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

Object recognition from sub-sampled image processing

Alex North

Submitted as a requirement for the degree

Bachelor of Science (Computer Science) Honours

Submitted: 19 September 2005

Supervisor: Dr William Uther

Assessor: Professor Claude Sammut

The *rUNSWift* team is a joint project between the University of New South Wales and National ICT Australia. National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

Abstract

Image processing systems for robot applications must be accurate, robust and efficient in order to provide reliable information to higher level agent processes. In the domain of robot soccer, most previously successful approaches to robot vision systems have focussed on colour recognition, classification and blobbing, which provide for relatively fast and simple object recognition algorithms. However, such methods require processing of every pixel in each camera frame, regardless of its information content, creating a high lower bound on processing time. In addition, colour-based methods respond poorly to variations in lighting conditions and are plagued with problems in distinguishing colour blobs representing valid objects from those caused by noise and background information.

This report moves towards an alternative image processing system for robots based on minimal sampling of each frame. Interesting features are extracted from information-bearing regions of each image based on relationships between neighbouring pixels, and object recognition is performed over these features. This method has many advantages, including efficient implementation and a reduced dependency on finely tuned colour classifications. These lead to robustness to changes in lighting conditions and a much reduced calibration time compared with previous methods. Sub-sampling methods were successfully implemented for the UNSW/NICTA team *rUNSWift*, which placed third at the RoboCup[1] Legged League[2] World Championships in 2005[3].

[1] http://www.robocup.org
[2] http://www.tzi.de/4legged/
[3] http://www.robocup2005.org

Acknowledgements

Thank you to the 2005 *rUNSWift* team: Nobuyuki, Wei Ming, Andrew and Josh. Working with you all was a fantastic experience and we achieved great things. Thanks also to Will for your limitless support and inspiration, and to Claude for your guidance throughout the year. Finally, unbounded thanks to Jessica for helping me to perfect this report; any remaining errors are my own.

Contents

List of Figures

Chapter 1

Introduction

Vision is an extremely important sense for humans, providing our most detailed interpretation of the environment surrounding us. Due to a combination of this extremely high level of detail and the strong dependence of vision upon environmental conditions, the development of vision systems for autonomous robots is a complex task. Many robots rely on more simple sensors such as sonar or range finders in order to process input accurately and efficiently. However, as robots are required to perform more complex tasks in a human-centric world it will be necessary to develop accurate and robust robot vision systems. These systems must be able to provide reliable and stable information, continuing to function under changing conditions. The visual processing must also be extremely efficient, allowing an agent to respond quickly to a dynamic environment.

Within the domain of the RoboCup four-legged league previous vision systems have relied heavily on the colour of objects, classifying symbolic colours detected in each visual frame and attempting to form objects by grouping neighbouring similarly-classified pixels. However, as lighting conditions change, the colours of real-world objects change and such methods become unstable, relying on a finely-tuned colour segmentation system. These symbolic colour-based methods must also process each image frame in its entirety, where there is frequently a large amount of redundant visual information. In contrast, this report presents a system that shifts

the focus to recognising sparse visual features based on the relationships between neighbouring pixels, and detecting objects from a minimal number of such features.

1.1 Robot vision

Robot vision must be accurate, robust and efficient. Accuracy is vital in order to provide reliable information to higher level agent processes. Accuracy entails reliably detecting objects that appear in an image and not falsely detecting objects that don't. Errors in either of these cases will cause unstable or contradictory information to be presented to higher processes, adversely affecting an agent's behaviour. There exist unavoidable trade-offs in tuning a system for reduction of both falsely recognised and falsely rejected objects. As a system is tuned to reduce the probability of one type of error the probability of the other occurring increases. This trade-off is common to many digital processing applications, such as biometric identification and speech recognition. It is also important that the properties of detected objects, such as size and distance from the observer, are accurate and stable in order for such information to enable complex and finely controlled agent behaviours.

Robustness is also of high importance. It is usually impossible to predict the exact conditions under which an agent may be required to operate, even where those conditions are highly constrained. As the robot's environment varies, its visual processing must remain accurate. Robustness also leads to a trade-off – that between continuing to provide information as its quality deteriorates with adverse conditions, and ceasing to provide any information at all. In many cases it may be that inaccurate information is more harmful to an agent's planning system than no information at all, thus an important balance must be struck.

Finally, efficiency is also important. An agent must be able to process visual information as fast as it is made available in order to react most effectively to a dynamic environment. Thus, processing of a single image must complete in a small, usually fixed, amount of time. Algorithmic complexity is therefore constrained and implementations must be highly efficient. This

requirement introduces a third trade-off between processing time and the quality of information gained.

1.1.1 The RoboCup soccer domain

This report is based on theory and an implementation developed for the RoboCup four-legged league. RoboCup is an international project that aims to promote and foster intelligent robotics research through provision of a standard problem. RoboCup uses robot soccer (football) as a standard measure whereby teams of robots compete in real-time sensor processing, physical mobility, planning, multi-agent collaboration and associated technologies. RoboCup provides a number of soccer leagues, each with a different focus. Competitors in the four-legged league all use physically identical robots – Sony ERS-7 or ERS-210 entertainment robots – and compete solely on the basis of agent programming: no hardware modifications are permitted.

The four-legged league specifies a field layout, with important objects being of a specified size and colour. Nevertheless, there is considerable scope for interpretation and variation allowed by the specification of the environment, particularly with regard to lighting intensity and uniformity. Agents must be capable of performing under varying conditions, albeit with time allowed for detailed calibration procedures. Thus the four-legged league provides a constrained environment within which to develop agent technologies. Some of the techniques developed are also applicable to more general domains.

While the precise rules of the competition are not relevant to this report, some specifics of the environment are necessary to provide context. The four-legged league field is a green carpeted area of 6m × 4m as shown in Figure 1.1. The white side lines and goal lines describe a playing area of 540cm × 360cm inside this. Unlike in previous years there is no wall around the boundary lines so both the ball and robots are able to leave the field. The environment is undefined beyond the green carpet; the low wall shown in Figure 1.1 is one of many possibilities. The *goals* are 80cm wide, 30 cm high and 30cm deep, consisting of three pale blue or yellow coloured walls. Surrounding the field at a distance of 15cm from the sidelines and one-quarter of

Figure 1.1: The RoboCup four-legged league field. Note the coloured goals and localisation beacons. The outer wall is optional.

a field length from the goal lines are four *localisation beacons* (see Figure 1.2). The beacons are cylinders 40cm tall and 10cm in diameter. The top half of each beacon has a unique two-colour pattern consisting of a 10cm high ring of bright pink and a 10cm high ring of either pale blue or yellow, matching the colour of the goal at that end of the field. These beacons may be used for self-localisation by the robots. The *ball* (see Figure 1.3) is a hollow plastic sphere of radius 4.2cm and coloured bright orange. The object of the game is for each team of four robots to move the ball into one goal, while preventing the other team moving the ball into the other goal.

Figure 1.2: One of the four cylindrical beacons surrounding the field. Each has a unique combination of blue, yellow and pink coloured rings. These beacons can be used for self-localisation.

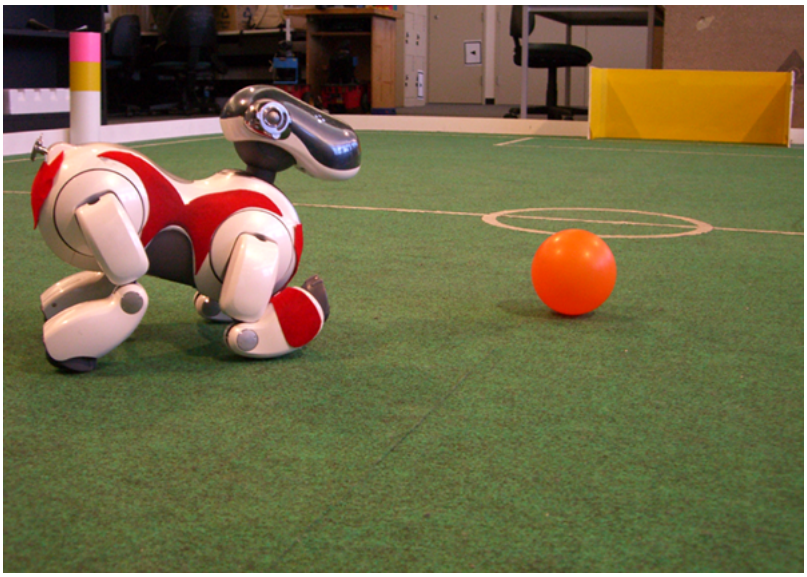


Figure 1.3: An AIBO ERS-7 wearing the red team uniform, with the four-legged league ball.

1.2 The state of the art

Within RoboCup leagues the most popular vision systems are currently based on colour classification and blobbing fundamentals. These systems have proved very effective in the past but have a number of significant shortcomings.

Colour classification seems a natural way to approach image processing. Humans respond strongly to colour, and digital images lend themselves to interpretation as an array of coloured points. However, the depth of colour available from even a standard digital camera (typically 24 bits of colour information per pixel) is far too high to be processed as is. Instead, pixels are classified as belonging to one of a small number of colour classes, corresponding to symbolic colours expected to occur in the agent's environment. In implementation this usually results in a look-up table or decision tree that quickly maps the detected pixel value to a symbolic colour. The generation of this table or decision procedure is an area of active research [15, 18, 24]. Typical approaches involve a supervised machine learning algorithm, where a human expert provides classification examples to a computer program, which generalises these to form the complete segmentation.

Regions of uniform colour may be recognised as projections of objects onto the image frame. Classified pixels are grouped, or blobbed, to form candidate sub-objects. Since objects are often of a uniform colour, or contain regions of uniform colour, neighbouring pixels of the same symbolic colour can be grouped together with a blobbing algorithm. These blobs form candidate objects to be tested for size, shape and other properties before final recognition as objects. Since the entire image is usually classified and blobbed, many blobs are formed that do not represent objects. These are discarded through a frequently complex system of rules and checks for the likelihood of the blob representing a valid object. Sets of blobs are then checked for mutual consistency.

These classification and blobbing systems have a number of shortcomings. Pixel values are classified as belonging to only one symbolic colour, since multiple classification leads to computationally expensive object recognition. However, many pixel values may legitimately

appear within objects of different colour, particularly under changes in lighting conditions. A finely tuned segmentation is necessary to allow accurate object recognition, but this leads to dependence upon the specific environmental conditions under which the segmentation was created. Any variation in lighting leads to a shift of the colour-space and the accuracy of colour segmentation deteriorates. Blobbing relies on regions of uniform colour, so any degradation in colour classification immediately affects the accuracy and robustness of object recognition.

There have been a number of modifications attempted to address some of these shortcomings, discussed in section 2.4. Dynamic classification approaches, where the classification of a pixel value may change over time, is one such modification. Approaches based on multiple colour segmentations or relative classification have had some success, but such methods rely on multiple fine calibrations or overly simplified representations.

A new approach is needed that offers robustness to environmental variations without relying on finely tuned, environment-specific classifications. Since it is a focus on colour that leads to many of these shortcomings, it would appear an alternative method of processing image data is required.

1.3 Sub-sampling images for colour relationships

In contrast to a purely colour-based approach this report presents a real-time robot image processing system for the RoboCup four-legged league based on minimal sampling of the image frame. Rather than process each image in its entirety this approach makes an intelligent estimate of the information content of regions in the image and samples those areas likely to be of importance. Rather than recognising candidate objects in the first pass, and subsequently filtering those deemed unlikely to be valid, features are detected in the sampled areas of the image and these are combined to form objects at a higher level of processing. Instead of relying on brittle colour segmentations, this approach focusses on the relationships between neighbouring pixels, which remain more constant under variations in lighting. Thus this approach aims to

extract reliable information from images without processing the entire image, and with a reduced reliance on statically classified colours.

This implementation aims to address the three major requirements for a robotic vision system identified above: accuracy, robustness and efficiency. The accuracy of sub-sampling is comparable to that of blobbing systems since most invalid object candidates are implicitly rejected. Only regions of an image likely to contain a particular feature are sampled for it, reducing the number of invalid objects it is possible to recognise falsely.

