too great.

Our Solution

While other solutions to the problem involve creating more symbolic colours to handle the ambiguous cases, the rUNSWift system has moved in the other direction, merging similar colours into a single group, with the disambiguation moved to a higher level. This way, the vision system is more robust to changes in lighting, and instead of time being spent on training the classifier and finding the optimal place to put a difficult colour boundary, the time can be spent improving other aspects of the code. Adding more colour classes just introduces more ambiguous boundaries at different points which will require experimentation to find the optimal classification. Although changes in global lighting would probably result in the game being stopped, local lighting variations are very common, with referees shadowing different parts of the field constantly, this means that a system that is robust to lighting variations will perform better in game situations.

We also make use of edge detection in YUV space, this method is also particularly stable with changes in lighting conditions because it relies on differences between YUV values, which are preserved under different lighting conditions. This work is described in great detail in [9].

The higher level code to distinguish between different objects is fairly simple. A set of recognised objects is passed to it, and it will apply sanity checks to rule out any objects which appear to be false positives. Extra checks can be added easily, and our tools will indicate which checks have been activated for a particular image to help fine tune the checks. The advantage of these checks is that it means that we can relax the constraints for seeing objects at the lower level, as long as checks can be added to identify the false positives this generates.

Beacon Sanity Checks

This year, an extra sanity check for beacons was added to try to cull the "freak" beacons, which appear in unpredictable spots (for example, the beacon appearing in the red robot in Figure 2.8).

This check would make sure there was enough of the expected colours in corresponding parts of the image. There are three areas we check, each of them 3×3 pixels. We check the centroid of the top section of colour (P_1), the centroid of the bottom centroid of colour (P_2), and then a third point ($P_3 = P_2 + 1.25(P_2 - P_1)$) just above the centroid of the white section (see Figure 2.8). In each of the areas around these points, we count the number of pixels classified correctly, according to the type of beacon we are expecting (P_3 is always expecting white). If any of these counts are too low, or the total of these counts is too low (indicating that no part of the beacon was particularly well coloured) then the beacon is rejected. A source listing for this check is available in Listing B.1.

Edge Detection

Another approach to handling the difficult areas is to take a conservative classification and then expand the region (in image space) until a hard edge is reached, classifying all pixels with the same symbolic conservative classification until the edge is reached. This approach is used to some extent in the rUNSWift code and has been investigated by others [7].

rUNSWift uses edge detection extensively in its ball detection algorithm – when ball features are found, they are typically moved to the nearest edge (looking at the image in YUV space) to allow for misclassification around the fringe of the ball. This technique works well unless the ball is blurry, when there is no sharp edge to detect.

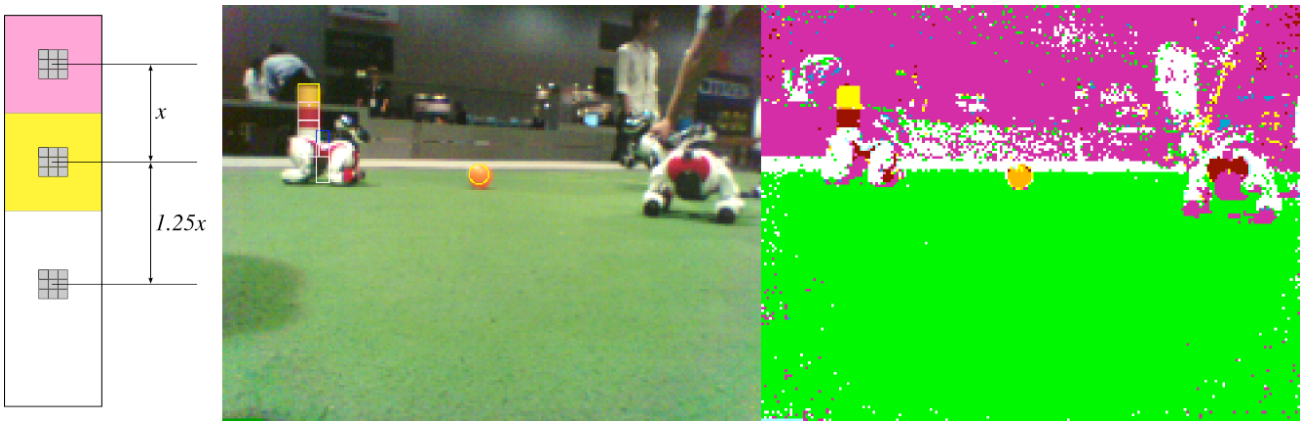

Figure 2.8: The regions checked for correct colours in a beacon. If enough pixels aren't classified correctly, then the beacon is rejected. For example, the false beacon on the right will be discarded by this check since there is not enough white at the bottom.

Merging Pink and Red

On arriving in Bremen, it was immediately obvious that the shade of pink used in the beacons was significantly different to the pink used in the rUNSWift lab. Initial attempts to classify the colour worked reasonably well, however beacons were misfit or not even detected, especially when either shadowed by a person, or looked at from one side as the pink on the beacons was often misclassified as uniform red. Since the rUNSWift vision code makes little use of the red uniform, it was decided to merge pink and red into a single symbolic colour and change the higher level vision code to prevent red robots from appearing as beacons.

The rUNSWift vision system detects horizontal runs of pink pixels ("pink features"), and then groups them together if they are aligned so they can be detected as a beacon. The entire code base was already dropping frames, so the code to reject red uniforms had to be fast. For this reason, it was decided that operating on the feature-level (as opposed to the pixel level) was the best – there were relatively few pink features for a given image (from 0 to a maximum of about 15) and they conveyed a good sense of the shape of the figure, which was the main way of distinguishing between a beacon and a robot.

Images of red robots and beacons from several angles were taken, and the common patterns of pink features were found. Examples of the pink features recognised by the vision are shown in Figure 2.9.

Figure 2.9: Examples of the shapes of pink features seen in red robots.

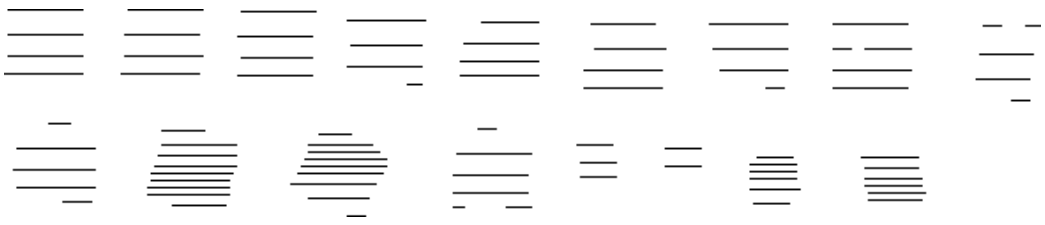

Figure 2.10: Examples of the shapes of pink features seen in beacons.

The distinguishing feature of a beacon is that the pink features form a rectangle. This rectangle could be skewed and occasionally had deformations at the ends (see Figure 2.10). Looking at the patterns from a red uniform, there are many cases, however none of them are as uniform as the beacon patterns.

From this, tests were written to reject non-beacon looking feature groups. Since the top-most and bottom-most features were often noisy (due to an incorrect horizon) tests needed to allow some noisy input. The tests to determine whether a set of pink features were part of a beacon were:

- Features must all be of a similar size. Specifically, the second smallest feature could not be

5 pixels shorter than the average feature length, nor could be the second largest feature be 5 pixels longer than the average feature length. By taking the second smallest and second largest features, some resistance to noise was added, however when there were only a few features (5 or less) this test was not used.

- The line connecting the centroids of the features was required to be relatively straight. Going through each pair of adjoining pink features, two values were found δ_l and δ_r. δ_l was the sum of differences between pink feature centroids where the bottom feature was to the left of the top feature, similarly δ_r was the sum for when the bottom feature was to the right. For a perfectly straight set of features, $\delta_l = \delta_r = 0$. However, typically the beacons had some sort of slant in one direction, so either δ_l or δ_r was large while the other was small.

  In the case of a red uniform though, the centroids would typically not sit on a line, vertical or otherwise. So, this test would look at the minimum of δ_l and δ_r and if it was greater than 3, then the features were rejected.

The code for this check is given in Listing B.2.

Although this check worked most of the time, another simple check could be added to further reduce the possibility of seeing a false beacon. Since the problem is likely to occur when a red robot stands in front of a goal, we discard any beacons which appear inside the bounding box of a goal (see Listing B.3 for implementation). This check is fairly straight-forward and will only fire when something is incorrect (though it is possible that it is the goal that is improperly detected, not the beacon).

2.3.4 Dark Green-Blue-Background Boundary

Especially in darker environments (like the rUNSWift lab), the distinction between green, blue and background can be difficult. Although "background" is not a fixed colour, it is typically the darker colours which appear off-field where lighting is not so strong. The shadowed blue present in a goal also leads to complications, as it is quite dark and often appears as green or background. This is unacceptable as it will cause shots on goal to disregard the corners of the goals. Confusing background with green will lead to robots leaving the field without realising

and confusing green with blue will lead to bad localisation, and even spurious shots on the blue goal.

The only important colour at this boundary is blue. While green is useful for determining if we are off the field, this behaviour is based on reaching some threshold of green in the image – this threshold can be lowered or raised if necessary so as long as the "easy" green colours are classified, then there won't be an issue. Similarly, the background colour isn't used for game-play decisions, except as the absence of some other colour.

This problem is solved quite simply by knowing where to allow misclassified pixels. In this case, letting a couple of pixels in shadowed blue goal corners is enough to remove false positives in the background and the field. We never allowed enough noise in the corner that caused the actual goal detection to change (or else we would shoot on goal poorly), but by having more darker pixels classified as background significantly reduces the chances of seeing false positives in the background.

2.3.5 Light Blue-Background Boundary

Dark blue was not an issue at the Bremen fields. Because the lighting was so much stronger than in the rUNSWift labs, it was fairly easy to build a colour table which distinguished between those problem colours. However, in Bremen, there was concern over some windows visible from on the field which appeared as a light blue shade from on field. The windows were at approximately the right height to be considered as goals based on their position relative to the calculated horizon. The other problem was that it was possible to see other fields from certain positions on the field. Although it is unlikely that the objects on the other field would be recognised (due to their distance and position), it would be possible that a beacon may be mis-recognised because it was seen in front of a blue goal, for example there is a blue goal from a different field directly behind the pink on blue beacon in Figure 2.11.

Again this problem was solved by allowing more misclassified pixels to exist in the goal. In this case, even unshadowed goals had some pixels misclassified as background. The background still had blue pixels in it, but not enough to form features. The audience also played a part in solving the problem. By surrounding the field, they blocked most views to the windows and other fields.

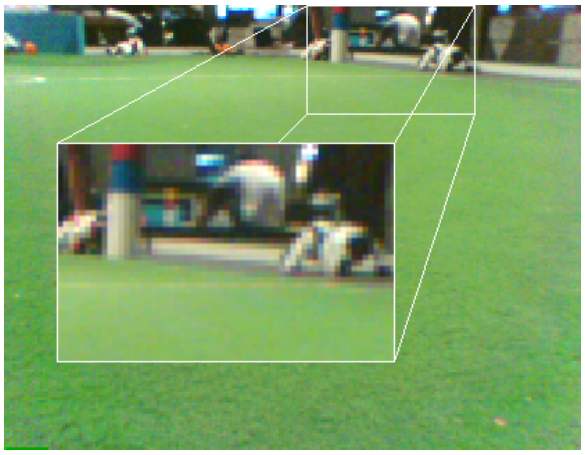

Figure 2.11: The blue goal from a different field visible right near the pink on blue beacon.

2.4 Evaluation

Evaluating the vision system is difficult. Looking through log files, we can count how many objects are recognised and how many are missed to give some idea, but we also need to measure the accuracy of the distance and heading estimates. To measure these in a static scene is easy, but it is also a much easier task. In a game, the dogs are rarely standing still, so blurred images are common, and the ball and head are moving too, leading to elliptical balls and other strange images. However, being able to measure how well the distance estimates are in a game is nearly impossible without extra sensors such as overhead cameras.

Another benchmark which might be useful to look at is calibration time. One of the goals with the rUNSWift vision this year was to make building the colour table a much quicker process. However, measuring the time taken to produce a good colour table is not easy – there needs to be some way to know when the colour table is "finished", and for both methods of calibration, this needs to be when an equally good colour table has been built. But if there was some simple metric for the fitness of a colour table, then rather than manually calibrating a dog, we would use a machine learning algorithm to do the work for us. For this reason, rather than measuring the total time taken to create a colour table, we give the time taken to do the smaller actions involved.

So, with these difficulties in mind, this section aims to give some evaluation of the vision system used by the 2006 rUNSWift team.

Task	Old Tools (s)	New Tools (s)
New Weights	51	17
Modify Weights	87	32
Test Weights (Offline)	111	33
Create Table	10	< 1

Table 2.2: Time taken to perform various tasks involved in building a colour table.

2.4.1 Calibration Time

The following activities were timed using both the new vision tools and the older 2005 tools:

New Weights Starting from a log file recorded on the dog, how long it takes to get to a point where colours can be classified.[7]

Modify Weights Starting from a log file recorded on the dog, and a file containing the weights for each symbolic colour, how long it takes to get to a point where more colours in the existing colour table can be classified.

Test Weights (Offline) With a set of weights and a log file, how long it takes to look at the final result of the vision system using the weights given.

Create Table With the weights loaded in memory, how long does it take to produce a colour table which can be loaded onto a dog.

The results are shown in Table 2.2 and should not be surprising, considering the new tools were created with the primary intention of speeding up the calibration process.

The speed gains are a result of:

1. A single program to manage all of vision. This leads to less time wasted switching between programs.

[7] Although the new vision tools could begin calibrating even earlier, while the log file was being recorded, this would typically require two people – one to move the dog and one to classify colours. Since we never seriously calibrated this way, we don't consider this way of calibrating in this evaluation.

2. The tool is compatible with all the file formats used in the vision system, so no extra conversions are needed.[8]

3. The new vision tool is written in C++. The old colour classification tool was a Java program, which added a few seconds to any operation.

4. The newer tool has more efficient memory usage. Instead of loading all images into memory, it only loads images as they are needed, caching recently viewed images[9]. The final colour table is always kept in memory, and updated as it changes. This makes creating the colour table for the dog very fast, as it is just copying memory to a file.

A more realistic evaluation was done recently – a field was set up in an entirely new location, and the colour table was calibrated in in approximately 10 minutes[10]. This time would increase if only one person was available (forcing one person to both train the classifier and take logs). The colour table was created from scratch, and required approximately 100 samples to produce. However, the task was made significantly easier by not using red robots in any of the games.

2.4.2 Vision Quality

The approach to evaluating the quality of the vision system was to take a large log file, and for each image depicting a game situation, compare the results of multiple systems, including "perfect" recognition done by a human. The test was designed to be as objective as possible, however due to the lack of availability of log files, the log file tested was not from a game situation. However, the log file used was large, and contained many samples of each relevant object (a minimum of 230) with different scenarios being tested.[11]

[8] There are at least 4 different bitmap file formats used in the rUNSWift codebase.

[9] A much larger cache is used when we connect directly to a robot, allowing rewinding of a live stream.

[10] This was done at the University Open Day to demonstrate to school children that colour classification can be done quickly and safely at UNSW.

[11] Although the log only includes game legal situations, the sampling of these images will probably not reflect a sampling of images taken from a game.

Log File Choice

The choices of log files are limited by resources at the time of writing due to a lack of robots. The only logs from a game-like situation were taken in the rUNSWift lab in a 3-vs-3 game with all robots wearing red uniforms. There are 341MB of logs in this collection taken from each dog playing at the time. All logs taken in Bremen are available, although none are from game-like situations, they were taken specifically for calibrating and testing vision, so they contain many different kinds of images. In total, there are 1.5GB of logs taken from Bremen available. There are also two robots in the rUNSWift lab which are available but batteries are still in shipping. In any case, this is not enough to give a good simulation of a game.

Since the main focus of the vision system was the fields in Bremen, we will primarily use the logs from Bremen for our analysis, though the logs taken from the rUNSWift lab will be briefly looked at to demonstrate weaknesses in the approach taken in Bremen.

Systems Tested

We are primarily seeking to evaluate the vision system used in Bremen. However, in order to do this, we need to make comparisons with other systems. All of these systems (bar the reference system) are very similar, and are just variations on the same codebase. So, we will refer to the following systems:

Reference This is a "perfect" vision system done by a human. It makes no mistakes, and will recognise every relevant object as long as sufficient information is present (for example, if only the bottom section of a beacon is visible, then it is only identified if it is yellow or blue since there are two beacons with pink bottoms). This system is supposed to be a theoretical maximum, so any machine vision system can approach but not surpass it. While it is possible for human error to be present, extra consistency checking and verification by re-classifying certain frames[12] did not pick up any errors.

Separate This is the vision system as used in the rUNSWift labs with a colour table created for the purposes of this evaluation. This colour table was derived from two logs taken on field

[12] Any frames giving false positives by a vision system were thoroughly checked, as were any differences between the two merged vision systems, and a selection of random frames were also re-checked.

C (neither of which was the log used for the evaluation). This colour table was created after returning from Bremen and will never be used on an actual robot but it gives some indication of how the rUNSWift vision system would have performed without merging red and pink.

Merged without checks This system is the result of merging pink and red without applying the extra checks added to prevent robots from being seen as beacons. It was never used in games. The purpose of this vision system being evaluated is to test the added checks to ensure they aren't so strict they discard valid beacons or that they are so relaxed that they do nothing.

Merged The vision system used in Bremen for the preliminary games. Pink and red are merged together and the extra checks described in section 2.3.3 are used.

Testing Method

The process for then comparing the vision systems was as follows:

1. A log file is viewed, marking all frames which are possible in a game situation.
2. For each marked frame, we then noted which objects were visible and could be identified.
3. For each marked frame, we test the other vision methods and record the objects visible, as well as whether the ball (if present) was misfit and if either goal (if present) had a correct gap found.
4. After the logs have been processed, we then made sure the results were correct by looking at all false positives (since they are very infrequent in the correct result, but typically an error will show up as a false positive) and any other anomalies (for example, since the Merged and Merged Without Checks systems are very similar, every frame where they disagreed was checked a second time).

The variables we were recording for each frame were:

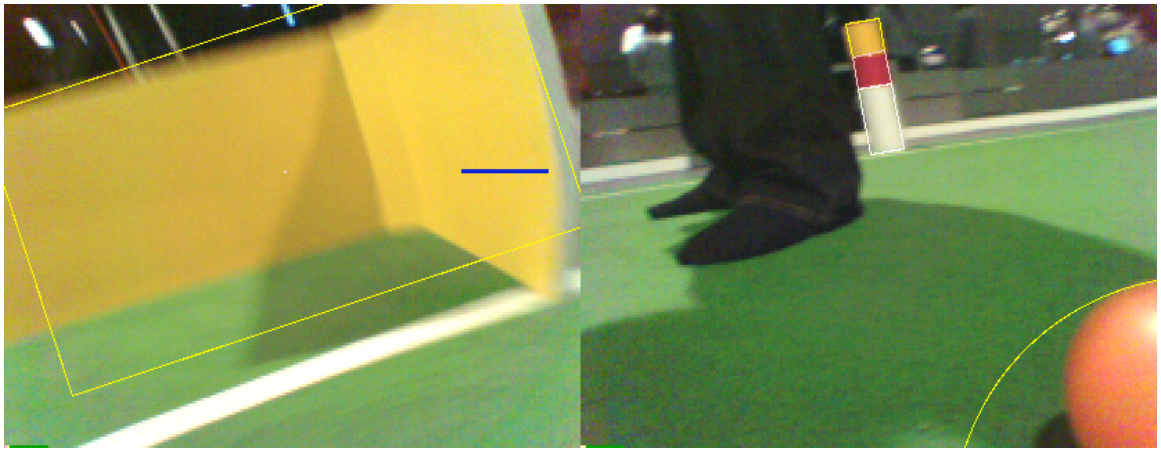

Figure 2.12: Left: An example of a bad gap (blue line).Right: An example of a misfit ball.

Balls, Beacons and Goals All of these visual objects can either be recognised or not. If they are not recognised when they actually exist, then a false negative is declared. When an object is recognised when it was not present in the frame, then this was a false positive.

Misfit A ball is misfit if 50% of the area inside the recognised ball is not the ball, or 50% of the area outside the recognised ball is the ball. For a non-circular ball (which is common when the head is moving) the condition was relaxed somewhat as it is unclear how to fit a circle to such a shape. In these cases, if the circle fit a possible extrapolation of the blurred ball according to the criteria above, then it was deemed correct. See Figure 2.12 for an example.

Incorrect Gap Whenever we see a goal, we try to find the best gap in the goal, if we were to have a shot on goal this frame. This was deemed as incorrect if the gap passes through the goalie or doesn't reach within a ball-width and a half of the extent of the gap. We also treated frames as incorrect if the gap had a discontinuity in it where there was no obstacle. In most cases, the frames did not capture the dog in a potential goal scoring position, so this metric is of limited value. See Figure 2.12 for an example.

There were no cases in the testing data where an object was seen in the wrong spot on an image (thus making a situation where the reference system and test system agree, but have the object in different locations). However, had this situation appeared, then we would have made note of it. There was a situation where a goal was misfit, which is shown in Figure 2.13 however

Figure 2.13: The misfit goal from the testing data in frame 71497. All three systems being tested misfit the goal in this way, however here the beacon is rejected due to the extra check added for the Merged system (in the other vision systems, this beacon was recognised).

for localisation purposes, the goal is not a great landmark so it does not carry much weight[13] and the gap found in this case was correct.

Training Data

Two colour tables are used in this evaluation – one which distinguishes between red and pink, and one which doesn't. The table which doesn't distinguish between the two colours was built using all the training data available, though later logs were taken only to refine certain areas. The table which does distinguish between the two colours was built by looking at two log files (36MB and 18MB), both taken for the purpose of building a full colour table. The contents of these two log files include all the elements required to create a good colour table (such as close ups of robots, blurred balls, shadowed objects, grabbed balls and people on the field).

Training Time

The colour table used in the Separate vision system was created after returning from Bremen using the training data described above in section 2.4.2. To try and make the comparison fair,

[13]The distance estimate is typically poor, and the heading information is of limited value, since part of the goal can be seen facing most directions on the field, especially by the goalie when standing on one side of the goal.

Weight-space	Total Weight	Approximate # Samples
Single Gaussian of regular size	296	1
Bremen Field C: Competition[a]	667452	2255
Bremen Centre Court	953960	3223
Bremen Field C: Separate Pink and Red[b]	1206192	4075
rUNSWift lab	3715609	12552

Table 2.3: Total weights for different classification files. These give a rough approximation as to how many colour samples were classified for each case by dividing the total weight by the weight of a single Gaussian (296).

[a]Used in the Merged and Merged Without Checks vision systems

[b]Used in the Separate vision system

we have attempted to keep the time taken in building the table similar to the time taken building the table used by the other systems.

Since we never timed ourselves when creating the colour table, we could only try to roughly approximate these times by looking at the classification function itself. However we can approximate the time taken to produce the classification because each sample given adds a Gaussian of a roughly constant magnitude. We can calculate the total value in weight-space of all the Gaussians in a particular classifier. So the total weight is roughly proportional to the time taken in producing the classification. This method is only a rough approximation, and especially in the case of the classifier which separated pink and red, there would be images where a lot of fine tuning was done around that colour boundary, this fine tuning increases the weights very quickly for the amount of time it takes to perform.