Robustness, particularly to changes in lighting conditions, is greatly improved through a reduced reliance on static colour segmentation for object recognition. As lighting conditions change and the colour content of images shifts, the colour relationship between pixels changes far less than the absolute pixel values. Detecting these relationships rather than relying on a static colour classification allows consistent feature detection even as the ambient lighting varies.

Efficiency is improved through sampling only high-information areas of each image. Regions of an image with little information value are sampled sparsely, while regions with a high information content are sampled much more densely. A dynamic element to sampling allows detection of objects even when an initial sampling fails to provide sufficient information. However, a somewhat random-access approach to image sampling has negative implications for hardware performance.

This approach also presents solutions to, or implicitly avoids, some of the problems identified with purely colour-based approaches. A colour segmentation need not be so tightly defined and calibration time is reduced; the calibrated colour relationship tests are environment independent; complex series of manually coded object validity tests are minimised; redundant information is avoided as only information-rich areas are sampled densely; and object recognition is robust to falsely detected features and unexpected background information.

These results represent important progress in robot vision towards colour invariance and reliability under variable conditions. Although it remains the case that robot soccer, and RoboCup in particular, is a tightly defined domain the techniques presented here are applicable to other

domains in which similar assumptions about the appearance of objects may be made and processing is constrained by limited resources.

1.4 Report overview

In the following chapter of this report the problem of robot vision systems is discussed in more detail and a number of existing approaches are introduced. The difficulty of evaluating such systems is discussed and a number of evaluation frameworks are presented. In Chapter 3 a sub-sampling image processing system is presented for the RoboCup four-legged league and a detailed implementation overview is given. In Chapter 4 results are presented and a number of design decisions in the implementation are justified. In conclusion, Chapter 5 summarises the progress embodied by this work and its implications for the development of future robot image processing systems.

Chapter 2

Background

Robot vision refers to the digital processing of images, often from a real-time digital video stream, in order to extract information about a robot's environment and situation. Robot vision systems, as a subset of the more general computer vision systems [11], attempt to detect useful features or recognize objects in real time under a range of environmental conditions, and from these features make inferences about the state of the robot and its environment. Such inferences are frequently based on implicit geometrical and physical knowledge of the environment and the robot's sensor model [23].

For many robots, vision is the most information rich, and hence most important, sense. Certainly in terms of raw data rate, a fairly ordinary digital camera can provide much more data[1] than a high quality microphone[2], sonar or laser range finder[3], or touch and pressure sensors[4]. The information content of camera images is correspondingly high, so complex and precise processing is required to extract useful information from the raw data stream.

This image processing is often a fragile and finely tuned process. Given the detail available in even a single image there are myriad variations on any given scene, but a good vision system

[1]The ERS-7 camera has 208*160 pixels * 24 bpp * 25 fps = 2,496,000 bytes/sec
[2]44.1 KHz * 16 bits = 88,200 bytes/sec
[3]Typically < 100,000 bytes/sec
[4]< 1000 bytes/sec

is required to infer accurate and consistent information from nearly all of these variations. As environmental conditions such as lighting intensity or colour balance vary, a vision system must robustly present a constant representation to higher level processes. Further, robots frequently have highly limited processing power available to them – constrained by limitations on size, power consumption and cost – so robot vision systems must perform their processing using minimum computing power. This presents a trade-off between processing an image thoroughly and doing so fast enough to avoid discarding any information-bearing data.

2.1 Accurate, robust and efficient robot vision

Robot vision must be accurate. While this may seem obvious there are a number of different measures of accuracy, and the effects of any inaccuracy vary depending on the situation and type of error that occurs. A robot's environment, and perception of this environment, are continually changing due to external influences, the robot's own actions and random distortion from physical sensors. In spite of this, a robot's vision system must provide stable and accurate information to higher level processes.

Robot vision must be accurate in the sense that objects that lie within the visual frame must be reliably detected; *false negative* errors should be minimized. Conversely, objects that do not appear within the frame must not falsely be detected to be present; *false positive* errors should also be minimised. The effects of failing to detect valid objects and of falsely detecting invalid objects depend on the higher behavioural processes.

A falsely detected object will most likely confuse localisation and behavioural functions in the agent. Within the robot soccer domain this may be expressed overtly, such as chasing a ball that isn't there, or more subtly, such as becoming mislocalised and moving to the wrong position. Failing to detect an object that should be visible is likely to have more subtle effects. An agent might not become mislocalised merely due to failing to perceive some landmarks, but its localisation confidence and accuracy will be lower than it might have been otherwise.

Similarly, a RoboCup robot that sometimes fails to see the ball will not chase the wrong thing, but will react more erratically than it might have, had its vision been more accurate. If higher level functions rely heavily on the assumption that if an object cannot be detected then it is not within the vision frame, the confusion arising from such errors will be magnified. In practice, however, there are many reasons an object might not be detected, such as it being occluded by another object, so such assumptions have complex side-effects regardless of the accuracy of the vision system.

These two types of error present an important trade-off in the balance between false positive and false negative errors. As a system is tuned to reduce false positives, it is usually a consequence that the number of false negative errors will increase. As the conditions for recognising an object become more specific, the number of valid objects that fail to meet these conditions will increase. Similarly, if a system is tuned to reduce false negatives, there will be a corresponding increase in the number of false positives. As conditions for recognising an object relax to accommodate more and more extreme cases, the occurrence of falsely detected objects will increase. This trade-off is common to many systems that process digitised real-world input, such as biometric identification systems [9] and speech and language recognition [8] software.

A second dimension to visual accuracy is the accuracy of perceived properties of an object. Under the assumption that an object is correctly determined to be present in a given image, the accuracy of such attributes as size, distance and orientation is important. Inaccuracies here are likely to take the form of a probability distribution and vary with the noise present in an image.

As well as being accurate, robot vision must be robust. While the concepts are closely related, robustness refers to consistent accuracy under varying conditions, and graceful failure when accurate detection is not possible. A robust vision system should be able to detect objects reliably, not only under ideal conditions, but also under changing lighting intensity and colour balance; when fully or partially shadowed; with specular and other reflections; with uniform or non-uniform backgrounds of varying colour; when blurred or distorted by the object's or agent's motion; when partially occluded and under many other uncommon and unpredictable conditions. When conditions render it unlikely that an object can be accurately detected a

robust visual system should fail to detect it, or relay information that the object was seen but is highly unreliable, rather than detect the presence of the object with properties grossly differing from the ideal.

This presents another trade-off: the balance between detecting objects with unreliable properties and not detecting them at all. As the data from which an object is recognised becomes more and more extreme, the properties inferred on the object vary more and more from the ideal. If a system is tuned to detect objects even with the most unreliable data, then the properties associated with these objects will become similarly unreliable. Conversely if a system requires high confidence in the accuracy of properties in order to detect an object, then much information will be lost from objects rejected due to unreliable data.

In addition to accuracy and robustness, robot vision must strive for efficiency. Image processing systems for robots that react to a dynamic world in real-time must execute in minimal time. In many domains, including robot soccer, the entire agent program must complete in a fixed time slice determined by the frame rate of the robot's camera. In order to avoid discarding vision frames each one must be processed before the next arrives. Such bounds place tight constraints on the amount of compute power available to process a single frame, which in turn constrains the feasible algorithmic complexity.

The requirement of efficiency produces yet another trade-off: that between the amount of compute power consumed by image processing and the accuracy and robustness of that processing. As more and more processor time is devoted to vision, the accuracy and robustness of processing can improve and more information can be extracted from the image. However, this is at the expense of compute power for the other modules comprising the agent, and usually has a hard upper bound. As a vision system is optimised to use fewer and fewer processor cycles, the amount of possible work decreases. Beyond some point this efficiency necessarily comes at the expense of accuracy and robustness of the system.

2.2 Common challenges

There are a number of commonly occurring problems that most robot vision systems must overcome. These complexities challenge a system's accuracy and reliability, and failing to address them leads to implementation of systems with poor robustness.

2.2.1 Camera variation

Cameras are imperfect instruments and both systematic and random distortion are common. A robust vision system must detect features consistently in spite of this distortion. A case in point are the noticeable distortions of the camera (a Fujitsu MB86S03 with CMOS sensor) on the Sony AIBO ERS-7 robot used in the implementation of the work presented below. The first is a chromatic distortion that appears as a ring around the centre of the frame, where pixel colours are darkened and shifted towards blue. This chromatic distortion is quite overt and of a similar geometry and magnitude between individual robots. A vision system that relied on colour or colour-relationships and did not compensate for this distortion might fail to accurately detect features affected by this ring. Significant progress has been made quantifying [4] and correcting [27] for this error on the ERS-7.

A second systematic distortion of this camera is a colour shift between individual sensors. Two robots observing essentially the same scene experience a constant chromatic difference in their observed images. This colour shift is not easily noticed by a human, but in the presence of finely tuned colour classification methods it results in a noticeable difference to the classified colours. Compensating for this systematic difference is a challenge that must be overcome by a vision system intended to be executed on a number of individual robots; a calibration for one robot would not necessarily be appropriate for another [13].

A third systematic distortion of the camera is a geometric distortion close to the edges of the frame caused by the lens assembly. This distortion causes mathematical projection of camera pixels onto the environment to become inaccurate around the image edges. A vision system

that relied on pixel-accurate projection from the camera frame that did not compensate for this distortion would experience differing results, depending on where in the image frame a given feature appeared. This distortion has been quantified for the ERS-7 and ERS-210 in [18].

Cameras, like all other digital sensors, capture and discretise a continuous real-world phenomenon. As such they are subject to random variation. A robot that is stationary in a seemingly static environment will never see the same image twice. Even while there are no apparent changes in its situation, the light incident on the camera varies continuously and randomly. [4] quantified this variation under ideal conditions for the AIBO ERS-210; they found standard deviations of 0.7523, 0.9524 and 1.6318 for the Y, U and V components (see section 3.2.1) of pixels in the image respectively from a sample of six images of solid colour, although this deviation was far from uniform over the image area. In effect, the least significant bits of each channel are randomised and should not be relied upon for robust vision. Robustness to this random noise is a problem encountered by many vision systems.

2.2.2 External variation

As well as the sensor itself inducing variations to the input, the actions of a robot, other agents and external influences also give rise to significant variation in the sensed images. One common such variation for mobile robots is motion blur. When a robot or its surroundings move quickly, images captured by its camera become blurred and distorted. Blurring softens the edges visible in an image and reduces definition, particularly of distant objects. Vision systems that rely on detecting sharp edges are likely to perform poorly on blurred images. Fast motion of the camera-bearing part of the robot may also introduce a geometric distortion to the image. Many cameras use a raster scan when capturing images, which leads to a small time difference between when the top and bottom of the image are captured. This distortion is visible as a skewing of the image to one side or the other, distorting lines and shapes. Vision systems that rely on consistent geometry over images may loose accuracy when an image is skewed.

Changes in lighting conditions also lead to significant variations in images processed by a vision system, and a robust system must provide consistent information to higher levels of processing even so. Lighting variations fall into two groups: local variations and external variations.

Local lighting variations are those changes in lighting in a small region of a robot's environment that are independent of the light source. The most prominent of these are shadows and reflections. A shadow necessarily changes the light that falls on the shadowed area, reducing intensity and directness. For example, a ball sitting on a flat surface casts a shadow both on the surface and itself. The lower part of the ball is shadowed by the upper part and will appear to have a different hue and intensity. A robust colour-dependent vision system must be able to recognise that these two parts are of the same object, despite differences in appearance. Similarly, reflections alter the hue and intensity of light incident on surrounding objects. A brightly coloured object will reflect some of that colour onto surrounding surfaces, thereby changing their appearance. A robust colour-based vision system should not be confused by this reflected colour. Specular reflections also alter the appearance of objects, appearing as high-intensity spots among otherwise uniform colour.

External variations are changes in the light source and its incidence on the environment. No light source or combination of light sources can provide a perfectly uniform illumination, and variations will be observed when a robot moves between different regions of its environment. It may be the case that some areas are lighter or darker than others, or that natural light falls more strongly on one area, and artificial light more strongly on another. These variations mean that an object observed in one area will appear different when in another area or when observed from a different perspective. These myriad variations in local and external lighting conditions present a difficult challenge to implementors of a robust robot vision system.

2.2.3 Calibration

Calibrating a robot vision system for a particular environment takes time. Apart from the camera-dependent calibration, discussed in [13], it is particularly the case that colour-dependent

vision systems require extensive calibration of colour classification methods to be effective. There are some $2^{24} = 16,777,216$ possible values for each pixel and, although a vast majority of them may be irrelevant to a particular domain, an effective calibration must address all those likely to carry useful information. In particular, a calibration must be sure to address non-ideal conditions that may occur during execution such as shadows, reflections and other variations mentioned above. A calibration that failed to cater for these conditions would likely be ineffective when they arose.

Many systems use a machine learning approach to generalise the colour classification from a smaller set of training data provided by a human expert (see section 2.4.1). Over simple regions of the colour-space, where adjacent values are all classified to the same symbolic colour, this provides great benefits, saving time and improving robustness. However, it is at the complex boundaries and fine distinctions between colours that the challenge lies. Colour-based systems must rely on a very fine boundary between two similar colours to maximise accuracy in the face of reflections and shadows. This boundary must be precisely determined by a human expert as it is frequently too complex to be generalised from a small number of samples by any automated approach.

Thus the calibration process for any particular environment is time consuming. The accuracy of the system depends on the skill and patience of a human expert in defining appropriate boundaries. This fine calibration leads to a brittle system, where small changes in conditions disrupt colour recognition. A system that is calibrated as far as the least significant bits of each channel becomes dependent on the random variations in the camera sensor. This over-fitting leads to dependence on the precise conditions and calibration data used during the calibration process. Further, fine calibrations are not portable to different environments and the entire process must be repeated if there is a change in conditions.

2.3 Methods of evaluation

Evaluation methods for robot vision systems present a difficult problem of their own. There are a large number of possible images for a camera with more than a trivial resolution and colour depth, and it is intractable to test a system with any significant proportion of these. In addition, the most common reference point for interpretation of an image is a human's interpretation of the same image. Thus any evaluation technique that relies on simple testing of visual accuracy has a certain qualitative aspect. Three methods of evaluation were utilised in the development of the system presented in Chapter 3.

Firstly, in our domain of robot soccer, the accuracy and robustness of a vision system reflects strongly in the performance of a team of robots in a competitive soccer match. In a very limited sense this is the most valuable method of evaluation. The goal of the vision system is to have robots play the best soccer and a vision system that results in a team consistently winning matches is better, in some sense, than a system that does not – all other things being equal. However, all other things are never equal so, while being the most important test of a system's quality, this test is also the most subject to random variation, noise and external influences. The performance of a team also depends highly upon the performance of the opposing team. Thus unless the opposing team remains constant it is difficult to compare the performance of systems across matches. Throughout the course of a match the environment both on and off the field changes subtly, and not so subtly, as robots move and interact and as external conditions change. Further, this method of evaluation is highly non-repeatable; it is impossible to substitute an alternative vision system and have exactly the same game play out with the exception of changes directly related to vision. Indeed, it is impossible to play the same game twice even when no code or conditions are explicitly changed.

Nevertheless, evaluation by playing matches remains an important measure of progress. It is the case that a team that plays well and wins consistently against varied opponents has a "good" vision system. However, it is not the case that a team which performs poorly necessarily has a poor vision system; there are many other factors that may result in poor play. However, if

otherwise identical code is used in both teams in a number of competitive matches the influence of different vision systems may be observable in qualitative terms.

At a slightly finer level, behavioural evaluation of a single robot agent presents another important method of evaluation. While still in qualitative terms, the behaviour of a single agent provides information about the performance of its vision system in particular circumstances, and the removal of teammates and opposition removes much of the random variation and chance from the test. For example, an agent's behaviour may provide clear indication of whether or not it can see a given object. It may be possible to display informative indicators in the form of LEDs, or such information might be accessible via a remote log-in session or other data stream. Thus the quality of a single agent's visual information may be subjectively assessed. Alternatively, two independent agents might be active simultaneously, and the behaviour and data streams from each compared. Although each agent is processing different input, over time any significant systematic differences in visual processing will become apparent.

Single agent tests bear some semblance of repeatability. In the absence of active teammates and hostile opponents situations can be constructed and the performance of agents evaluated over a number of similar trials. In order to test robustness these set-ups should come as close as possible to emulating the match environment; for example, robots wearing both red and blue uniforms should be placed on the field but remain stationary. In particular, single agent tests are useful for comparing modifications to a vision system, and were used extensively in the implementation of the system outlined in Chapter 3.

The RoboCup four-legged league also presents a number of technical challenges that are useful in evaluating an agent's vision system [28]. The *Almost SLAM Challenge*, while primarily a test of an agent's localisation and odometry systems, also requires robust image processing. The Almost SLAM Challenge requires the ability to recognise objects similar to but differing from the standard four-legged league landmarks (with tight constraints on colour, size and uniformity) alongside the standard objects, and extract sufficient information for self-localisation from the new objects alone. A robust and somewhat flexible vision system aids performance in this challenge.

RoboCup also presents the *Variable Lighting Challenge* which explicitly tests an agent's vision system's robustness to changes in lighting conditions over time. In this challenge an agent must consistently recognise the standard RoboCup objects while the field lighting changes in different ways. While still heavily dependent on higher level behaviours this challenge tests robustness of image processing systems to shifts in lighting intensity, colour temperature and dynamic shadows. For an agent to perform well in this challenge it necessarily requires a highly robust vision system.

At yet a finer level an agent's image processing system may be compiled and executed in isolation on a standard PC ("offline"), where its performance for particular images or image sets may be qualitatively and quantitatively evaluated. This allows direct observation of the performance of a system in precise circumstances, and in many cases provides insight as to why a system behaves as it does. A log file of the image data as sensed by the camera may be captured then "played back" and processed offline by the agent's image processing system. This allows the performance and internals of the processing system to be visualised and examined in detail.

Unlike the two previous methods, offline evaluation is fully repeatable and allows multiple image processing systems to process exactly the same data and the results to be compared. Further, an evaluation tool may be constructed that allows a human to process this same image data and provide (subjectively) "correct" information based on the objects recognised in the image. Although this evaluation method is subject to a human's interpretation of the image, it allows direct comparison of vision systems, and comparison against a subjective ideal interpretation. Alongside single agent evaluation this method is effective at comparing modifications to a vision system and was used extensively in the development of the system presented below.

These three evaluation methods present trade-offs between their importance, accuracy and ease of administration. While evaluation through playing soccer matches remains the most important method of evaluation for RoboCup, it is also the method that provides the least detailed information about image processing specifics, and is most subject to noise and random

variation. It is, however, a relatively easy test to administer and the evaluation method remains consistent across agents with highly differing vision systems and software architectures.

Evaluation through observation of a single agent's behaviour provides much more specific information about the performance of its vision system. Although it is ultimately the in-game performance that is the most important, single agent evaluation can provide important qualitative evaluation of specific aspects of its image processing system. For example, the robustness of ball recognition may be tested in isolation by observation of informative LEDs or another information stream. Single agent tests are easily administered but depend somewhat on the agent's other systems and behaviour. The vision systems of two entirely different agents are not directly comparable with this method; this is only possible if the agents' programs are identical except for the vision system.

Offline evaluation provides by far the most accurate and specific information about the performance of image processing. This method of evaluation is highly repeatable and allows for direct observation of the performance of the system on individual images. However, offline evaluation is simultaneously both the easiest and hardest method of evaluation to administer. Offline evaluation tests the image processing system in isolation, so it is easily administered repeatably to similar vision systems, or evolutions of the same system. However, offline evaluation is highly dependent on the implementation internals of the system, so unlike the previous evaluation methods it is very difficult to administer across radically different systems.

2.4 Review of existing work

Systems that implement robot vision are tightly constrained, particularly by the requirement to be real-time, and it is infeasible to implement many general computer vision techniques under these conditions. In this section recent work in the RoboCup four-legged league and related domains will be discussed and evaluated, and a number of challenges yet to be adequately addressed will be identified.

2.4.1 Colour segmentation

Given the definition of the RoboCup domain in terms of objects of a uniform, specified colour it is natural to base a RoboCup vision system on colour detection and classification. The four-legged league defines eight standard colours for objects on the field: a green field with white lines; an orange ball; pale blue, yellow and pink goals and landmarks; and red and dark blue robot uniforms. Each pixel value that might appear in an image may be classified as belonging to one of these symbolic colour classes (or other classes such as grey, or an unrecognized value), thus breaking the colour-space into segments. A number of approaches have been taken to perform this segmentation, and it remains an active area of research.

The first class of approaches to colour segmentation attempt to generate an efficient decision procedure for membership of a particular colour class. A simple approach such as manual thresholding can be very efficient [1] but has a number of shortcomings, not least that it requires all members of a colour class to lie within a cube in colour-space, and classifies every value within the cube to that colour. As the cube is expanded to include more of the outlying values caused by reflections and shadows it necessarily includes more values that should not belong to that symbolic colour. Quadric surfaces can more accurately capture the shape of colour-space segments ([29] used ellipsoids approximated by least-squares vectors), but still suffer from this imprecision.

A more complex decision tree allows greater precision but is usually too complex to create manually. Supervised learning approaches have been relatively successful in forming these complex trees. Quinlan's C4.5 decision tree learning algorithm has been used in both the mid-size and four-legged leagues for colour segmentation [2, 27]. The resulting decision tree may by executed at run-time or used to create a look-up table. However, such methods still implicitly assume that segment boundaries lie along axis-parallel planes.

The second class of colour segmentation approaches generate a look-up table which is loaded into memory at run-time, and classification simply involves indexing into this table. Rather than be restricted to a decision tree these methods allow segments to be arbitrarily complex regions

in colour-space. Working from a number of hand-labelled example images, a nearest neighbour algorithm [6] has each unlabelled pixel take on the classification of its nearest (Manhattan) explicitly classified pixel within some fixed radius. This method requires that similar numbers of each symbolic colour are present in the example images, since an abundance of any particular classification will weight the nearest neighbour decision towards that colour. This approach can be generalised with a geometric [24] or exponential [18] decay, allowing multiple example pixels to influence the classification of each unclassified value. While these approaches are far more robust to errors in the example values they still require a careful balance of the number of examples of each symbolic colour provided. None of these methods allow real-time feedback of the results of changes to the example classifications provided; the implementations take minutes to run.

These segmentation methods all place pixel values into exactly one colour class. This is not always appropriate and in fact there are many pixel values that can legitimately occur in two or more different coloured objects. As an example, the orange of the ball can appear red when shadowed, and similarly the red team uniform can appear orange in bright light. The pixel values around the red/orange boundary in colour-space might appear as part of either the ball or a red uniform. This problem is most evident when lighting conditions change. The usual solution to this problem is to finely balance the number of values classified into each class in order to ensure maximum average performance.

This solution is clearly imperfect and a number of techniques have been trialled to avoid the problem. One possible method is to classify a number of values with high confidence – those which rarely or never appear in other objects – and mark remaining classified values as low confidence classifications ("maybe-colours" in [6]). This does not directly solve the problem, since each value must still belong to at most one colour class, but provides more information to the higher level object recognition systems. A more flexible extension would allow values to belong to more than one colour segment. However, it becomes computationally expensive to perform feature and object recognition when values may simultaneously belong to more than one class, so existing methods utilise a dynamic classification, whereby the classification of a pixel value may change over time.

[16] used two static colour tables – one for normal conditions and one for shadowed images – and chose to use the shadowed table when it was known the camera was directed steeply downwards. This was extended to five colour tables for use in the RoboCup Variable Lighting Challenge. The average value of the green field was used to select which of five colour classifications to use for the next frame, each classification tuned for a particular ambient lighting intensity. [21] used three colour tables with a more complex selector based on a KL divergence measure. Clearly these approaches require construction of many colour tables, a time consuming and non-scalable approach. [12] also used the average value of a selection of "green" pixels but had no static classification – other colours were defined relative to the reference green, and colour segmentation was dynamic. While this approach is promising, it required simplification of colour segments to simple threshold cubes. [27] experimented with dynamic histogram equalisation to compensate for fluctuating lighting but found it generated an unacceptable amount of noise in the chroma channels.