To give an estimate of what the total weight actually means, we also give the total weight of a single Gaussian, created by making a new classifier which was given only a single sample. This sample was somewhere near the middle of YUV space to prevent the edges of the Gaussian from being clipped. However, it should be noted that we use Gaussians of varying sizes and weights, so these are only rough figures.

For each particular classification, we can estimate how time was spent in terms of classifying particular colours by showing proportions of each weight per symbolic colour. These proportions

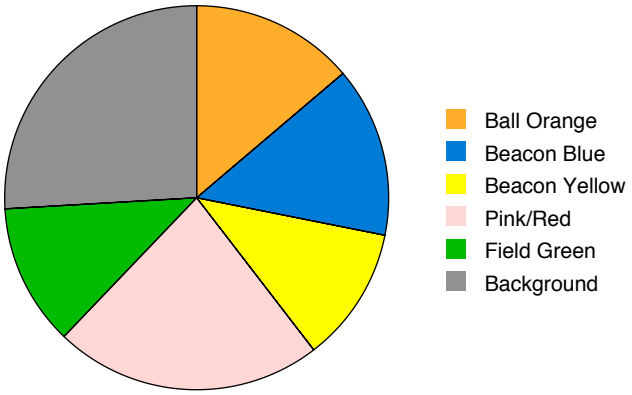

Figure 2.14: Proportion of symbolic colour weights in classification for Bremen Field C, with red and pink *merged*.

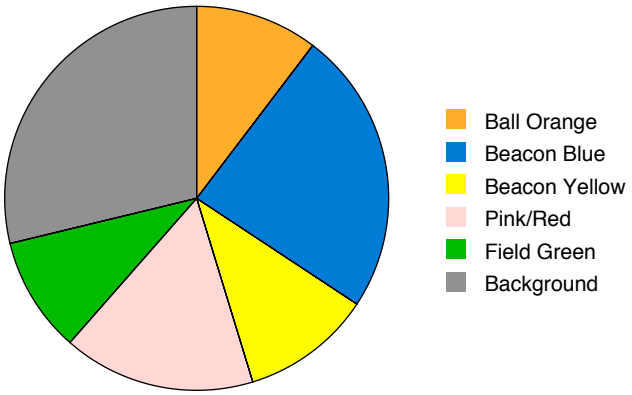

Figure 2.15: Proportion of symbolic colour weights in classification for Bremen Centre Court, with red and pink *merged*.

are shown in Figure 2.14 through to Figure 2.17. A larger area indicates a larger number of samples given while training (thus it correlates with harder to classify colours).

Note the great proportion of weight needed in Figure 2.16 to try to create the border between pink and red. Similarly, the weightings of pink, red and orange in Figure 2.17 are much higher than in the well-lit environments in Bremen. So these figures give a good approximation to the time taken classifying each particular colour.

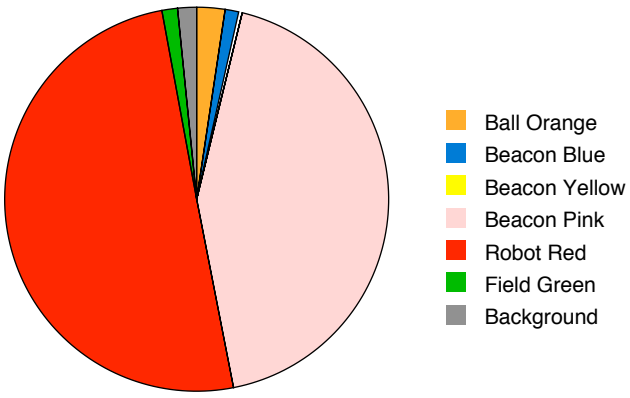

Figure 2.16: Proportion of symbolic colour weights in classification for Bremen Field C, with red and pink *separate*.

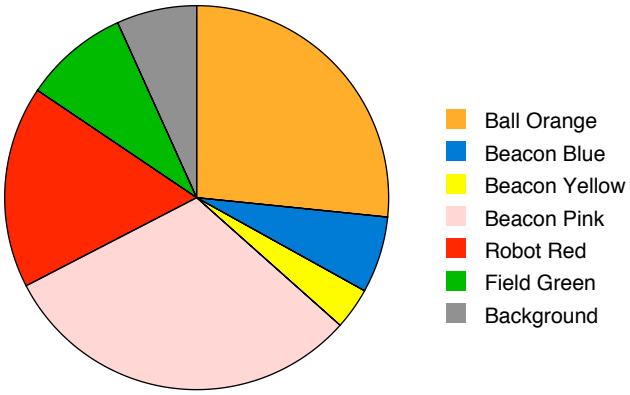

Figure 2.17: Proportion of symbolic colour weights in classification for the rUNSWift lab, with red and pink *separate*.

Testing Data

In total, there were 2298 frames used in our analysis. Each of these was classified by hand to provide a benchmark. Since the log was taken later in the competition, its influence on the colour table is much smaller than the earlier logs – this helps somewhat to avoid testing on the training data, though all logs from Bremen were used to improve the vision system, so the approach is not perfect.

The actual contents of the log are varied. Things which are obviously present in the log include:

- Tracking the ball as other robots attempt the passing challenge, with beacons in the background. This is done from many angles and distances.
- Looking off the edge of the field from all sides.
- Looking at each goal from different angles, with and without a person shadowing some or all of the goal.
- Looking at people's faces and skin tones (with the people standing off field).
- Looking at the field from each corner.
- Looking at beacons when they are shadowed by a person.
- Looking at difficult to see balls, such as obscured, shadowed, grabbed or fast moving (and blurred) balls.
- Grabbing the ball, and shooting on the yellow goal, with a person standing in the goal.
- Grabbing the ball near the blue goal and near sidelines, then turning around to face toward the yellow goal before kicking.

So while this log is lacking certain elements (scrums, shooting on the goal with a red robot in the goal, many robots on the field etc) it gives a lot of different elements of varying difficulties.

A tally of the items present in the log is given in table 2.4.

All frames used for testing depicted legal game situations. If a person was present in the image, then only if they were off the field, or dressed as a referee was the image included in the

Object	Count
Frame	2298
Ball	812
Blue Goal	306
Yellow Goal	545
Pink on Yellow beacon	275
Yellow on Pink beacon	261
Pink on Blue beacon	230
Blue on Pink beacon	263

Table 2.4: Summary of contents of the frames used in this vision comparison.

analysis. Pictures with multiple balls on the field were discarded, and images when the dog is off the ground were discarded.[14]

Results Summary

The number of false positives is quite low in all cases, which is always an intentional feature of the rUNSWift vision system. Although the numbers are low and within a noise margin of each other, they do give us some data for comparison. The false positive counts are much greater, which was to be expected – the human vision system is by far superior to that of a robot for detection of objects although a robot will still give more accurate information when it detects an object (such as distance and heading estimates).

The separate vision system gave more false negatives compared to the other vision system. This is mostly attributed to the separate colours making the criteria for beacons higher. An alternate theory is that the time taken distinguishing between the two colours was time that should have been spent working on other parts of the classifier, thus the performance could have been improved.

The extra checks added to the code for the two vision systems with merged pink and red did not make much difference to the false negatives, however they did reduce the number of false

[14]Working out when the dog is on the ground is a matter of judgement. If there was doubt, we discarded the frame – this was done before we tested the machine vision systems so as not to inadvertently skew the data.

Test	Separate	Merged Without Checks	Merged
Ball	107 false negatives	122 false negatives	122 false negatives
Misfit Ball	3%	3%	3%
Yellow Goal	146 false negatives	135 false negatives	135 false negatives
	0 false positives	1 false positive	1 false positive
Blue Goal	155 false negatives	152 false negatives	152 false negatives
Incorrect Gap	8%	6%	6%
Pink on Yellow beacon	205 false negatives	106 false negatives	107 false negatives
	0 false positives	1 false positive	1 false positive
Yellow on Pink beacon	192 false negatives	48 false negatives	50 false negatives
	1 false positive	4 positives	2 false positives
Pink on Blue beacon	202 false negatives	142 false negatives	142 false negatives
Blue on Pink beacon	199 false negatives	112 false negatives	112 false negatives
	3 false positives	2 false positives	1 false positive

Table 2.5: Summary of results from vision comparison. Full results are in Appendix A.

positives (the testing log was not used in the generation of these checks, so this is not a case of over-fitting). This was actually a surprising result for the author, it was expected that many more valid objects would be rejected due to corner cases, but the results indicated otherwise.

Performance in rUNSWift Lab

Looking at the few logs we have from the rUNSWift lab with a colour table designed for the lab, but pink and red merged together, false beacons are frequently seen in the orange wall and yellow goal (see Figure 2.18). Without further work, this method does not work satisfactorily in the rUNSWift lab. The orange wall is seen as a mixture of orange, background, yellow with a red fringe around the outside (see Figure 2.19). So, from most positions on the field, the yellow goal has a pink blob above it, which suggests that it is a beacon. Although the extra checks added will rule out many of the cases, the shape of the red fringe is essentially random, and at times the features detected will be rectangular.

It would be possible to add some more checks to handle this case, and it may improve the reliability and robustness of the vision system. However, it is much simpler to simply distinguish between the two colours here.

Conclusions

The improvements to the tools used in vision allowed for much faster calibration and evaluation of the entire vision system. This gave a great deal of flexibility to allow risky ideas such as merging red and pink together to be attempted and evaluated.

Looking at the results from merging pink and red in Bremen, it is clear that this was the best of the 3 methods evaluated. However, because the modifications made assumptions about the domain (specifically that there wouldn't be a orange wall behind the goal) the idea of merging the colours doesn't work very well in the rUNSWift lab.

Even though the initial tests in the rUNSWift were disappointing, it should be possible to refine the checks and add more checks to remove the extra false positives. This would result in a more robust vision system overall. In a way, this is an extension of the original approach to vision in the rUNSWift code [9] – a move away from colour classification which deteriorates with lighting changes, however rather than using edges in YUV space as a lighting invariant

Figure 2.18: Some examples of false beacons being seen in the rUNSWift lab when pink and red are merged into one colour.

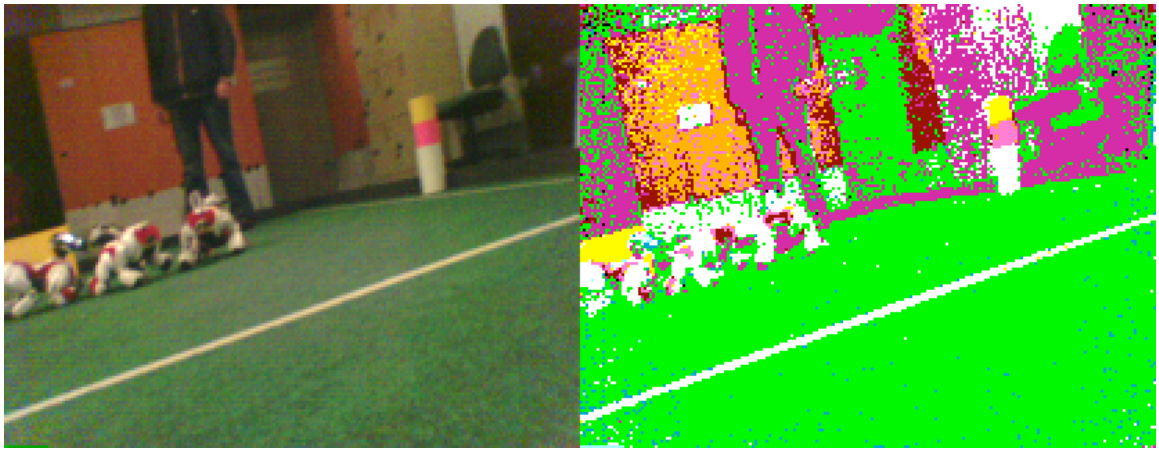

Figure 2.19: The classification for a typical frame taken in the rUNSWift lab. Note the red fringe around the orange wall.

property, we use the shape of objects.