A promising alternative to static tables is to perform seeded region growing over the image [25]. The initial segmentation is performed with simple thresholds but restricted to values of high confidence. The image region that this high-confidence classification creates is then grown with a blobbing algorithm until a significant discontinuity is encountered. This approach exploits spatial locality as it implicitly seeks edges as the regions are grown but suffers from dependence on a very finely tuned homogeneity criterion (i.e., edge threshold).

The conclusion to be drawn from all of these attempts is that colour segmentation is a difficult problem. Local variations, temporal variation, complexity of segments and overlapping classifications all thwart creation of a perfect static classification, and dynamic classifications have so far been simplistic or drawn unacceptable side effects. The more complex the classification becomes, the more training data is required, the more tightly it fits the training data rather than general conditions, and the more skill is required on the part of a human expert. The solution lies not in further improving colour classification methods, but in moving away from symbolic colour segmentation towards illumination invariant vision.

2.4.2 Blobbing

Once a colour segmentation is defined the entire image, or parts of it, may be classified according to symbolic colour classes. Regions of the same symbolic colour form candidates for objects, which must then be tested for size, shape, orientation and other expected properties.

Connected regions of the same colour class are usually formed with some variant of a blobbing algorithm. Standard four- or eight-way blobbing algorithms are computationally expensive, requiring multiple accesses to each classified pixel. Instead, an efficient approach is to run-length encode the image as the colour classification is applied. Connected segments are then formed from horizontal runs that overlap vertically [13]. The efficiency of this varies with the complexity of the image.

Some blobs must be associated together to form objects, such as the two colours forming the localisation beacons surrounding the field. Multiple blobs of the one colour must be disambiguated into one or several objects. These connected regions ("blobs") must then pass a number of tests to be recognised as valid objects. For example, a region of orange classified pixels is unlikely to be the ball if it is too small, or resides high in the image when the camera is directed upwards. Similarly these regions must be checked for mutual consistency: the ball is unlikely to appear above a goal or beacon and both goals cannot simultaneously be in view. These complex tests have previously been hand-coded (although [15] had success with a supervised learning approach) and are a source of much complexity and confusion in the vision algorithms.

The desired properties of objects are then inferred from the size, shape and orientation of the blobs or their bounding boxes. Properties such as distance from the camera may be determined from the known physical size of the objects or a projected distance based on the robot's stance. Aspect ratio and orientation may give information as to the relative location of the object.

The blobbing approach requires expensive processing of entire images followed by many complex, hand-crafted tests to determine the validity of each detected blob. A large number of

these checks are required to achieve accurate object recognition, and the checks must be executed for each candidate blob in an image. In order to robustly handle significant variations in lighting, more advanced feature detection and validity tests are required. These are computationally expensive in blob-based systems.

Blob-based object recognition lacks robustness since it relies on areas of uniform colour to be formed from a (usually static) colour-space segmentation. When lighting conditions differ from those under which colour segmentation was performed, regions of uniform symbolic colour break up into multiple regions of nearby symbolic colours. Lighting variations increase classified image complexity, leading to increased processing time and less accurate information.

2.4.3 Sub-sampling approaches

Sub-sampled approaches are a very recent development in RoboCup vision. [18] sampled three grids of scan lines over the colour-segmented image, each optimised for different expected objects. Boundaries were detected between regions of segmented colour lying on the scan lines. The magnitude of the Y gradient at these features was used as a filter for spurious boundaries. While still dependent on a static colour segmentation this implementation was highly efficient.

Along with the dynamic colour segmentation noted above, [12] recognised that colour relationships are invariant under linear shifts in lighting, and detected edges as *contrast patterns* – ternary 3-tuples representing the direction of change (up, down, none) in the three image channels. Particular tuples characterise one or typically more possible colour transitions. The classified colour of pixels surrounding the transition was used to resolve ambiguities. Both of these approaches provided inspiration for the procedures presented in Chapter 3.

Scan lines over the colour-segmented image have also been used for object detection in [24] and were used for field line detection only in [22, 26], where blobbing was recognised as a computationally expensive method for field line and border detection.

2.5 Conclusions and approach

Robot vision is a complex task, requiring real-time processing of a large amount of data with highly constrained resources. Accurate and robust robot vision involves complex trade-offs in order to provide reliable information in spite of random and systematic sensor errors and variations in lighting conditions.

Many previous systems have relied on colour segmentation, recognising objects as regions of uniform colour. A range of methods for generating colour classification tables or decision procedures have been developed; however, static colour segmentations are brittle and do not cater well for variations in lighting, regardless of the classification technique. Dynamic colour classifications show promise, but existing implementations lack scalability and flexibility. In addition, blobbing methods from even well-classified images are error prone and expensive, processing every pixel in each image and relying on complex sets of manually constructed tests for validation.

Sub-sampled approaches offer gains in both efficiency and robustness. Efficiency may be improved by processing only a subset of image pixels, and robustness improved by turning to local colour relationships rather than global segmentations. This report takes such methods a step further.

Chapter 3

Sub-sampled object recognition

In this chapter a robot vision system for the RoboCup four-legged league is presented. The system is based on feature detection through colour-gradient detection from a minimal sampling of each image, and object recognition from these relatively sparse features. This system aims to achieve the goals of being accurate, efficient and robust in the RoboCup four-legged league domain and addresses some of the problems associated with purely colour-based approaches. This system was successfully utilised by the UNSW/NICTA RoboCup four-legged league team, *rUNSWift*, at RoboCup 2005.

3.1 Theory

The approach presented below is based on two major theoretical differences from existing systems. Firstly, this approach moves away from absolute colour classification techniques and focusses instead on the relationships between neighbouring pixels. This reduces the dependency of the system on static colour classifications and precisely controlled lighting conditions. Secondly, this approach ceases to process the entirety of each image and instead samples only areas

of the image likely to be of high informational value. This aids efficiency and reduces false positive errors caused by unexpected regions of colour.

3.1.1 Colour relationships

Static colour segmentations are brittle and depend highly upon the exact lighting conditions under which the segmentation is made. While some work has been done implementing dynamic colour classification these approaches have lacked generality or flexibility. Rather than attempt to change the classification at run-time this approach instead relies on the relationship between colours, which approaches invariance under changes in lighting.

As lighting conditions change the absolute colour values of pixels projected from a certain object change considerably. However, the colour-space differences between pixels representing distinct real-world colours change far less. While pixel values change under varying light, the values for all pixels will change in approximately the same way. As light intensity falls the intensity of pixels representing all colours falls correspondingly. Thus, while each colour may no longer be recognisable under a static classification, the difference between two colours remains discernible. Similarly, as the colour of ambient light changes the observed colours of all objects will shift correspondingly, but the difference between colours will change by a far smaller amount.

As evidence, observe Figures 3.1 and 3.2. The left-hand column comprises images as detected by the ERS-7 camera, with ambient light intensity decreasing down the page. The centre column displays a static colour classification of the images in the left-hand column under a calibration tuned for bright lighting. Note that as the ambient light intensity is reduced the images become successively darker and the colours recognised by a static classification become increasingly less accurate. The images in the right-hand column show a graphic representation of the colour-space difference between each pixel and its vertically adjacent neighbour above. Pixel values are calculated as given in Equation 3.1 (refer to section 3.2.1 for details of the colour-space). This difference precisely picks out the boundaries between the orange ball, white lines and green background. The white wall around the edge of the field is also apparent. The upper and lower

boundaries of each object appear as different colours; the lower boundaries represent transitions away from green towards orange or white, and the upper boundaries represent corresponding transitions back into green. Searching for particular boundaries in the original images reduces to searching for these particular "colours" over the difference images. Note that these images remain relatively stable as the lighting intensity falls, in contrast to the rapid deterioration of the colour-segmented images in the centre column.

$$\begin{pmatrix} Y_{x,y} \\ Cb_{x,y} \\ Cr_{x,y} \end{pmatrix} = \begin{pmatrix} 2\,|Y_{x,y} - Y_{x,y-1}| \\ 2(Cb_{x,y} - Cb_{x,y-1}) - 128 \\ 2(Cr_{x,y} - Cr_{x,y-1}) - 128 \end{pmatrix} \tag{3.1}$$

As the lighting intensity decreases the difference between each pixel and its neighbour remains more constant than the values of the individual pixels. Although the intensity of the difference falls slightly, the direction of difference (represented as colour) remains almost constant. While the relative pixel values do change as lighting conditions change, they do so less rapidly than the absolute pixel values. Note that this change in relative values is not exactly linear, depending on the particular characteristics of the light source. Thus while relative colour is more stable than absolute colour, it is not so stable that it remains constant under excessively wide lighting variations.

Detecting differences in neighbouring pixels is very similar to edge detection. Figure 3.3 shows that applying Roberts' operator [17] results in a similar image to that obtained by the single-pixel comparison. General edge detection methods such as Roberts' and Sobel operators are computationally too expensive to execute on every frame from the ERS-7. Instead, a simple one-dimensional gradient is calculated. A crucial difference between this method and using many standard edge detection algorithms is that the direction of the change in colour-space is explicitly captured, and "edges" can be easily classified by the colour difference they represent. Rather than relying only on slope intensity for feature recognition, the direction of change in colour-space provides crucial information about what the transition represents.

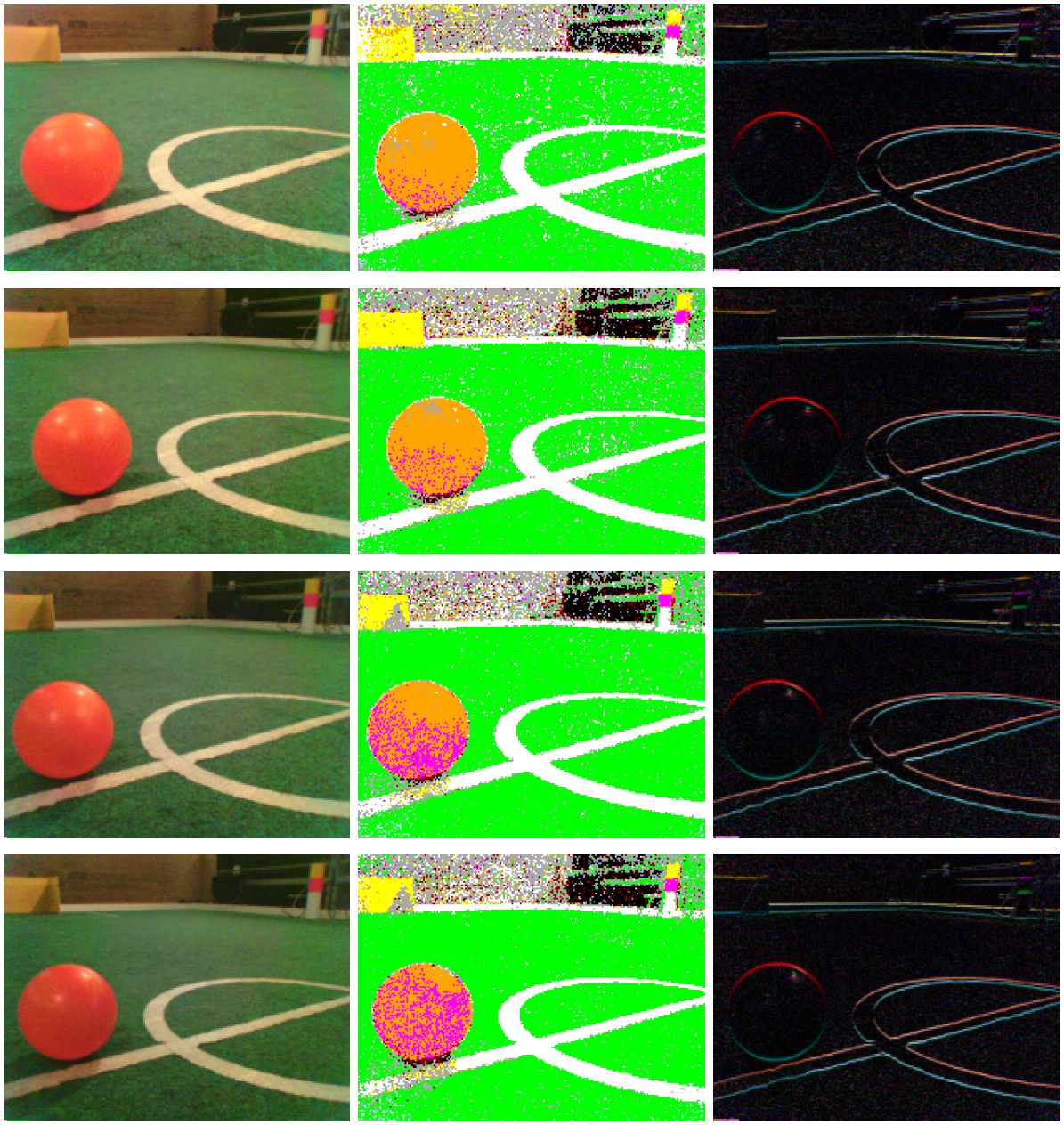


Figure 3.1: A scene under progressively darker lighting conditions. The left-hand column shows images as detected by the ERS-7 camera (after correction for chromatic ring distortion). The centre column shows the result of colour segmentation with a colour table optimised for bright lighting. The right-hand column shows the result of applying Equation 3.1 to the images at the left, subtracting from each pixel the value of the pixel immediately above it. Note how this remains stable as the lighting intensity falls, while the colour segmented images deteriorate.

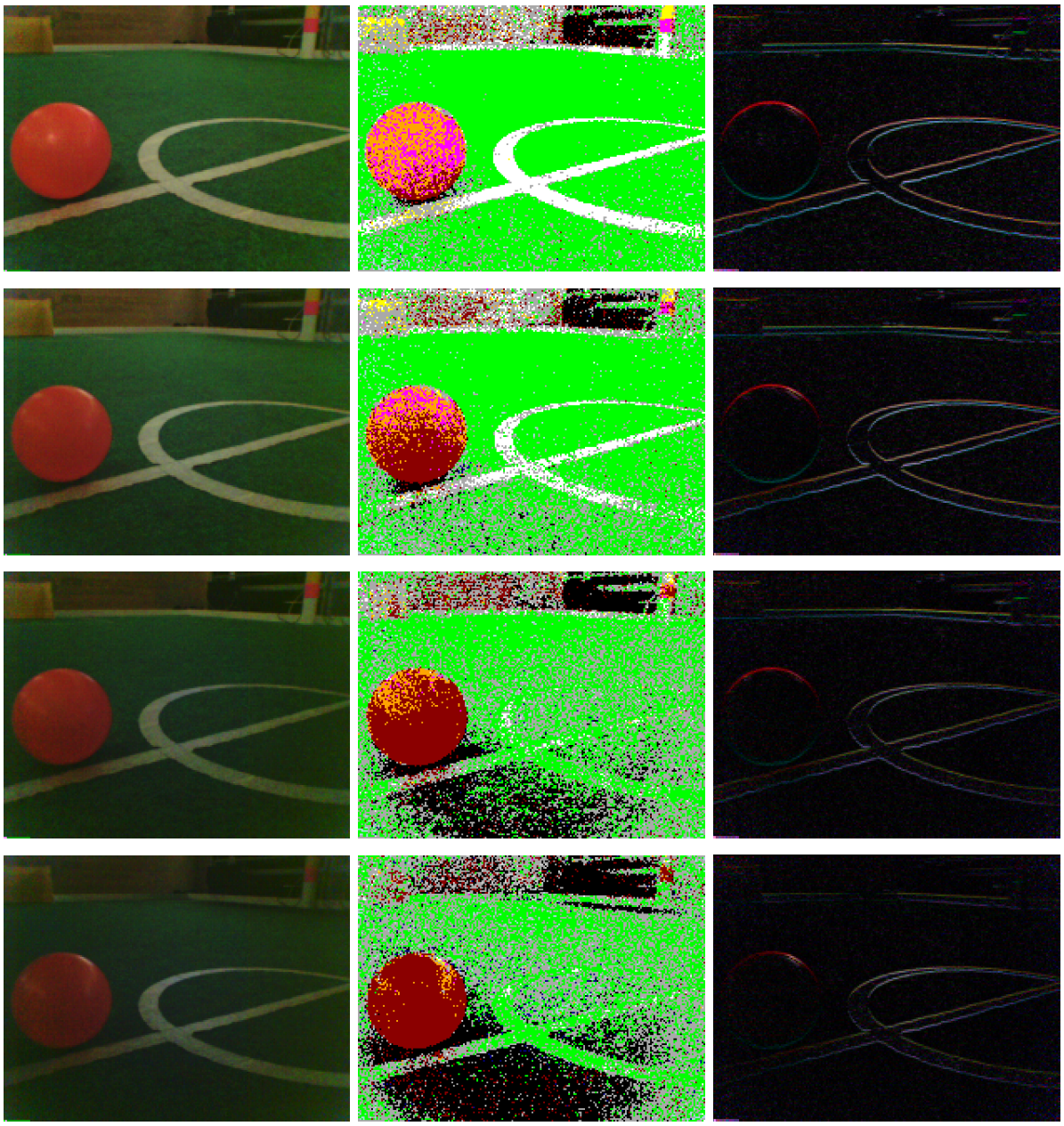
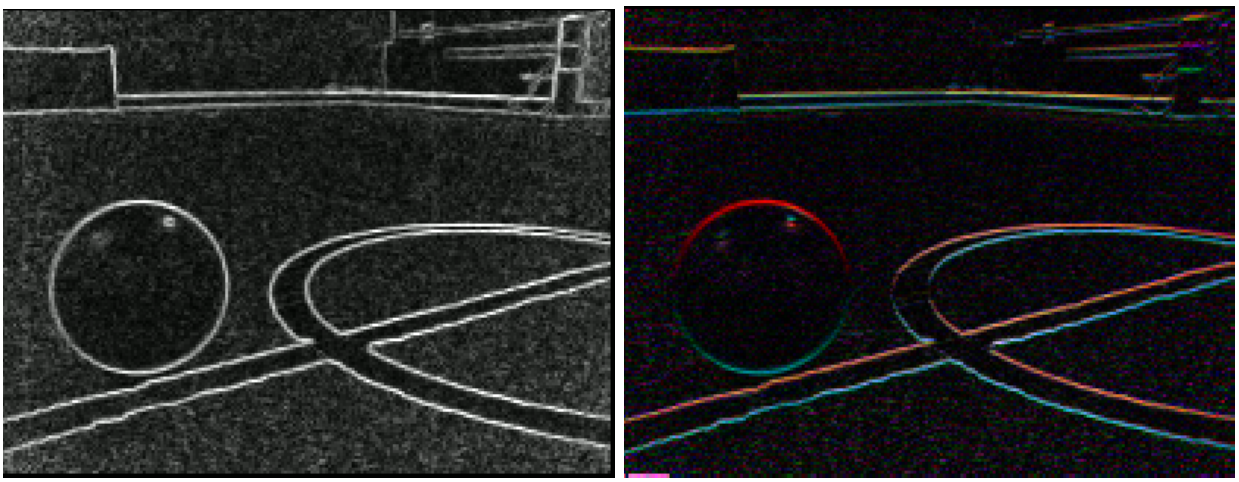


Figure 3.2: The same scene as in Figure 3.1 under progressively darker lighting conditions. The implementation described below successfully detected the ball in all but the darkest image.



(a) (b)

Figure 3.3: (a) Roberts' operator applied to one of the images in Figure 3.1. The aggregate of the operator result over the three image planes is shown as brightness, normalised to a maximum value of 255. (b) A simple single-pixel difference (see Equation 3.1) applied to the same image. Both would normally be thresholded before use to reduce noise.

Another significant difference between this and established edge detection methods is that each difference-value depends on only two pixels, in this case a base pixel and the one immediately above it. Thus, to determine the colour-space transition direction at any particular point requires access to only two pixels. Determining the transition directions over a line of n pixels in any orientation requires access to only $n + 1$ pixels. This is an important result for the sub-sampling approach discussed below. Standard edge detection algorithms process the entire image, and even when localised each output value is dependent on many pixels. Such techniques are too computationally expensive to be effective in a robot vision system with limited resources, such as the ERS-7.

The difference between two pixel values may be represented as a discrete vector in a three-dimensional colour-space. The direction of this vector is independent of the actual colours perceived, capturing only the relationship between the two pixels. An edge between two objects of different colour will be characterised by a particular set of similar vectors in colour-space. This may be visualised as a region in colour-gradient space (see Figure 3.4), in some ways similar to

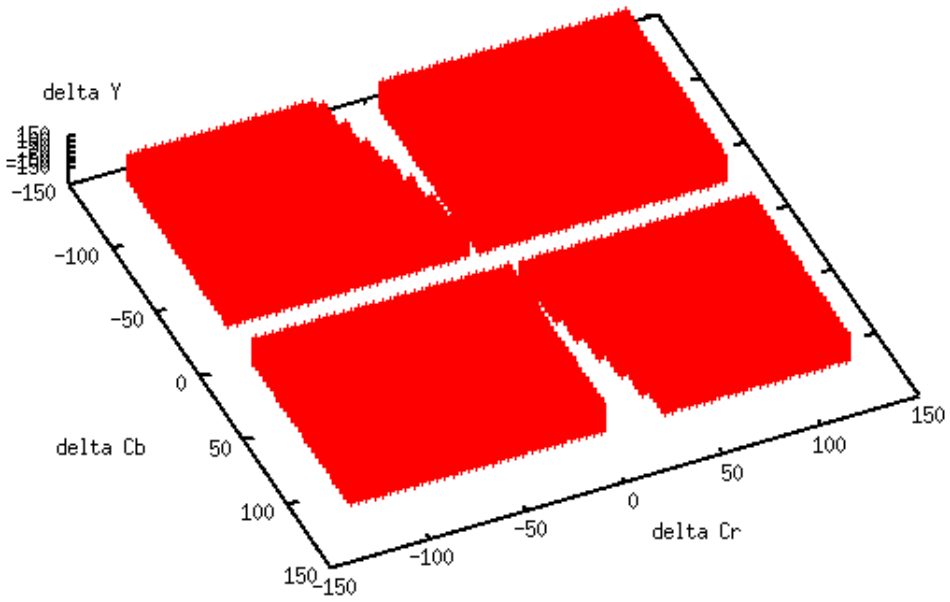


Figure 3.4: A representation of green-orange transition vectors as a region in the YCbCr-gradient space. While this captures transitions in both directions it is an overly generous fit.

the representation of symbolic colours as regions in colour-space. Pixel transitions with similar vectors to those that characterise particular object boundaries may be explicitly detected, while other transitions can be ignored. Many different colour pairs may be detected in just one pass over the image.

Since the direction of the vector does not depend on the actual pixel values detected these vectors will be independent of any linear shift of the colour-space. Although no colour-space shifts experienced are perfectly linear, many approximate linearity, being a compression or expansion of the colour-space in one particular direction. For example, shadowing, or otherwise reducing the intensity of ambient light, intuitively compresses the set of observed pixel values towards black (i.e., makes them darker, but bright pixels experience greater change than dark ones). This reduction in intensity results in a much smaller change (depending on their magnitude) to the vectors representing object boundaries in the image. Thus, detecting these vectors, rather than specific colour values, is likely to be more robust to changes in lighting conditions.