The extra checks added to the code perform very well for the cases they were designed for, they rarely discard valid beacons, and for most angles of viewing a red robot, they discard any potential beacons seen before any of the other sanity checks are required. This gives some confidence in the general approach of relaxing the low-level recognition while tightening the high-level constraints.

Chapter 3

Hacks

Although most changes to the vision code were well understood and could be justified and explained, there were still some parts which manage to escape the scientific approach and called for solutions which got "results" without necessarily knowing why they were the case. This section seeks to give a short description of the two major areas where this applied – the YUV transforms we apply to every image, and distance calculations.

3.1 YUV Transformations

As noted in Section 1.5.2, rUNSWift uses camera specific YUV transformations to attempt to negate the variation in the images captured by the different dogs. The transforms are linear equations for each channel of colour. The process of calculating these shifts is described more fully in [6] however we will briefly describe the method here for completeness:

1. Each dog has a series of coloured cards shown to it under constant lighting conditions with the vision frames being logged.

2. One dog[1] is chosen as the "base dog" and given identity functions for the transformation.

3. Each dog's images of the coloured cards are compared to those of the base dog, and in YUV space, a line of best fit is found such that the colours will appear the same to all dogs.

[1] Typically the dog which we expect won't break for a while.

4. The constants for the functions are put into the code, indexed by MAC address, enabling memory sticks to be dog-independent.

This process took a significant amount of time (we would find shifts for two dogs at a time, logging pictures for one dog, then the base dog, then the other dog to minimise the effects of changing lighting conditions) and effort was required to make sure the images were taken in identical conditions (frames including shadows from people walking past had to be discarded). Although it would be nice to be able to calculate the transformations once and then leave the numbers as is, the significant turnover of robots as they returned from repair or were sent for repair meant that finding these transformations was an ongoing task.

3.1.1 Hand Tuning Transformations

While in Bremen, some of the dogs which had recently returned from repair were found to have poor vision, even though they had been calibrated using the process described above. Instead of repeating the process with the cards, we opted to hand tune the transformations. Some modifications had already been made to the vision tool to allow instant feedback from modifying the values in the functions (see Figure 3.5) so this task was feasible. The approach used was:

1. Find an image which contains as many interesting objects in it as possible (for example, a shot containing a goal, a ball, a beacon and a robot).

2. Switch the view to display the classified image.

3. Search through transformations of the form:

$$y' = y + k_y$$

$$u' = u + k_u$$

$$v' = v + k_v$$

   Looking for a transformation which minimises classification error in the test image.

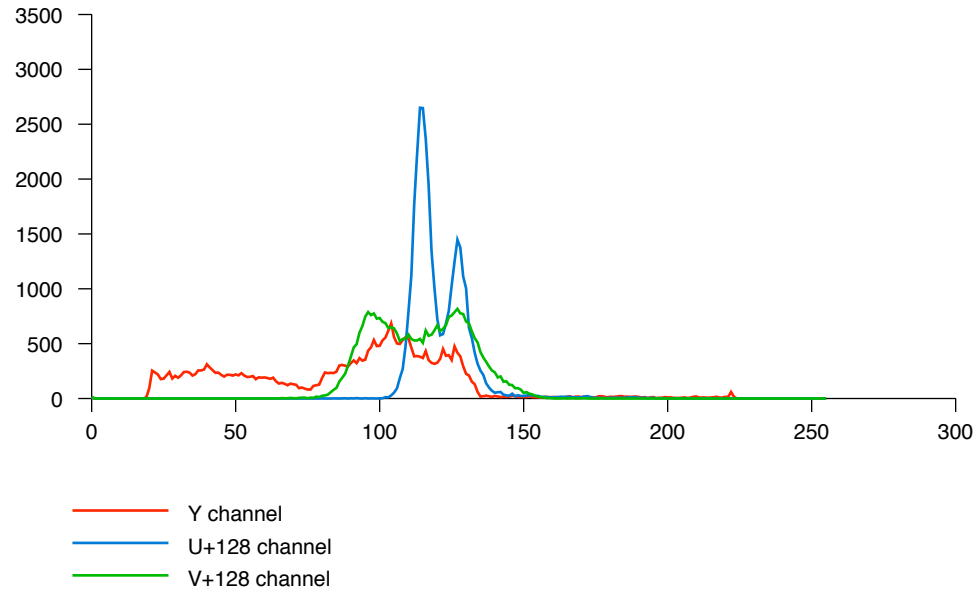

Figure 3.1: The histogram for the image in Figure 3.2.

4. Once a good transformation was found, it was tested on other images, and compared to the identity transform and also the 6 adjacent transformations (with $k'_i = k_i \pm 1$ for each $i \in \{y, u, v\}$)

Note that we never tried modifying the coefficient, leaving it at 1 for each channel. The reason for this is that typically the coefficients are very close to 1. (most coefficients are in the range 0.98–1.02, with only a few outliers). Since the numbers will be quantised to integers, the actual effect of these multipliers is quite small. For a coefficient of 0.98, there are only 6 extra 'steps' in the resulting function, so in this case there are only 6 points[2] where the function locally behaves as anything other than a constant offset. Looking at the histogram in Figure 3.1, the different channels are concentrated in specific ranges. The range for the U channel is the narrowest, being significant only between about -28 and 22. Over a range this small, a coefficient in the range of 0.98 to 1.02 will make no significant difference. The U and V channels have wider ranges, but even the Y channel, which ranges from 19 to 223 has only 4 points where the coefficient in the transformation makes a difference when it is in the range from 0.98 to 1.02.

[2]Over the range from 0–255 or -128–127 that is.

Figure 3.2: An image containing small samples of all the colours on the field.

These changes are difficult to quantify. All the dogs which had hand-tuned transformations were selected because they were performing poorly. After the hand-tuning, they appeared to be greatly improved, so from that perspective the it gave an obvious improvement. Whether this method gives better transformations than the automatic method is unknown. What is known is that this method allows for the dog specific calibration to be done at any time, and without requiring any special cards or lighting conditions.

3.1.2 Odd/Even Scanlines

An observation was made that the images captured from every camera had a pattern of alternating light and dark lines. Demonstrating this phenomenon on paper is at the mercy of the printer. An example image is shown in Figure 3.3. To help show the banding, a image with modified colour curves is included, these curves are continuous, but are much steeper around the colours which are prominent in the image to give them more contrast. Finding a single curve which showed the lines clearly for both background and foreground elements was difficult, so the processed image is actually a combination of two sets of curves, one for the field and one for the background.

Often, the alternating colours would fall on either side of a symbolic colour boundary, causing the banding to appear in the classified image as well, this led to some investigation to see if the banding could be removed. The investigation was only brief, as it did not appear to be a

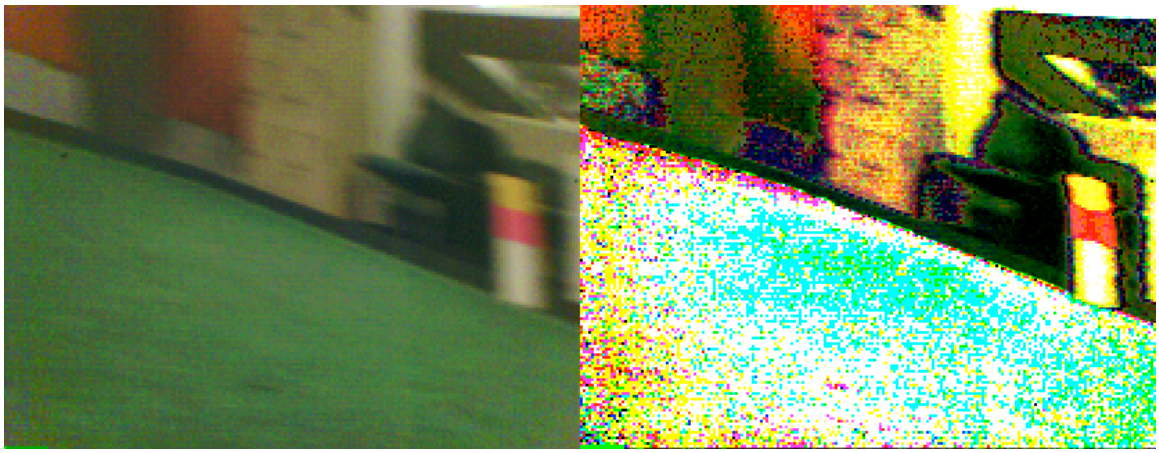

Figure 3.3: An image taken from a robot which shows the odd/even scanline pattern and the same image with modified colour curves in an attempt to highlight the banding.

significant weakness of the system, nor was it evident that a useful solution existed.

Further analysis of the bands after this work was done indicated that the noise was only present in the U and V channels. This can be seen in Figure 3.4 where each channel of two images is displayed separately. In order to help see the banding (or lack thereof in the Y channel), an Unsharp filter was applied to half of each channel. This processing seems to give better results than modifying the colour curves for these grey-scale images.

3.1.3 Noise Model

Looking at the images the bands appear in, it seemed that the bands were being introduced by the camera, since they were present in blurred images, and images without ring correction applied. We made the assumption that we could then model the difference between odd and even scanlines by applying different linear transformations to each channel. It should be pointed out that the linear transformations are applied after ring correction, so we have made an implicit assumption that the error remains invariant after ring correction.[3]

[3]Whether or not this is true is uncertain, however due to time restrictions, it did not seem worthwhile to change this ordering.

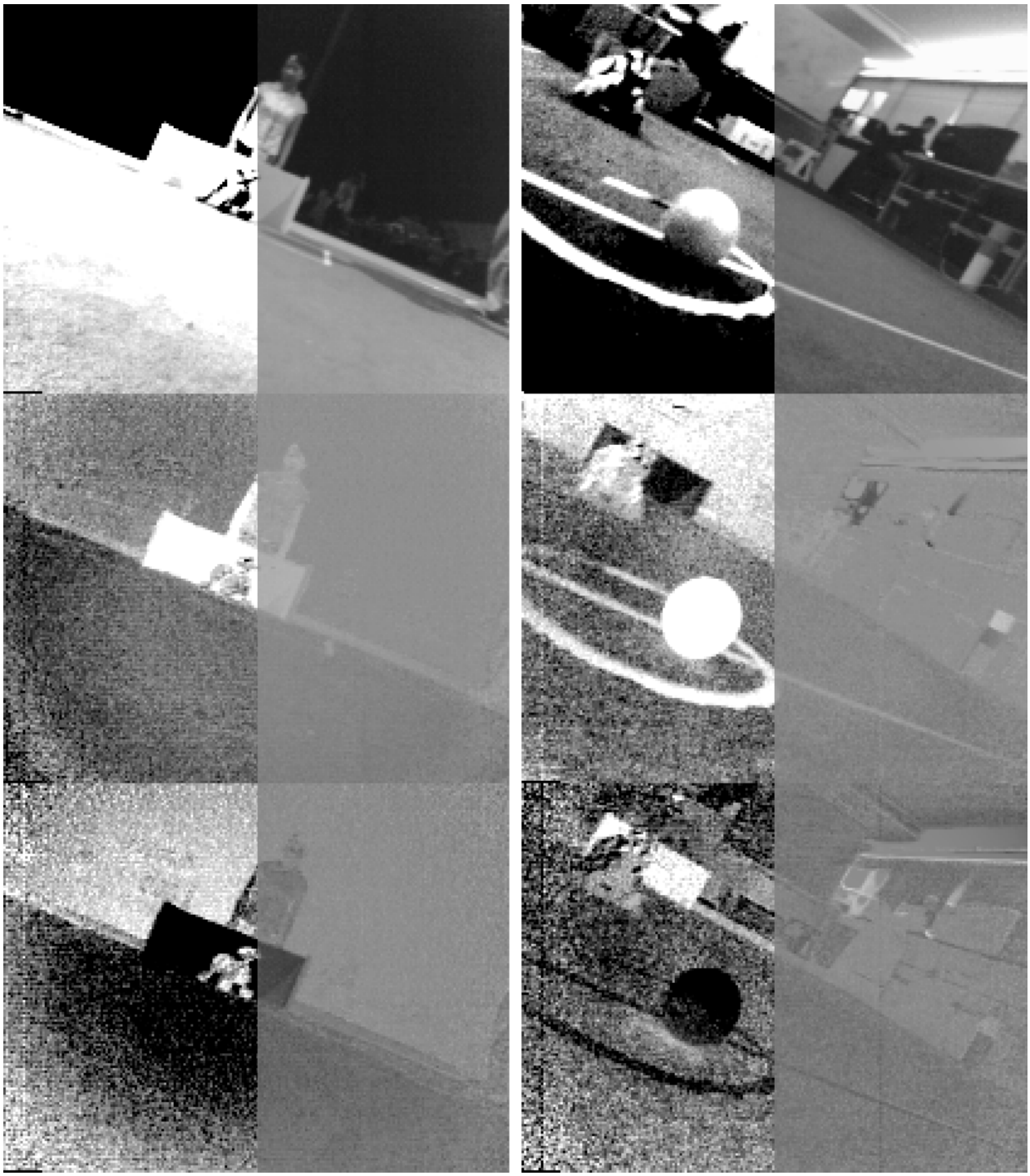

Figure 3.4: Separated channels for an image taken in Bremen (left) and and image taken in the rUNSWift lab (right). The left hand side of each image has been sharpened with an Unsharp filter to help show the banding effects. Top: Y channel. Middle: U channel. Bottom: V channel.

3.1.4 Method

Our method for attempting to remove the banding was to log camera images of matte colours, and then run a function minimiser over the function B given the input of a set s of linear transforms for the Y, U and V channels for odd and even scanlines. The function B was the squared colour difference between training pixels and the pixels directly above or below them. More formally:

$$B(s) = \sum_{p \in P} \left(p - \frac{p_{above} + p_{below}}{2} \right)^2$$

Where P is the set of pixels which are in training images, some distance from the edge occurring on odd scanlines and relatively close in YUV space[4], p is the YUV value at a pixel and p_{above} and p_{below} are the YUV values of the pixels directly above and below p respectively. We define the square of a YUV triple as the sum of the squares for each component.

The transforms for the even scanlines were fixed, and the odd scanlines were modified by the function minimiser. A brute force minimiser was used for its simplicity, searching the domain near the even transform within a certain range.[5]

The process for selecting training data and evaluating different transformations was built into the offline vision tool and the interface can be seen in Figure 3.5.

3.1.5 Results

After the function minimiser finished[6] the difference between the odd and even transformations was minimal. Often they were identical, other times only one or two numbers were changed. The search domain was increased in some dimensions and reduced in others, to see if there was some other minimum which was just outside the range we were searching, however similar results were found with the different sized domains.

[4]This was to prevent noise from skewing the data.

[5]Coefficients were allowed to vary by ± 0.02 in 0.01 steps and offsets could vary by ± 5 with integer steps.

[6]Had the method proved useful, a faster minimisation method would have been used.

Figure 3.5: The UI for experimenting with different YUV transformations and learning different odd/even shifts.

3.1.6 Conclusions

From the negative results, the obvious conclusion to draw is that at least one of the assumptions made about the noise model was incorrect. Although it would have been possible to remove some of the assumptions, the time taken would have increased significantly, still without any guarantee of success so the project was abandoned.

3.2 Beacon Distance Calculation

While in Bremen, it was noticed that the goalie positioning was consistently offset to one side of the field. Further investigation led to the discovery that the beacons with pink on the top were consistently estimated to be too near. We estimate beacon distances using the following method:

1. The beacon width is estimated by the widest pink beacon feature found.

2. The height of the pink section is estimated by searching the pixels above[7] each beacon feature for the highest pixel that isn't pink. A similar search is done but in the opposite direction. The maximal distance between top and bottom is stored.

3. The beacons have a diameter of 10cm, and each coloured section has a height of 10cm, so in an ideal situation the width and height of the pink section should be identical. If the width of the pink section is greater than 90% of the height then we estimate the beacon height as $pink_{width} * 4$ otherwise, we use $pink_{height} * 4$.

4. The distance to the beacon is then:

$$D = \frac{a}{beacon_{height} + b}$$

   For constants a and b.

In order to correct the beacon distance calculations, different constants a and b were used for the beacons with pink on top to those with pink on the bottom. We took 10 sample images

[7] The direction to search is dependant on the horizon.

of each beacon from three different angles[8] from 100cm, 150cm and 200cm[9] away. Taking the average distances, we then fit the hyperbola to the points given by the 100cm sample and the 200cm sample for a single beacon. This hyperbola was then evaluated by checking it against the other beacon with pink on the top, and for the 150cm case.

3.2.1 Evaluation

The obvious evaluation criteria was that the goalie stood in the middle of the goal box for kickoff. This was the case, so at least the symptom of the problem was fixed. Being able to evaluate the correctness on other players is very difficult without being in Bremen, however from observing their game-play, none seemed to be particularly lost, and all players positioned themselves reliably for kick-off. However, given that this was a change which only moved the goalie a few centimetres to one side, it is unlikely that anything would have been noticed for non-goalie players even if it made the distance estimate worse.

3.2.2 Explanation

The exact reason for this is unknown. The most likely reason seems to be that the colour occurring at the boundary of pink and background was classified as pink, but the colour occurring at the boundary of pink and white was classified as white. This led to beacons with pink on the top from appearing taller, and therefore closer while the beacons with pink on the bottom didn't appear any closer. Given that we were very conservative when classifying the blue colour to avoid seeing too much blue in the background, this would make sense, as the shade of blue at the very top of the beacon will be the most ambiguous. The pink at the top of the beacon will have quite a relaxed classification, as there aren't any other competing colours in that area (in the past, the edges of pink would often be red, but merging pink and red makes this disappear).

[8] One directly infield, one in line with the goal at that end of the field and one from near the centre of the field.

[9] This distance was chosen not only because it is a round number, but the goalie will usually be about 200cm away from a beacon.

Chapter 4

Robot Recognition

The gameplay strategy of the rUNSWift team mostly ignores robots from other teams, basing global strategy only on the location of other teammates, as announced by wireless. On a smaller scale however, there is some degree of robot recognition, for example when shooting on a goal, a gap is selected in the goal – while this is detected as an area that is goal coloured, it has the effect of avoiding the robot standing in the goal. This is a primitive form of robot recognition which works well for the purpose it was made for.

There is also support in the code for the detection of shadows on the field. These shadows are assumed to be cast by other robots, so we can see robots in this manner as well. This method of robot recognition works very well for a short range, but as robots get further away, the size of the shadow in the image becomes too small and it is no longer detected. This year, this form of robot recognition was never used in competition, as the performance penalty didn't offset the gains made in the strategy.[1]

From these two methods, gap detection will only work if the other robot is in the goal (and it doesn't explicitly locate the other robot) and shadow detection only works at close range. This section looks at a method which attempted to be slightly more general.

[1]The infra-red sensor was used for a similar purpose instead.

4.1 Passing Challenge

The RoboCup competition in 2006 had a technical challenge called the passing challenge. In it, three dogs would be placed on the field, where they would stand in designated areas and pass the ball between each other[12]. While our localisation is generally quite good, as distances between robots increased, any errors in heading would be amplified leading to missed passes. To try and increase the accuracy of passes, some robot detection code was written.

4.2 Goal

The goal of this robot recognition was a method for identifying the heading to a robot. The distance was not essential, nor was the pose of the other robot[2]. Since the rules stated that robots of any chosen colour could be used, it was decided that this recognition would only work for red robots (especially in the rUNSWift lab, blue robot uniforms appear very much like the background in the lab).

The recognition was required to comfortably work at distance from 75cm to 2m (which was the range of distances between the centroids of the areas given to the robots[3]) and to be robust to background noise.

4.3 Method

When we are processing an image, we keep track of points which appear to be field edge boundaries. A line is fit to them using the ransac algorithm as described in [4]. In some cases, the we don't find a good enough field edge, in which case we do not try to do any processing. Assuming a field edge is found, we count the number of red pixels in each column when they are between 5 and 25 pixels below the field edge. After creating these tallies, we process the list of numbers:

1. Decrease resolution by pairing together tallies.

[2]While knowing the pose could be useful, it was assumed that the other robot would always be facing towards the robot doing the recognising, as it was awaiting a pass.

[3]We assume the robots are standing near the centre of their circles.

2. Reduce noise by thresholding.

3. Find each non-zero tally and store the heading in a list.

4. Remove the left-most and right-most items from the list.

5. Find the average heading of the items remaining.

Many variations on this method were tried (not changing resolution, different thresholding, removing outliers differently and taking a weighted average for the final step) however this combination seemed to give the most reliable results.

4.4 Evaluation

Unfortunately, the method was not reliable enough to be used in competition. On analysis of log files, the weaknesses in the method were found to be:

- When the ball is grabbed, the head is lower than normal, this means that other robots will be seen higher up in the image, relative to the horizon. The higher an object is in the image, the more error there will be, as it is much more likely to have background interference.

- The front uniform of a dog is typically shadowed by its head, making it appear the same colour as the bottom of the wall in the rUNSWift lab.[4] This leads to either false positives when facing the wall or more false negatives all the time.

- When the robots are close to each other, the field-edge calculation will often be incorrect as there won't be a clear line in the image. This could potentially be fixed by using the horizon, instead of the field-edge, although the horizon is also error prone.

- Often the receiving robot is seen as two distinct clumps of red (since there is shadow in the middle which doesn't appear red). If one of the clumps is missed for a single frame,

[4] Although the wall should theoretically appear well above the field-edge, it only takes a single frame where the field-edge is calculated too high and a large number of red pixels will be seen. Filtering this out would be difficult.

the robot will appear to jump to one side. Given the time restrictions, we did not have time to implement a robust filter to handle this problem.

- The field of vision for the robot is very narrow. If we do not line up well at first, then there is little chance of lining up at all. While it might be possible to turn until we detect the robot with this method, then switch to a visual line-up, this method would fail when the receiving robot was not seen for some reason (which occurred frequently if the robot was in shade). After a timeout, we would switch back to using the localisation module to line up, but because there are very few visual updates while grabbed (we can't see beacons, the goal provides little information and the ball will not be visible) the lining up will be based mainly on the odometry calculations.

From real world tests, the method performed best at a range of about 80-90cm. However, even at this range, the robot detection was intermittent. At closer ranges, it was common for the robot to attempt to line up with one side of the receiver, but over-turn and end up facing away from the other robot. At longer distances, the receiving robot was usually not detected reliably enough for the method to work within the short time required.[5]

4.4.1 Conclusion

Without further work addressing the problems given above, this method is not quite suitable for use in lining up with other robots. A combination of relaxing the criteria for seeing the other robot and robust filtering may give better results. Also, the method described by the GermanTeam in [13] whereby the receiving robot will turn side-on to maximise the visible uniform while lining up would help reliability.

[5]The rules for the passing challenge at this stage meant that the ball could only be grabbed for 3 seconds.

Chapter 5

Conclusion

The ultimate evaluation function is real world competition, and after all the work put into the code, the rUNSWift team ended up placing 2nd in the World Championship after a relatively comfortable run through to the finals. While this doesn't show that the vision system was great, it does confirm that it (along with the rest of the code) wasn't bad.