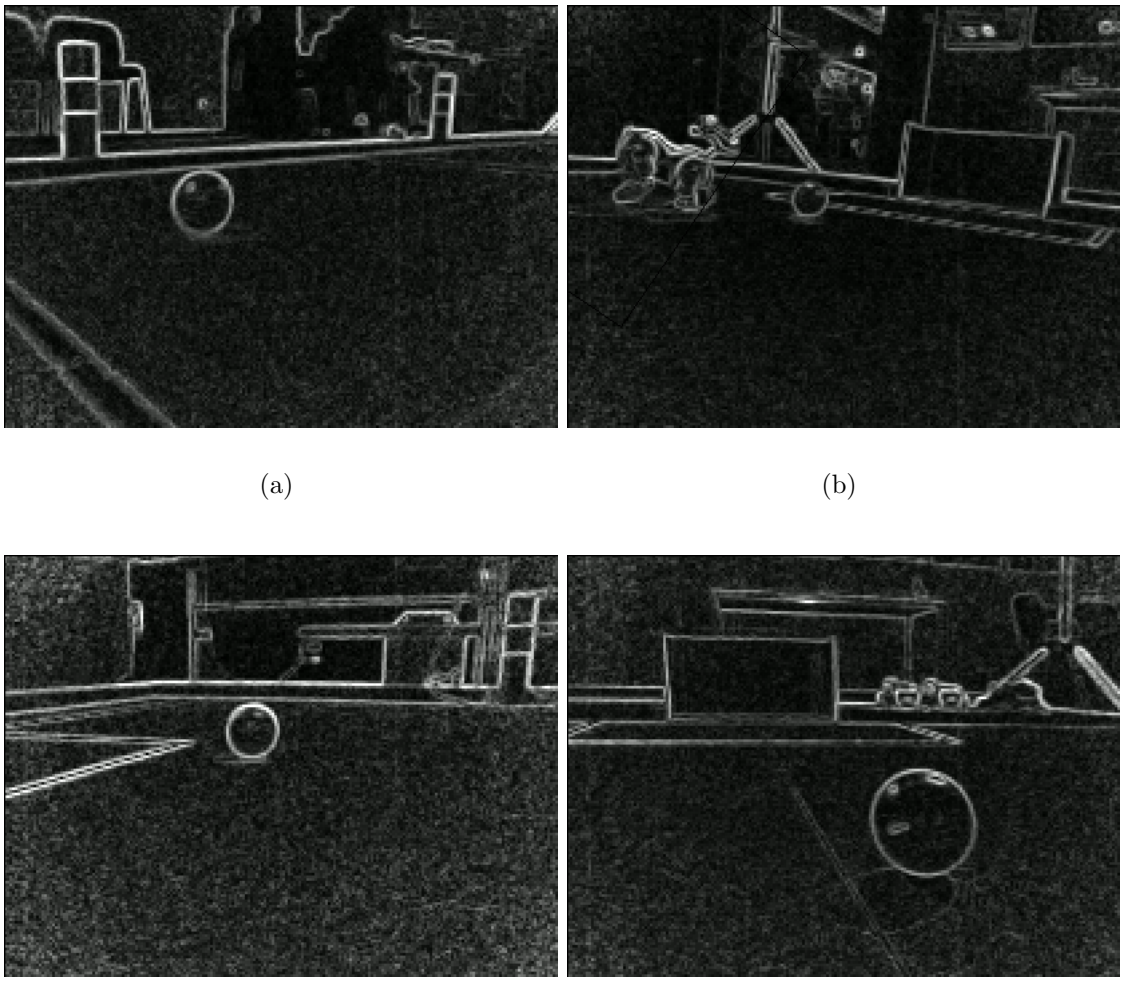
3.1.2 Sub-sampling images

The information contained in any given image is distributed over its area, usually in a non-uniform fashion. The distribution is non-uniform because neighbouring pixels are highly correlated, so the image carries redundant information [5]. Thus a vision system that processes every pixel in each image necessarily processes redundant information. Given the constraints on resources available to robot vision systems a more efficient approach should be able to extract almost the same feature information without sampling all pixels in an image.

The problem remains, however, to select which pixels to process. In the RoboCup four-legged league domain important objects and landmarks are regions of uniform colour, such as the orange ball, green field, white lines and yellow, blue and pink beacons and goals. Pixels in the centre of these regions of colour carry little information; their presence can be inferred from the surrounding similarly coloured pixels. It is the edges of these objects that carry important information about the position, size and orientation of these objects. Ideally, only pixels near information-bearing edges should be sampled with a high probability, while pixels in regions of relatively uniform colour should be sampled less frequently. In other domains the texture of objects may also provide useful information and samples in highly textured regions may provide additional information.

Without more information about typical image composition the only appropriate method would be to begin with a random sampling, but in the domain of the four-legged league knowledge about the environment and geometry of the robot allows a better estimate. Figure 3.5 shows the results of applying Roberts' operator to a number of typical images. Regions of high information are displayed as edges. It is apparent that in general the lower part of an image carries less information (edges) than areas higher up, although it must be noted that the very top part of each frame usually captures background information from outside the playing field. There is a visible amount of noise in areas of uniform colour, such as the field.

Intuitively, the expected information density should reflect the fact that objects close to the robot appear lower in the image and larger than objects far from the robot. An object such



(a) (b)

(c) (d)

Figure 3.5: The results of executing Roberts' operator on a number of typical in-game images. Note that the bottom part of each image, corresponding to objects close to the robot, typically contains sparse information. Objects farther from the robot appear smaller and higher up the image, leading to greater information density near the horizon.

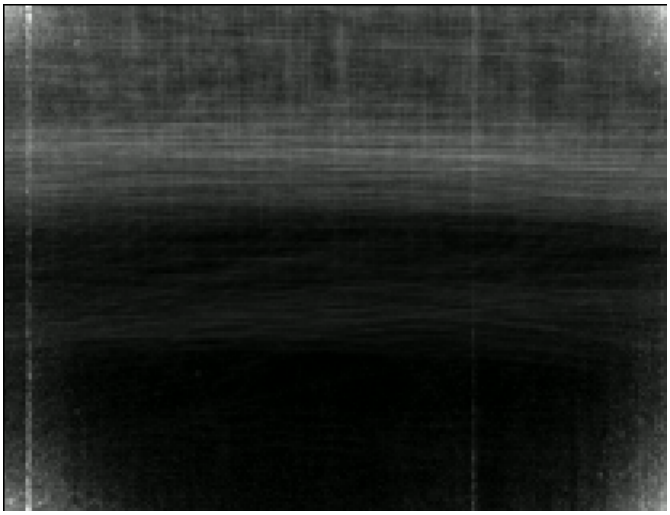


Figure 3.6: The aggregate result of Roberts' operator over four hundred images. Note the dark region near the bottom of the image, signifying an area of low average information density. Near the top of the image the brighter region represents an area of higher average information density. Noise due to chromatic distortion correction is visible in the image corners.

as the ball carries the same amount of information in either case, but when closer to the robot this information occupies a larger area in the image. Similarly, field lines close to the robot are sparse in the lower areas of the image. Lines further from the robot appear higher up and more closely spaced in the image.

Figure 3.6 shows a normalised sum of the results of Roberts' operator over a series of four hundred images captured from an ERS-7, holding its head level while turning its body on the spot at approximately 4.5 seconds per revolution. The robot is positioned one quarter of the field length from one goal-line and laterally in line with the nearest goal box corner. The result of Roberts' operator on each individual image is thresholded at 40 before being added to the aggregate, in order to the reduce noise present in Figure 3.5. From this it can be seen that the average expected information density is low near the bottom of each image and increases to a peak about one third of the distance from the top of the image. Noise from chromatic distortion correction is prominent in the corners of the image. Again note that the very top of the image is largely background information. The information density is approximately constant along horizontal lines.

This generalisation, that higher parts of an image are likely to carry more information, is only valid when the robot's head is held upright. If the position of the head changes then the expected information density within the image changes too. An artificial horizon may be calculated from the geometry of the robot's limbs to provide a reference point invariant with its stance. As the robot's head and legs move this horizon may be recalculated. This suggests that regions of the image close to the horizon should be sampled with higher frequency than areas far from the horizon.

Another important factor when selecting pixels to sample is the relative importance of information obtained in different parts of the image. For example, in the four-legged league the ball is generally held to be of the highest importance, and accurate detection of the ball vital to the behaviour of a successful agent. The accuracy of this information becomes even more important when the robot is close to the ball. In these situations the ball frequently appears in the lower average information areas calculated above. It remains important to sample the lower regions of the image sufficiently to give accurate detection of the ball in these cases.

Hence, advantage may be taken of the non-uniform information distribution over typical images. Knowledge of the domain may be used to direct processing power to information-rich areas of images, avoiding unnecessary processing of redundant information.

3.1.3 Conclusion

From the theoretical basis presented above a number of conclusions may be drawn. Relying on detected pixel values without regard for neighbouring pixels leads to a reliance on brittle, static colour classification procedures which are prone to failure under variations in lighting conditions. An alternative is to focus instead on the relationships between neighbouring pixels. While absolute pixel values change with variations in ambient light the difference between neighbouring pixels remains somewhat constant, allowing consistent detection. A vision system that relies on these colour relationships rather than absolute colour is likely to be more robust to variations in lighting conditions, in a similar way to existing edge detection algorithms.

Information content is not uniformly distributed across typical images. Rather, some areas of an image contain dense information while other areas express very little useful information. In the four-legged league areas of high informational value are typically around the edges of objects, rather than in areas of uniform colour. An analysis of typical images in a given domain can provide hints as to which areas of an image average high informational value. Thus, a robot vision system aiming to maximise efficiency need not sample the entire image. Rather, a vision system may sample areas of high average information at a greater density than areas of typically low information.

This approach breaks the processing of the vision system into two distinct levels: low-level feature detection from image sampling, and high-level object recognition from the detected features. This separation allows object recognition to become somewhat independent of the conditions under which input features are detected, although there remain advantages in allowing feedback between the object recognition and feature detection systems.

3.2 Implementation

The following section outlines in detail the implementation of a sub-sampling robot vision system for the RoboCup four-legged league. This implementation is strongly biased towards working accurately, robustly and consistently in the RoboCup competition rather than towards any notion of elegance or mathematical correctness. Despite goals of domain independence implied above, advantage is taken of any reasonable domain-specific assumptions that may be made. Nevertheless, this system is implemented with fewer lines[1] of less complex code than previous *rUNSWift* vision systems. This implementation was used with considerable success in the RoboCup 2005 competition.

[1] A rough count of lines of vision-related code reveals that this implementation spans approximately 10,900 lines of C++ code, while the *rUNSWift* system used in the RoboCup 2004 World Championships used 17,300

3.2.1 The YCbCr colour-space

The native colour-space of the ERS-7 camera is YCbCr. This colour-space is a discrete specification of the YUV colour-space, widely used in broadcasting, and YCbCr is often incorrectly termed YUV. The Y component in this space represents luminance, or brightness. The Y channel alone presents an 8-bit grey-scale representation of the camera image. The Cb (U) and Cr (V) channels are 8-bit chroma (colour) channels, representing the blue and red components respectively. The YCbCr colour-space is a linear transformation of the more widely known RGB colour-space, although the gamuts[2] do not completely coincide.

Although some implementations of colour segmentation [22, 2] and prior *rUNSWift* research have reported some benefits from conversion to alternative colour-spaces, this implementation continues to use the YCbCr space. A separation of the luminance component from chroma components is advantageous but since YCbCr provides this, the benefits of alternative colour-spaces do not justify the added complexity of conversion.

3.2.2 Colour classification

Although this implementation attempts to move away from a reliance on statically classified colour such classifications nonetheless provide useful information. Given the specification of the four-legged league domain in terms of objects of uniform colour it would seem foolish to ignore symbolic classification completely. In this implementation symbolic segmentation is used extensively in landmark detection and provides useful confirmation of features detected through other methods. However, in contrast to most previous attempts the system described in this chapter does not routinely classify all pixels in an image; only those pixels selected for examination by the sub-sampling approach are classified.

Colour classification is performed at run-time via a look-up table. This table, approximately 2MB in size, maps the seven most significant bits of the Y, Cb and Cr components of a pixel to

[2] A gamut refers to the subset of colours that can be represented in a particular colour-space.

one of ten symbolic colours. The table is constructed before run-time by generalisation from a number of hand-labelled images.