The work done prior to arriving was useful, and enabled more time to be spent solving bigger problems rather than classifying colours which then allowed the merging of pink and red, along with the associated sanity checks required. The evaluation of this change gives a fairly good indication that this was a good approach to solving the problem, even though it is not quite general purpose enough to work in the rUNSWift lab.

The mixed results from the hacks and robot recognition highlight the experimentation required in this field. The ideas which shouldn't work did, and those which should have worked did not. The constant evaluation of methods used prevented the poor ideas from gaining too much momentum and allowed the good, but nonsensical ideas to survive. Although this will surely lead to some interesting conversations if the code is reviewed at a later date, like most of the rUNSWift code – it wasn't included for its elegance, it was included because it got results.

Bibliography

[1] David Billington, Vlad Estivill-Castro, René Hexel, and Andrew Rock. Non-monotonic reasoning for localisation in RoboCup. Technical report, Griffith University, 2005.

[2] Jin Chen, Eric Chung, Ross Edwards, and Nathan Wong. Rise of the AIBOs III - AIBO Revolutions. Undergraduate thesis in computer and software engineering, University of New South Wales, 2003.

[3] Mark de Greef and Dave van Soest. Automatic color calibration on a robosoccer-field. Technical report, Universiteit van Amsterdam, 2006.

[4] Enyang Huang. Road to robocup 2006. runswift 2006 behaviors & vision optimizations. Technical report, University of New South Wales, 2006.

[5] Luca Iocchi. Robust color segmentation through adaptive color distribution transformation. Technical report, Università di Roma, 2006.

[6] Daniel Chi Kin Lam. Visual object recognition using sony four-legged robots. Undergraduate thesis in computer and software engineering, University of New South Wales, 2004.

[7] Craig L. Murch and Stephan K. Chalup. Combining edge detection and colour segmentation in the four-legged league. Technical report, University of Newcastle, 2004.

[8] Walter Nisticò and Thomas Röfer. Improving percept reliability in the sony four-legged robot league. Technical report, Universität Bremen, 2004.

[9] Alex North. Object recognition from sub-sampled image processing. Honours thesis, School of Computer Science and Engineering, University of New South Wales, 2005.

[10] Kim Cuong Pham and Claude Sammut. RDRVision – learning vision recognition with ripple down rules. Technical report, University of New South Wales, 2005.

[11] Michael J. Quinlan, Steven P. Hicklin, Kenny Hong, Naomi Henderson, Stephen R. Young, Timothy G. Moore, Robin Fisher, Phavanna Douangboupha, Stephan K. Chalup, Richard H. Middleton, and Robert King. The 2005 NUbots team report. Technical report, University of Newcastle, 2005.

[12] RoboCup Challenge Rules, 2006.

[13] Thomas Röfer, Jörg Brose, Eike Carls, Jan Carstens, Daniel Göhring, Matthias Jüngel, Tim Laue, Tobias Oberlies, Sven Oesau, Max Risler, Michael Spranger, Christian Werner, and Jörg Zimmer. GermanTeam team description report. RoboCup 2006 International Symposium CD, 2006.

[14] Jing Xu. rUNSWift 2004 low level vision. Undergraduate thesis in computer and software engineering, University of New South Wales, 2004.

[15] Juan Cristóbal Zagal, Javier Ruiz del Solar, Pablo Guerrero, and Rodrigo Palma. Evolving visual object recognition for legged robots. Technical report, Universidadde Chile, 2004.

Appendix A

Analysis of Log Files

These are the results from the analysis of vision. Only frames with at least one error are included in the results. A '+' indicates a false positive, a '–' indicates a false negative. An 'M' means either misfit (for the ball) or poor gap (for a goal which was detected).

	Separate							Merged w/o Checks							Merged					
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP
9281						–							–							–
9381						–							–							–
9435			–			–					–							–		
9450																				
9581																				
9619																				
9677										–							–			
9727																				
9773																				
9823																				
10065							–													
10107			–			–					–		–					–		–
10311							–													
10357						–							–							–
10599						–				–							–			
10757				–								–							–	
10832																				
11066				–																
11366	–		–					–			–				–			–		
11542						–														
11650							–													
11858			–																	
12985			–																	
13197			–																	
13418						–							–							–
13631							–					–							–	
13756				–			–					–							–	
13873	–						–							–						–
14127				–			–					–							–	
14261							–													
14428							–													
14503							–													

	Separate							Merged w/o Checks							Merged					
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP
14582							–													
14674							–													
14741							–													
14807					–		–					–							–	
14916							–													
15149					–		–													
15241					–		–					–							–	
15333	–				–		–	–							–					
15416					–		–													
15500					–							–							–	
15579					–		–													
15666					–															
15846					–															
15925			–		–					–							–			
16021			–		–					–		–					–		–	
16104					–															
16296					–															
16538					–							–							–	
16609	–		–					–		–					–		–			
16676	–				–			–							–					
16759	–			–				–							–					
16805	–			–				–				–			–			–		
16868	–			–				–				–			–			–		
16934	–			–				–				–			–			–		
17005	–							–							–					
17085				–								–						–		
17226				–																
17285	–			–				–				–			–			–		
17343												–						–		
17431	–			–				–				–			–			–		
16506																				
17744				–								–						–		
17798				–								–	+					–		
17864				–								–						–		
17923				–								–						–		
17990				–								–						–		
18052				–								–						–		
18315						–														
18607				–		–						–						–		
18928																				
19112					–															
19170										–							–			
21301				–																
21468	–	–						–	–					+	–	–				
21685		–						–	–						–	–				
21756				–																
21964																				
22235	–	–						–	–						–	–				
22306	–							–							–					
22394	–								–							–				
22473	–					–		–					–	+	–				–	+
22540				–																
22632	–																			
22682	–							–						–	–					–
22753				–																
22832						–							–						–	
22899						–	–	–					–	–	–				–	–
22957						–							–						–	
23111	–						–	–						–	–					–
23282																				

	Separate							Merged w/o Checks							Merged					
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP
23349						–							–							–
23416				–																
23478							–													
23754	–	–							–					–		–				–
23820		–						–	–						–	–				
23887	–	–						–	–						–	–				
23941	–						–	–							–					
24000		–							–					–		–				–
24066		–					–							–						–
24121		–					–		–							–				
24162		–							–					–		–				–
24217		–					–		–							–				
24375		–					–	M	–					–	M	–				–
24559					+		–							–						–
24663								–							–					
24713							–							–						–
24830	–						–	–							–					
24892	–					–														
24963						–							–							–
25009				–							–							–		
25072	–					–		–							–					
25138				–							–							–		
25209						–		–							–					
25364						–							–							–
25501	–					–		–					–		–					–
25635																				
25722	–						–	–							–					
25793					–															
25943					–							–							–	
26018	–							–							–					
26123	–						–	–						–	–					–
26173							–	–						–	–					–
26231	–						–	–						–	–					–
30652																				
31332			–							–							–			
31549			–							–							–			
31803			–							–							–			
31866																				
32179														+						+
32629					–							–						–		
32884					–							–						–		
33659							–													
33880							–													
34093	–	–					–	–	–						–	–				
34168		–					–							–						–
34243							–													
34331								–	–						–	–				
34473		–					–		–							–				
34548		–					–		–							–				
34844							–													
36024							–													
36249							–													
37054					–															
38735				–																
39081				–																
39136		–				–			–				–			–			–	
39182							–													
39828							–													
40003				–																
40837		–				–			–				–			–			–	

	Separate							Merged w/o Checks							Merged					
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP
40904		–				–			–				–			–				–
40954		–				–			–				–			–				–
41004		–				–			–				–			–				–
41050		–				–			–				–			–				–
41104		–				–			–				–			–				–
41655		–																		
41843		–				–			–							–				
41913		–				–			–							–				
41972		–				–			–							–				
42034		–				–			–							–				
42097						–														
42222													–							–
42293																				
42656				–																
42777	–	–						–	–						–	–				
42827	–	–						–	–						–	–				
42885	–	–						–	–						–	–				
43035							–													
43081		–				–			–							–				
43135		–				–			–							–				
43252		–				–			–							–				
43357		–				–		–	–						–	–				
43461		–				–			–							–				
43511		–				–			–							–				
43565		–				–			–							–				
43619		–							–				–			–				–
43803		–					–		–							–				
44383							–							–						–
45112				–																
45321																				
45509				–																
45767				–							–							–		
45947				–																
46472				–																
47039				–							–							–		
47152						–														
47281				–																
47436						–														
47577				–							–							–		
47811			–	–																
47978						–														
48257						–							–							–
48562						–							–							–
48829						–							–							–
49050		–																		
49141						–							–							–
49208				–																
49408						–							–							–
49784						–							–							–
50414																				
51944						–														
52132	–	–						–							–					
52624	–	–						–	–						–	–				
52670		–						–	–						–	–				
52724	–	–						–	–						–	–				
52853		–					–		–							–				
53087																				
54713		–					–		–							–				
53909								–							–					
54038					–															

	Separate							Merged w/o Checks							Merged						
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP	
54171							–					–							–		
54247					–		–														
54292					–		–														
54426							–														
54547							–	–							–						
54597							–														
54972		–					–		–					–		–					–
55026		–																			
56023																					
56082		–							–							–					
59185																					
59706	–					–		–					–		–					–	
60015	–					–		–					–		–					–	
60065	–							–							–						
60111	–							–							–						
60257	–							–							–						
60303	–							–							–						
60732	–							–							–						
60966	–							–							–						
61016	–							–							–						
61153	–							–							–						
61337	–							–							–						
63005							–														
63235			–																		
63264			–	–						–	–						–	–			
63310																					–
63566				–		–					–		–					–		–	
63606				–							–							–			
63643				–		–					–							–			
63693				–							–							–			
63743				–							–							–			
63785				–							–							–			
63831											–							–			
63877				–							–							–			
63919				–		–															
63973				–		–					–							–			
64015				–		–					–							–			
64052				–		–					–							–			
64106				–		–					–							–			
64152				–							–							–			
64186				–		–					–							–			
64231				–							–							–			
64269				–		–					–							–			
64323				–							–							–			
64377				–		–							–							–	
64444				–							–							–			
64511	–		–	–						–	–						–	–			
64561			–	–						–	–						–	–			
64611			–	–						–	–						–	–			
64724			–	–						–	–						–	–			
64940										–							–				
65132			–		–					–		–					–		–		
65178							–							–							–
65395							–							–							–
65495					–							–							–		
65591					–		–					–							–		
65787							–							–							–
65933												–							–		
66000							–							–							–
66271				–																	

	Separate							Merged w/o Checks							Merged					
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP
66317								M			–				M			–		
66371				–		–					–							–		
66417				–		–		M			–				M			–		
66549				–		–					–							–		
66517				–							–							–		
66575						–														
66634				–		–					–							–		
66680						–														
66730				–							–							–		
66780						–														
66817				–							–							–		
66859				–							–							–		
66913						–														
67164				–																
67405							–													
67572						–														
67626																				
67689							–													
67843				–		–					–							–		
67927							–													
68123						–					–							–		
68227																				
68381						–														
68436				–	–															
68636						–							–							–
68703				–							–							–		
68778																				
68919						–							–							–
68982											–							–		
69186				–																
69245					–		–					–							–	
69303																				
69487																				
69649				–							–							–		
69704					–							–							–	
69900				–																
69958	–			–				–		–					–		–			
70087				–				–			–				–			–		
70217				–																
70442					–							–							–	
70671			–	–						–	–						–	–		
70838				–																
70892					–							–							–	
71051				–		–					–							–		
71097	–							–							–					
71142							–													
71293						–														
71339					–															
71497		–							–							–				–
71539				–																
71601					–		–													
71760				–																
71822			–		–						–		–					–		
71872							–							–						–
71989						–														
72039			–								–		–					–	–	
72098							–													
72206																				
72252						–						–							–	
72486				–		–						–							–	

	Separate							Merged w/o Checks							Merged					
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP
72536					–															
72586							–													
72690																				
72740						–														
72794			–									–							–	
72840							–													
72982																				
73032				–		–					–		–						–	–
73069			–		–					–							–			
73115							–													
73265		–																		
73307				–							–								–	
73366					–		–					–		–					–	–
73512																				
73645							–							–						–
73816			–	–						–	–						–	–		
73862					–		–					–							–	
74025									–							–				
74091	–		–					–		–					–		–			
74137																				
74187					–							–							–	
74513	–				–		–	–				–		–	–				–	–
74563	M				–															
74608	–				–			–				–			–				–	
74646				–							–							–		
74696	–				–															
74729					–															
74779	–																			
74829					–			–				–			–				–	
74867					–			–				–			–				–	
74917			–		–					–							–			
74959				–																
75080																				
75126				–							–							–		
75222					–		–					–							–	
75263					–							–							–	
75305				–							–							–		
75413																				
75509				–																
75551				–							–							–		
76697				–							–							–		
75639				–							–							–		
75730			–	–						–	–						–	–		
75764				–						–							–			
75805				–		–		–		–			–		–		–			–
75856				–				–							–					
75897				–																
75939				–		–					–							–		
75985				–		–														
76031				–																
76110						–		–							–					
76152				–		–					–							–		
76193				–		–														
76235						–														
76327			–	–						–	–						–	–		
76381						–														
76419						–							–							–
76523				–							–							–		
76569						–														
76615						–							–							–

	Separate							Merged w/o Checks							Merged					
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP
76748			–	–						–	–						–	–		
78781																				
76823		–				–			–				–			–				–
76936			–		–							–							–	
76990				–		–							–							–
77028																				
77078						–							–							–
77182			–		–					–		–					–		–	
77244						–														
77290		–				–			–				–			–				–
77445			–	–						–	–						–	–		
77516						–														
77561						–							–							–
77653					–		–					–							–	
77703				–							–							–		
77753						–														
77795		–				–			–				–			–				–
77949				–	–						–	–						–	–	
77987				–		–					–							–		
78033																				
78220					–							–							–	
78316																				
78371																				
78429			–		–					–		–					–		–	
78504			–		–					–							–			
78592			–							–		–					–		–	
78713			–		–					–							–			
78767					–							–							–	
78971					–							–							–	
79155					–															
79188																				
79234					–															
79326								M							M					
79392																				
79447																				
79509																				
79630					–		–													
79743												–							–	
79797							–													
79918				–								+							+	
79964					–															
80005							–													
80164				–																
80210					–							–							–	
80264							–													
80427	–			–				–							–					
80485							–													
80535							–													
80648				–							–							–		
81027	–							–							–					
81073					–															
81132	–							–							–					
81198							–													
81294							–	M							M					
81357	–						–	–							–					
81528					–							–							–	
81732					–															
81786										–							–			
81987					–															
82028				–						–							–			

	Separate							Merged w/o Checks							Merged						
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP	
82324			–							–							–				
82508				–																	
82708				–																	
83355						–															
83642																					
83692				–																	
83738		–		–							–							–			
83784				–							–							–			
83863				–							–							–			
83918																					
83959				–																	
84005				–							–							–			
84055				–																	
84110				–							–							–			
84151				–																	
84185				–							–							–			
84235																					
84281				–																	
84326				–																	
84376				–																	
84418				–																	
84460				–																	
84502				–																	
84543				–																	
84585				–																	
84635				–																	
84677				–																	
84719				–							–							–			
84764		–		–																	
84806		–		–					–							–					
84869		–		–																	
84906		–		–					–							–					
84994				–																	
85040				–																	
85081						–					–							–			
85140						–															
85181				–		–					–							–			
85232				–																	
85273				–		–															
85327						–															
85369											–							–			
85419				–																	
85457				–		–					–							–			
85498				–							–							–			
85540				–		–					–							–			
85590				–		–					–							–			
85640				–		–					–							–			
85686				–		–					–		–					–		–	
85732				–		–					–							–			
85782				–							–							–			
85836				–		–					–		–					–		–	
85886				–		–					–		–					–		–	
85936											–		–					–		–	
85986				–		–					–							–			
86036				–							–							–			
86082				–		–					–		–					–		–	
86132				–		–					–		–					–		–	
86174				–		–					–							–			
86791				–									–							–	
86841				–		–							–							–	

	Separate							Merged w/o Checks							Merged						
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP	
86900					–																
87012					–		–														
87063							–														
87104							–					–							–		
87154					–		–														
87204					–							–							–		
87246					–		–														
87296					–		–					–							–		
87334					–		–					–							–		
87384					–																
87430					–		–														
87471					–		–														
87521							–														
87567					–		–														
87617					–		–														
87713							–					–							–		
87767							–														
87822					–		–														
87867							–														
87909							–														
87968					–		–														
88026					–							–							–		
89223										–							–				
89452					–																
89494																					
89619	–						–	–						–	–						–
89661					–																
89811													–	+						–	
89940			–																		
89970	–							–							–						
90103	–							–							–						
90508						–															
90570						–															
91008		–					–		–							–					
91137				–																	
91183						–															
91450	M	–							–							–					
91688		–																			
91767						–							–							–	
91809						–							–							–	
91842						–															
91880						–							–							–	
91922						–							–							–	
91972		–							–							–					
92259	–																				
92418	–	–						–	–						–	–					
92497	M	–							–							–					
92827		–							–							–					
92853							–							–							–
92964	–	–						–	–						–	–					
93006								–							–						
93048		–							–							–					
93131		–							–							–					
93298		–							–							–					
93448									–							–					
93682																					
93727								–							–						
93765				–							–							–			
93798				–							–							–			
93873			–		–					–							–				

	Separate							Merged w/o Checks							Merged					
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP
94245																				
94311		–							–							–				
94357		–							–							–				
94524		–							–							–				
94595																				
94628							–													
94662			–		–					–							–			
94795	–							–							–					
95216						–							–							–
95629	M	–							–							–				
95734								M							M					
95780	–							–							–					
95875		–							–							–				
95921																				
95967		–							–							–				
96009		–							–							–				
96051			–				–			–				–			–			–
96088					–		–													
96126					–		–					–						–		
96213																				
96251																				
96297																				
96334																				
96376																				
96539	–			–				–							–					
96593					–		–					–						–		
96622							–													
96952												–						–		
97006							–													
97085							–													
97131			–		–					–							–			
97268							–					–		–				–		–
97494						–							–						–	
97540				–		–					–		–				–		–	
97590				–							–						–			
97623				–							–						–			
97656				–																
97698				–																
97736	–							–							–					
97823						–														
97869						–														
98762																				
98803	–							–							–					
99012		–							–							–				
99404		–							–							–				
99450																				
99504																				
99629			–		–					–		–					–		–	
99688			–	–																
99746			–	–						–	–						–	–		
99804			–	–						–	–						–	–		
99846			–		–					–							–			
99929					–															
99963					–															
100088	–				–			–							–					
100126	M							–							–					
100939	–	–							–							–				
101014		–							–							–				
101051									–							–				
101181									–							–				

	Separate							Merged w/o Checks							Merged					
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP
101318				–		–														
101352	–							–							–					
101393				–							–							–		
101431				–							–							–		
101514											–							–		
101560						–							–							–
101594						–		M					–		M					–
101986						–							–							–
102386	–							–							–					
102620						–							–							–
102757			–																	
102978																				
103120												–							–	
103341																				
103383																				
103571				–							–							–		
103616																				
103888				–																
103933										–							–			
104092																				
104230						–								–						–
104367						–							–							–
104659	–							–							–					
104743	–							–							–					
104780	M							–							–					
105093	–							–							–					
105335					–							–							–	
105439																				
105514		–					–		–							–				
105552																				
105635		–					–		–					–		–				–
105760							–													
105881							–													
105977							–													
106056																				
106094		–							–							–				
106136									–							–				
106177																				
106378									–							–				
106590	–																			
106636									–							–				
106924							–							–						–
107074		–							–							–				
107112		–							–							–				
107233		–							–							–				
107270						–														
107341		–							–							–				
107416																				
107454	M	–						M	–						M	–				
107495		–							–							–				
107575						–							–							–
107675						–														
107716						–														
107754	–			–				–			–				–			–		
107787					–					–							–			
107867			–	–						–	–						–	–		
107912				–							–							–		
107942				–							–							–		
107979				–							–							–		
108013						–														

	Separate							Merged w/o Checks							Merged					
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP
108058						–							–							–
108350																				
108396		–							–							–				
108521		–							–							–				
108571									–	+					–	+				
108642	M	–							M						M					
108684		–							–	–					–	–				
108947				–		–						–							–	
108972				–																
109014			–	–							–	–						–	–	
109326																				
109535							–													
109577		–							–							–				
109710																				
109752																				
109798									–							–				
109848	–								–						–					
110010		–																		
110085		–							–							–				
110160		–							–							–				
110202																				
110244																				
110281																				
110319																				
110394					–															
110440				–								–							–	
110469				–								–							–	
110515																				
110598				–								–							–	
110807									–							–				
110890					–															
110924																				
111007	M								M						M					
111120							–													
111220							–													
111253							–													
111299							–													
111424							–													
111570		–							–							–				
111649																				
111741		–							–							–				
111779		–							–							–				
111933									–							–				
111975									–							–				
112021		–							–							–				
112050									–							–				
112092		–							M	–					M	–				
112167		–							–							–				
112538						–							–							–
112663				–		–							–							–
112984						–							–							–
113284				–																
113326						–														
113493		–							–							–				
113535							–													
113614			–	–							–	–						–	–	
113856						–							–							–
113943	–																			
114110	–								–						–					
114156				–								–							–	

	Separate							Merged w/o Checks							Merged						
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP	
114189																					
114361					–																
114398							–														
114594						–								–							–
114694				–																	
115161	–							–							–						
115553	–				–			–							–						
115595					–																
115670																					
115933																					
115975							–														
116096									–							–					
116221					–			M							M						
116342										–							–				
116750									–							–					
116905				–							–							–			
117059			–		+					–							–				
117293																					
117384							–														
117430																					
117518						–															
117722						–								–							–
117756						–								–							–
117877						–								–							–
117972	–					–		–						–	–						–
118022					+	–								–							–
118068						–								–							–
118114						–								–							–
118202						–								–							–
118277	–							–							–						
118957						–								–							–
118998				–								–							–		
119023			–									–							–		
119065							–														
119103		–					–		–					–		–					–
119140					–		–						–							–	
119624						–								–							–
119674					–																
119787	–						–	–	–						–	–					
119824		–					–		–							–					
120137				–								–							–		
120183		–					–		–					–		–					–
120275									–							–					
120521						–								–							–
120563						–								–							–
120600						–								–							–
120650						–								–							–
120683						–								–							–
121117							–														
121163							–														
121438									–							–					
121484									–							–					
121614		–							–							–					
121664		–							–							–					
121743					–																
121780					–								–							–	
121818			–		–					–			–				–			–	
121972			–		–					–							–				
122018			–		–					–							–				
122177			–																		

	Separate							Merged w/o Checks							Merged					
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP
122218	–																			
122264	–							–							–					
122564				–							–							–		
122610						–							–							–
123040																				
123073																				
123223		–							–							–				
123278																				
123319			–	–						–							–			
123399																				
123440				–							–							–		
123520						–							–							–
123595				–																
123641	–							–							–					
123782			–	–							–							–		
123820	–							–							–					
124083																				
124170					–															
124216									–							–				
124262						–							–							–
124300						–							–							–
125225									–							–				
125422		–							–							–				
125530		–							–							–				
125576							–													
125772		–							–							–				
125909				–		–														
126022					–					–		–					–		–	
126076			–		–					–		–					–		–	
126193			–		–					–							–			
126352	M							M							M					
126760												–							–	
126810							–													
126860		–							–							–				
126894						–														
126931						–														
126973						–							–							–
127611	M						–													
127661							–													
127820		–							–							–				
128066		–						–	–						–	–				
128103		–							–							–				
128145																				
128216		–																		
128253																				
128320																				
128358																				
128358				–							–							–		
128641																				
128687						–														
128729			–	–						–	–						–	–		
128900		–							–							–				
128942																				
129092						–														
129125				–							–							–		
129171			–		–					–		–					–		–	
129213				–		–					–							–		
129250						–														
129284						–							–							–
129396						–							–							–