Two approaches are presented for the generation of this table: a ripple-down-rules classifier and a simple weighted kernel classification. In both cases a large number of images are captured from the robot to a desktop PC via a wireless Ethernet connection. Correction is applied for chromatic distortion [27], and the particular robot used to capture images is treated as the reference robot for linear adjustment between robots as described in [13]. These corrected images are then loaded into a classification application for manual classification. [15] implemented a real-time application for colour classification that provides immediate feedback of the effects of any changes to the classification. The application displays images and the symbolic classification of each pixel simultaneously, allowing the user to re-classify pixels and displaying the results of the re-classification immediately. Such immediate feedback provides a significant time-saving benefit to the user over previous systems, in which manual classification and automated generation of the look-up table were two distinct tasks.

[15] implemented a ripple-down-rules (RDR) classifier that was used for much of the development of this system. Combined with the interactive application mentioned above this dramatically reduced classification time, in turn increasing the number of training samples that could be provided by an expert within a given amount of time. Typically more than two hundred images were used as training data with this system, compared with the 70 described in [27]. However, as the classification becomes finer with repeated training the depth of the RDR decision tree increases and the cost of traversing this tree to classify pixels close to classification boundaries grows. Since the RDR tree is simplified to a look-up table for loading to the robot this doesn't affect performance of the vision system, but it significantly slows the classification application, which must traverse the tree many times to provide real-time feedback of classification changes. While allowing arbitrarily fine classification, the RDR always generalises along axis-parallel lines, which may not be appropriate. Thus a structure allowing arbitrarily complex classification and with constant time training and classification was sought.

As an alternative to the RDR classifier a simple weighted kernel classification was used for the final rounds of the four-legged league competition. Using this method each training sample increases a weighting for that particular YCbCr value toward the classified colour. This increase in weighting also flows to neighbouring values in colour-space within a fixed Euclidean radius, giving generalisation. Classification involves the constant-time calculation of the highest weighted symbolic colour for a particular YCbCr value. User experience indicates both that this method of generalisation is more appropriate than axis-parallel generalisation, and that there is a significant useability improvement in a responsive real-time classification application. By way of demonstration, more than one thousand images were used for training data within a twenty-four hour period in preparation for the final rounds of the four-legged league, with no noticeable behavioural differences from the more time-intensive training methods.[3]

While the colour segmentation used in this approach is similar to many others, in that the optimal segmentation is a finely tuned balance between the colour classes, the optimal segmentation for the methods outlined below does not generously classify the orange of the ball in order that it appears as orange even under adverse lighting conditions. The ball detection methods are highly robust to much of the ball being classified pink or red; only a little orange is necessary to confirm that the object recognised is indeed the ball. This means that colours that might otherwise falsely appear as orange, such as the pink of beacons, red team uniform and yellow of the goal, can safely be classified to the correct colour. In fact, it is relatively important that orange classification does not occur widely outside the ball, creating false candidate ball features.

Colour classification, along with chromatic correction, is applied at the time of access to pixels in each image frame. Unlike previous approaches, where corrections and classifications are applied to the entire image, the sub-sampled approach never consults many pixels in each image. Applying correction and classification at the beginning of image processing would thus be poor use of compute power.

[3]Thanks to Dr William Uther for the concept and implementation of the kernel classifier.

3.2.3 Scan lines

Selection of which pixels to process is by means of a horizon-aligned, variable-resolution grid placed over each image. This grid is inspired by, and quite similar to, that used in [18], however, the processing of pixels under the grid differs significantly.

An artificial horizon may be calculated from knowledge of the geometry of the robot's stance and camera. The horizon represents a line through the image with constant elevation equal to that of the camera. This horizon provides a reference point that is invariant with the stance of the robot and aligns the grid with the high-information areas of the image as described in section 3.1.2. The horizon is calculated through projection of two fixed-elevation points, and the line joining them, onto the image plane. This is then adjusted for the optimised motion gaits that were simultaneously developed in [7]. Sensor noise results in significant inaccuracies in the estimated horizon while the robot is moving; [3] have had success stabilising the horizon by consulting the ERS-7's internal accelerometers.

Under the assumption that the camera image is oriented "upright", such that pixels higher up in the image are projections of features at greater altitude in the environment, a horizon-aligned grid of scan lines is constructed. These scan lines represent the pixels in each image that will be initially processed by the feature detection algorithm; pixels that do not lie on these lines may not be processed.

A scan-line is constructed from the centre of the horizon line and perpendicular to it to the lower edge of the image. Scan lines are then constructed parallel to this first line on either side of it at a fixed spacing of sixteen pixels; scan lines continue to be constructed at this fixed spacing until such lines lie entirely outside the image frame. Between each pair of these "full" scan lines a parallel line segment is constructed from the horizon line with a fixed length of 64 pixels. These "half" scan lines are thus eight pixels from each of their neighbouring full scan lines. Between each of these half scan lines and their adjacent full scan lines a "quarter" scan line is constructed with a fixed length of 48 pixels, spaced four pixels from the scan lines on either side. Finally, beginning sixteen pixels below the horizon and doubling this gap for each

line, scan lines are constructed parallel to the horizon line until such lines lie entirely below the image frame.

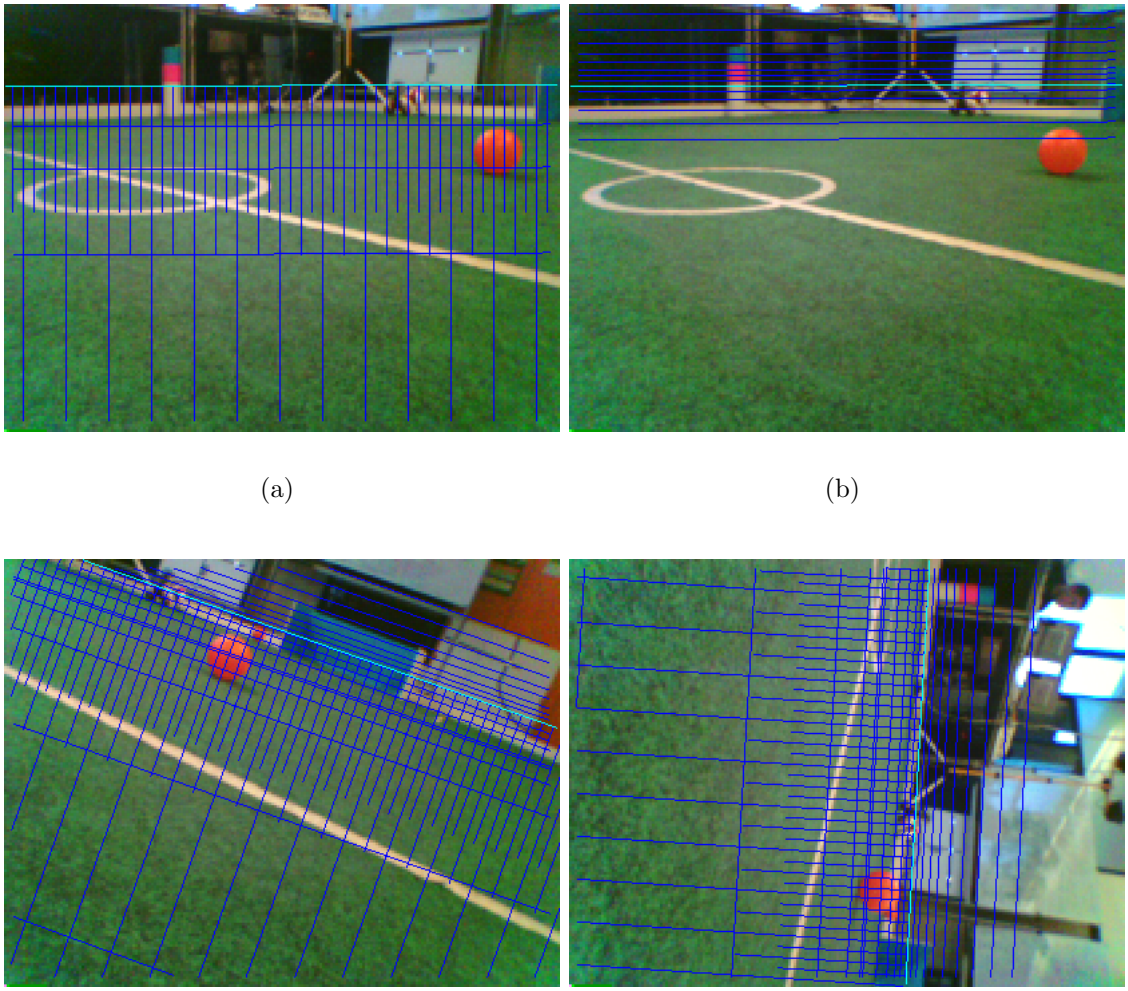
This constructs a grid as shown in Figure 3.7 (a), with a greater density of scan lines closer to the horizon and more sparsely spaced lines further from it. This grid is used for detection of features related to the ball, field, lines and obstacles. The majority of the scan lines are perpendicular to the horizon, running from pixels that project close to the robot to pixels that project further away. The few scan lines parallel to the horizon are placed to capture features on field lines aligned with the camera axis that would otherwise lie entirely between scan lines. Since no features are expected to appear above the horizon, no scan lines are created above it.

A separate set of scan lines is constructed for detection of landmarks. Beginning just below the horizon line and continuing with an exponentially increasing gap above it, scan lines are constructed parallel to the horizon line as shown in Figure 3.7 (b). These lines are used for detection of the landmarks and goals around the field.

This same pattern of scan lines may be constructed regardless of the camera orientation, as in Figure 3.7 (c). In cases where the camera is rotated further than one quarter-turn about its axis the lower edge of the images becomes the upper edge, as shown in Figure 3.7 (d); in such cases scan lines continue to be processed in the same order and direction. In situations where the camera is tilted up or down in the extreme the horizon may not appear within the image. In these cases an approximation is made: if the horizon would appear above the top of the image it is constructed along the upper edge of the image instead; if the horizon would appear below the bottom of the image it is constructed along the bottom edge (and no vertical scan lines are created).

C++ code for constructing the scan lines is presented in Appendix A.1.

This grid of scan lines defines the initial access pattern for pixels in the image. Vertical scan lines are processed from bottom to top; horizontal scan lines are processed from left to right. If no features of interest are detected then these are the only pixels processed in the image. The scan lines typically cover about 6,700 pixels, or twenty percent of an image. More pixels in



(a) (b)

(c) (d)

Figure 3.7: (a) The pattern of scan lines (shown in blue) used to find features on the ball, field, lines and obstacles. The horizon is shown as a pale blue line. (b) The pattern of scan lines used to search for landmarks. (c) The scan lines are constructed relative to the horizon. (d) When the camera rotates more than 90° about its axis the scan lines run up the image.

the vicinity of features and objects of interest are accessed during the image processing but the majority of pixels are not processed at all.

In some cases scan lines are also constructed dynamically. After processing of the initial scan line pattern described above more information is available about the information distribution over an individual image. It may be the case that some features have been detected, but not a sufficient number to robustly recognise a particular object. It is likely that additional processing in the vicinity of the previously detected features will lead to detection of further features, since there is significant spatial locality in edge information. This dynamic scan line creation is implemented for detection of ball edge features. If fewer than seven ball features (see section 3.2.4) are found in the initial pass then an extra fixed-size grid of scan lines is constructed around each of the detected features. This grid is offset from the original scan line grid, and of a density equal to that of the quarter scan lines in the original grid (four pixels separation). The grid consists of two vertical scan lines of sixteen pixels to the left and right of each detected feature, and one horizontal scan line of length sixteen above and below the feature. These additional scan lines are oriented without regard to the horizon. Figure 3.8 shows the result of processing extra scan lines around sparse ball edge features.

3.2.4 Feature detection

Feature detection takes place over the scan lines described in the previous section. Features of the ball, field, lines and obstacles are detected in the vertical and low horizontal scan lines. These scan lines are processed from bottom to top (or left to right) and each pixel is compared with the previous pixel in the scan line. Rapid changes in the Y, Cb or Cr values indicate edges in the image, and particular edge vectors are recognised as being features of a particular environmental object.