	Separate							Merged w/o Checks							Merged					
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP
129680						–							–							–
129713						–							–							–
129751								–							–					
130297								–							–					
130506								M							M					
130560	M							M							M					
130806				–							–							–		
130994				–		–					–							–		
131035																				
131140																				
131181					–							–							–	
131223					–							–							–	
131423							–													
131461							–													
131490							–													
131603							–													
131719							–							–						–
131878							–							–						–
131961							–													
132612							–													
132641																				
132679			–	–						–							–			
132716				–							–							–		
132745						–														
132787			–		–															
132841					–															
132891								M							M					
133033	–							–							–					
133075										–							–			
133104								–							–					
133138					–							–							–	
133496					–															
133525												–							–	
133792	–							–							–					
133834	M																			
133980					–															
134297											–							–		
134368								–							–					
134401	M	–							–							–				
134547		–							–							–				
134576						–														
134622						–														
134652						–														
134685						–														
134718						–							–							–
135023		–							–							–				
135794						–														
135936		–							–							–				
135965																				
135995							–							–						–
136320																				
136441				–							–							–		
136478						–							–							–
136720			–	–						–	–						–	–		
137137				–																
137304							–													
137350					–							–							–	
137429				–																
137604					–							–							–	
137692				–																

	Separate							Merged w/o Checks							Merged					
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP
137959			–		–					–							–			
138013				–							–							–		
138138							–													
138293				–		–					–		–					–		–
138389							–													
138580						–							–							–
138660							–													
139043				–																
139152							–													
139231					–															
139402							–													
139823						–														
139898							–													
139965					–								–						–	
140094	–	–						–	–						–	–				
140161		–							–							–				
140290						–														
140328				–																
140361				–						–	–						–	–		
140436				–							–							–		
140466						–							–							–
140624						–														
140703						–							–							–
140829				–							–							–		
141045				–																
141350						–							–							–
141387				–							–							–		
141429					–							–							–	
141475					–							–							–	
141779																				
141942									–							–				
142142																				
142480						–							–							–
142518						–							–							–
142551						–							–							–
142601						–							–							–
142830																				
143135																				
143223	M	–							–							–				
143285		–							–							–				
143339	–	–						M	–						M	–				
143410	–	–						–	–						–	–				
143765							–						–							–
143836				–						–	–						–	–		
143877																				
143961					–		–						–							–
143994					–		–						–	–						–
144036					–															
144069					–								–							–
144169																				
144265				–							–							–		
144303						–														
144858	M							–							–					
145208						–							–							–
145245		–				–			–				–			–				–
145450		–				–			–				–			–				–
145529		–				–			–							–				
145567						–														
145600	–					–		–							–					
146680		–							–							–				

	Separate							Merged w/o Checks							Merged						
Frame	B	BG	YG	PB	BP	PY	YP	B	BG	YG	PB	BP	PY	YP	BG	YG	PB	BP	PY	YP	
146822							–							–							–
146893							–														
147210				–							–							–			
147247			–	–						–	–						–	–			
147289			–									–							–		
147314			–		–					–		–					–		–		
147431					–																
147481	M				–																
147523												–							–		
147710	–							–							–						
148052					–																
148086					–							–	+						–	+	
148570								M							M						
148924			–							–		–					–		–		
148966			–		–					–							–				
149099							–							–							–
149395							–														–
149579						–							–							–	
149700		–				–			–				–			–				–	
149754	–					–		–							–						
149796	–							–							–						
150396				–							–							–			
150442			–	–						–							–				
150488										–							–				
150647							+														
151147						–							–							–	
151256						–															
151297						–															
151481	–							–							–						
151627	M																				
151769	–							–							–						
151814	–							–							–						
152373																					
152465					–		–						–							–	
152503					–								–							–	
152594			–	–						–	–						–	–			
152649				–							–							–			
152686			–	–						–	–						–	–			
152824					–																
152861					–								–							–	
152999				–							–							–			
153045			–	–						–	–						–	–			
153166													–							–	
153253				–																	
153287						–															
153320		–				–			–				–			–				–	
153395		–						M	–						M	–					
153474	M							–							–						
153883						–															
154175	–							–							–						
154296	–	–						M	–						M	–					
154379		–																			

Omitted 1254 frames.

Table A.1: Full results from comparison between the vision system used in competition and a similar system which tries to distinguish between pink and red.

Appendix B

Code Listings

This appendix contains the code used for the extra sanity checks added.

B.1 Beacon Colour

```
Insanity SanityChecks::checkBeaconSanity(VisualObject& vBeacon, Color topColour,
    Color bottomColour) {
  double beaconDirectionX = vBeacon.beaconBottomColourCentroidX -
      vBeacon.beaconTopColourCentroidX;
  double beaconDirectionY = vBeacon.beaconBottomColourCentroidY -
      vBeacon.beaconTopColourCentroidY;

  // Get the point in the middle of the white part of the beacon
  int bottomOfBeaconX = static_cast<int>(vBeacon.beaconBottomColourCentroidX +
      (beaconDirectionX * BEACON_BASE_CENTROID_RATIO));
  int bottomOfBeaconY = static_cast<int>(vBeacon.beaconBottomColourCentroidY +
      (beaconDirectionY * BEACON_BASE_CENTROID_RATIO));

  int whiteCount = 0;
  for (int x = bottomOfBeaconX - 1; x <= bottomOfBeaconX + 1; x++) {
    for (int y = bottomOfBeaconY - 1; y <= bottomOfBeaconY + 1; y++) {
      if (x >= 0 and y >= 0 and x <= CPLANE_WIDTH and y <= CPLANE_HEIGHT) {
        Color c = cortex->img.classify(x, y);
        if (c == cWHITE) {
```

```
            whiteCount++;
          }
        }
      }

int topCount = 0;
for (int x = static_cast<int>(vBeacon.beaconTopColourCentroidX) - 1; x <=
    static_cast<int>(vBeacon.beaconTopColourCentroidX) + 1; x++) {
  for (int y = static_cast<int>(vBeacon.beaconTopColourCentroidY) - 1; y <=
      static_cast<int>(vBeacon.beaconTopColourCentroidY) + 1; y++) {
    if (x >= 0 and y >= 0 and x <= CPLANE_WIDTH and y <= CPLANE_HEIGHT) {
      Color c = cortex->img.classify(x, y);
      if (c == topColour or (mergePinkAndRed and (topColour == cBEACON_PINK and
          c == cROBOT_RED))) {
        topCount++;
      }
    }
  }
}

int bottomCount = 0;
for (int x = static_cast<int>(vBeacon.beaconBottomColourCentroidX) - 1; x <=
    static_cast<int>(vBeacon.beaconBottomColourCentroidX) + 1; x++) {
  for (int y = static_cast<int>(vBeacon.beaconBottomColourCentroidY) - 1; y <=
      static_cast<int>(vBeacon.beaconBottomColourCentroidY) + 1; y++) {
    if (x >= 0 and y >= 0 and x <= CPLANE_WIDTH and y <= CPLANE_HEIGHT) {
      Color c = cortex->img.classify(x, y);
      if (c == bottomColour or (mergePinkAndRed and (bottomColour ==
          cBEACON_PINK and c == cROBOT_RED))) {
        bottomCount++;
      }
    }
  }
}
```

```
  if(whiteCount + topCount + bottomCount < BEACON_NOT_COLOURED_GOOD_THRESHOLD){
    return insane(BEACON_NOT_COLOURED_GOOD);
  }

  if (whiteCount <= 2 or topCount <= 2 or bottomCount <= 2) {
    return insane(BEACON_NOT_COLOURED_GOOD);
  }

  return SANE;
}
```

B.2 Red Uniform Detection

```
// With robot-red and pink being treated the same, we need to try to avoid seeing
// any beacon features in a red robot. This will make sure all of the features
// making up the beacon group form a beacon looking thing.
// Criteria
// * Features are roughly the same size (relax for top and bottom feature)
// * Feature centres lie roughly on a line
//
void SubObject::removeWonkyBeacons(list<int>& feat, list<list<int> >& groups) {
  list<list<int> >::iterator next;
  for (list<list<int> >::iterator groupIter = groups.begin(); groupIter !=
      groups.end(); groupIter = next) {
    next = groupIter;
    next++;

    // Should this beacon group be beleted?
    list<int> thisGroup = *groupIter;

    // Get the feature sizes
    vector<double> sizes;
    vector<double> centroids;

    double leftSum = 0;
    double rightSum = 0;
```

```
double lastCentre = 3.14157;
double totalSize = 0;

for (list<int>::iterator iter = thisGroup.begin(); iter != thisGroup.end();
    iter++) {
  double thisCentre = ((features[*iter]).x + (features[*iter]).endx) * 0.5;
  double thisSize = ((features[*iter]).endx - (features[*iter]).x);

  totalSize += thisSize;
  sizes.push_back(thisSize);
  centroids.push_back(thisCentre);

  if (lastCentre != 3.14157) {
    if (thisCentre < lastCentre) {
      leftSum += (lastCentre - thisCentre);
    }
    else {
      rightSum += (thisCentre - lastCentre);
    }
  }

  lastCentre = thisCentre;
}

double m = min(leftSum, rightSum);

if (m > 3) {
  //cout << "Rejected this group. m = " << m << endl;
  groups.erase(groupIter);
  continue;
}

double averageSize = totalSize / sizes.size();
if (sizes.size() > 5) {
  sort(sizes.begin(), sizes.end());
```

```
      if (sizes[1] - averageSize > 5) {
        //cout << "Small beacon feature too awesome. sizes[1] = " << sizes[1] <<
            ". averageSize = " << averageSize << endl;
        groups.erase(groupIter);
        continue;
      }
      if (sizes[sizes.size() - 2] - averageSize > 5) {
        //cout << "Large beacon feature too awesome. sizes[siesf] = " <<
            sizes[sizes.size() - 2] << ". averageSize = " << averageSize << endl;
        groups.erase(groupIter);
        continue;
      }
    }

  }

}
```

B.3 Beacon in Goal Detection

```
void SubObject::pruneBeaconInGoal(void) {
  // This will remove any beacon objects where the bounding box overlaps
  // a goal.
  for (int i = 0; i < NUM_BEACONS; i++) {
    if (vBeacons[i].cf > 0) {
      if (vGoals[svYellowGoal].cf > 0 and objectsOverlap(vBeacons[i],
          vGoals[svYellowGoal])) {
        vBeacons[i].cf = 0;
        //cout << "Rejected a beacon in yellow goal" << endl;
      }
      else if (vGoals[svBlueGoal].cf > 0 and objectsOverlap(vBeacons[i],
          vGoals[svBlueGoal])) {
        vBeacons[i].cf = 0;
        //cout << "Rejected a beacon in blue goal" << endl;
      }
    }
```

```
  }
}

bool SubObject::objectsOverlap(VisualObject& a, VisualObject& b) {
  double rightMostLeftEdge = max(a.x, b.x);
  double leftMostRightEdge = min(a.x + a.width, b.x + b.width);

  if (rightMostLeftEdge > leftMostRightEdge) {
    return false;
  }

  double bottomMostTopEdge = max(a.y, b.y);
  double topMostBottomEdge = min(a.y + a.height, b.y + b.height);

  if (bottomMostTopEdge > topMostBottomEdge) {
    return false;
  }
  return true;
}
```