In the image presented in Figure 3.9 one scan line has been highlighted in red. Plotted below the image are histograms of the Y, Cb and Cr levels of each pixel along this scan line (top-to-bottom), the gradients of these components along the scan line, and a sum of the magnitude of

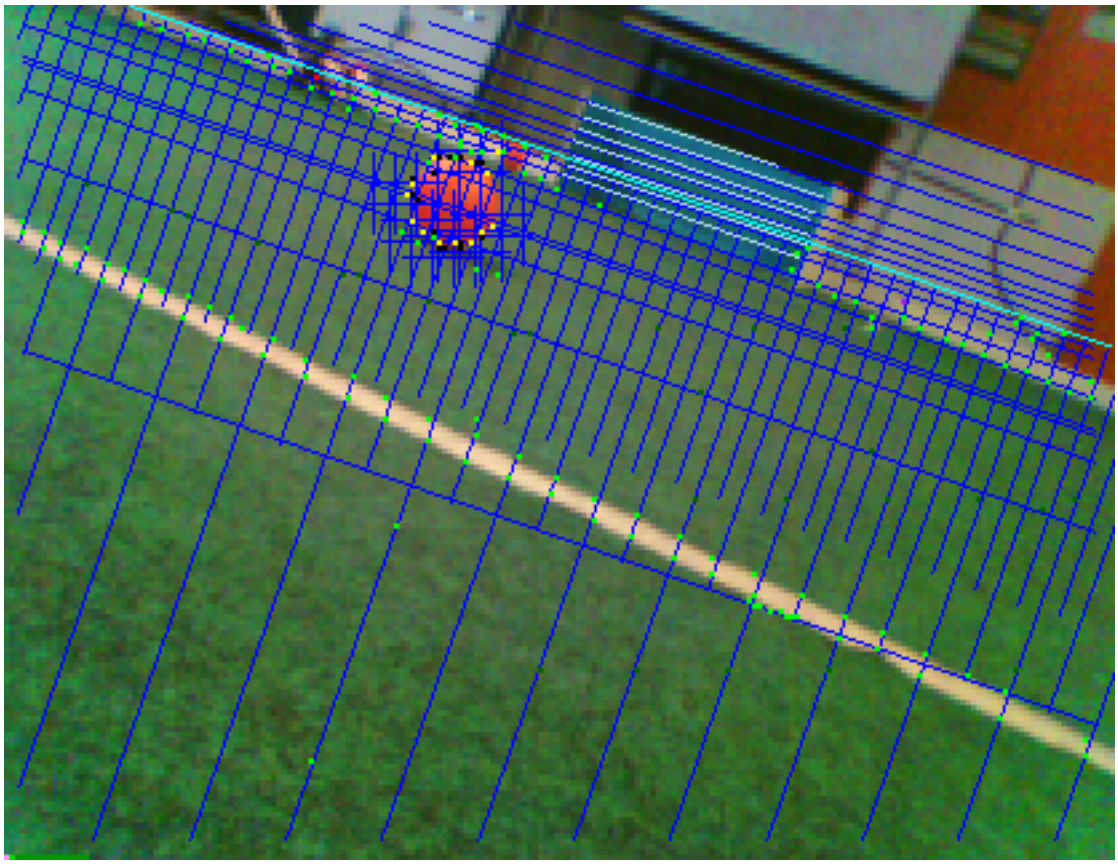


Figure 3.8: The complete scan line pattern with detected features displayed. Additional scan lines are constructed when few ball features are detected in the initial scan pattern. More features are most likely to be detected near existing features.

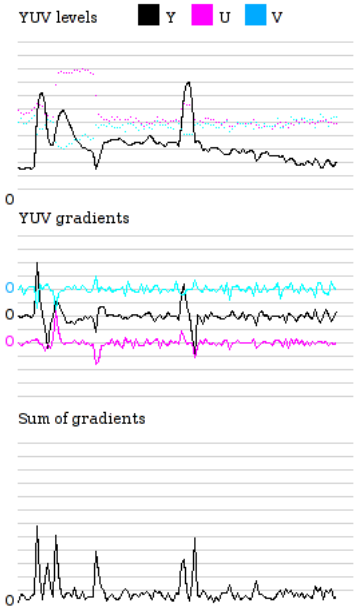
these gradients. The field line close to the camera shows clearly as a large peak in the Y level histogram, with a minor peak in the Cr level and little change in Cb. The two edges of this line show as a sharp peak and dip in the Y gradient histogram, with a smaller peak and dip in the Cr level. These combined peaks form an even more prominent feature in the sum-of-gradients histogram.

The upper edge of the ball is apparent as a large increase in both the Cr and Y channels and a decrease in the Cb channel. This also results in a prominent peak in the sum-of-gradients histogram. The edge of the green field, near the top of the scan line, is also recognisable in the histograms. Note, however, that there is also a significant amount of noise present, particularly in the sum-of-gradients histogram.

The edges of the field line and ball have different signatures in the gradient histograms. While a green-white field line edge is characterised by a strong difference in Y, a corresponding but smaller difference in Cr and little change in Cb, the orange-green ball edge is characterised by a strong change in Cr, a corresponding change in Y and an opposing change in Cb. Thus the transitions between different environmental colours can be identified by the corresponding differences in the Y, Cr and Cb channels of adjacent pixels. As discussed in section 3.1.1 these characteristic differences may be represented as regions in YCbCr-gradient space.

For this implementation these regions were manually defined through human analysis of a large number of images. Once defined, however, no changes were necessary in order for feature detection to function well under all environments encountered. In addition to detecting these transitions, statically classified colour was used to confirm the type of edge encountered. Thus, feature detection comprises two tests: detecting particular colour transitions, and confirming by checking the colour classification of surrounding pixels.

(a)



(b)

Figure 3.9: (a) A YCbCr image showing scan lines, with one highlighted. (b) Histograms of the colour information along the highlighted scan line (top to bottom). The top graph shows Y, Cb (U) and Cr (V) levels for each pixel. The middle graph shows the instantaneous gradients of these channels. Note the peaks in these gradients at the transitions between the ball, white line and green field. The bottom graph shows a sum of the magnitude of all three gradients.

Detecting ball features

Ball features represent transitions between the orange ball and the green field or white field lines. Occasionally transitions into the yellow or blue goal are also encountered and the majority are correctly detected by the procedure outlined below.

For each pixel in turn running up the scan line let (y, u, v) be the values of the Y, Cb and Cr channels respectively for this pixel, (dy, du, dv) be the difference between the channels of this pixel and the previous (i.e., next lower) pixel, and sum be the sum of the absolute values $|dy| + |du| + |dv|$. A ball feature satisfies:

- $sum >= 20$ (a minimum gradient over all channels), and
- $|du| > 15$ (a minimum gradient in the Cb channel), and
- either
  - $dv = 0$, or
  - $|\frac{du}{dv}| < 4$ (slope of Cb less than four times slope of Cr), or
  - $sign(du) = -sign(dv)$ (Cb changes in opposite direction to Cr).

This is visualized in Figure 3.4. In order to avoid confusion between strong specular reflection and white field lines a ball feature must also satisfy $y < 180$. Note that these rules capture transitions both towards and away from orange, without any explicit notion of orangeness.

These features are then confirmed by consultation with the statically classified colours of nearby pixels. While processing a scan line, information about the direction of the scan line in the image is calculated. For these tests this is simplified to the nearest basis direction: up, down, left or right. The value of du calculated above gives the direction of change at this pixel: if $du > 0$ then the transition is towards orange; if $du < 0$ then the transition is away from orange. Thus the direction of the centre of the ball can be approximated as equal to the direction of progress along the scan line if du is positive, or opposite to this direction of progress if du is

negative. A line of five pixels beginning at the pixel under test and progressing in the direction of the centre of the ball (simplified to a basis direction) are examined. In order for the transition to be confirmed as a ball edge the classified colour of these pixels must satisfy:

- at least three are classified orange, and
- no more than one is classified red, and
- no more than one is classified pink, and
- no more than one is classified yellow.

Thus we confirm that there are at least some orange classified pixels where they would be expected. Note that the pixel under test is not required to be classified orange; in fact it is quite often the case that the very edge pixels are somewhat blurred and take on a classification of white or yellow. Note also that a consistent run of orange is not required; any three from the five pixels may be orange.

From the images in Figure 3.1 it can be seen that pixels near the upper edge of the ball maintain the correct classification under the widest range of lighting intensities. This test therefore favours transitions at the upper edge of the ball. The lower part of the ball deteriorates to pink or red quite quickly, and indeed this was a major problem with purely colour-based blobbing approaches. This test will continue to recognise edges so long as three out of five pixels are classified orange, but thereafter will reject the transitions as likely spurious. Shadowed edges on the lower half of the ball are detected with a colour based approach outlined in section 3.2.5.

These colour based tests are necessary since other objects in the expected environment may satisfy the transition tests above. For example, the red team uniform is of a similar colour to the ball, particularly under bright ambient lighting. Thus a transition between a red uniform and the green field may be detected as a candidate ball edge. If a colour table is generous towards classifying red then these transitions may be filtered due to the presence of multiple red pixels near the transition point.

At a more fundamental level these colour-based tests are necessary because the transition tests are overly lax. The complexity of the transition tests here can be likened to the simple thresholding that was previously used for colour segmentation. The region satisfying the tests is not well fitted, and includes many transitions outside the desired set. This method was chosen because it is simple enough to provide an easy implementation and demonstrate the validity of the concept without introducing many unnecessary complexities; future approaches could adopt more complex transition definitions to reduce reliance on colour segmentation even further.

Detecting field lines

Field line features represent transitions between the green field and white field lines or boundaries. For each pixel in turn (after the first) running up the scan line define $(y, u, v), (dy, du, dv)$ and *sum* as for ball feature detection in the previous section. A field line feature satisfies:

- $|y| > 32$ (a minimum value of y), and
- $|dy| > 15$ (a minimum gradient in y), and
- $|du| < 40$ (a maximum gradient in Cb), and
- $|dv| < 40$ (a maximum gradient in Cr), and
- $|du| < 4$ or $sign(du) = sign(dv)$, (a small gradient in Cb, or Cb and Cr slope in the same direction).

The direction of change may be calculated in a similar fashion to that for ball features, using the indicator dy rather than du. No symbolic colour tests are applied to field line edges. Note that this test is likely to misrecognise the boundary between robots and the green field as being a field line edge; no attempt is made to prevent this since such noisy data is handled robustly in the localisation module, described in [20].

In addition to detecting field lines, a sparse sampling of each image is made for the purpose of detecting the green of the field. These green points are used in the localisation module to

resolve ambiguity in a number of otherwise symmetric cases. Every 32 pixels along half of the full scan lines a check is made: four line segments of nine pixels with a mutual intersection at the centre of each line segment form an axis-aligned star shape containing 33 pixels. If at least two-thirds (22) of these pixels are classified as green then the centre pixel is marked as a *field-green* feature.

Detecting obstacles

The method for detecting obstacles presented here differs from most other attempts. While [18] detected an absence of green as being indicative of an obstacle, this approach specifically detects the shadows on the field caused by objects lying on it. While processing a scan line the y value at each pixel is tested against a threshold value (35 was used at the 2005 RoboCup World Championships). If the y channel falls below this threshold the pixel is classified as an obstacle. A maximum of five pixels are classified as obstacles on any one scan line.

When the robot is looking downwards the ground near the robot is often shadowed by the robot itself. Human referees may also cast faint shadows on the field that should not be detected as an obstacle. Further, the ball also casts a shadow on the ground immediately beneath it which is also undesirable to detect. A check is necessary to prevent detection of many false obstacles caused by faint shadows and lighting variations. A state machine keeps track of the classified colour of pixels on the scan line as they are processed. The twenty pixels immediately above a candidate obstacle are tested. If more than ten green classified pixels or five orange classified pixels are encountered then the obstacle candidate is discarded.[4]

No attempt is made to classify the source of the obstacle; in particular the shadows caused by other robots are not classified into teammates and opponents. The obstacles detected in each frame are projected onto the field and passed to an obstacle tracking module which maintains a map of recently detected obstacles. This obstacle map and its uses are described in [19].

[4]Thanks to Nobuyuki Morioka for the concept and implementation of these checks.

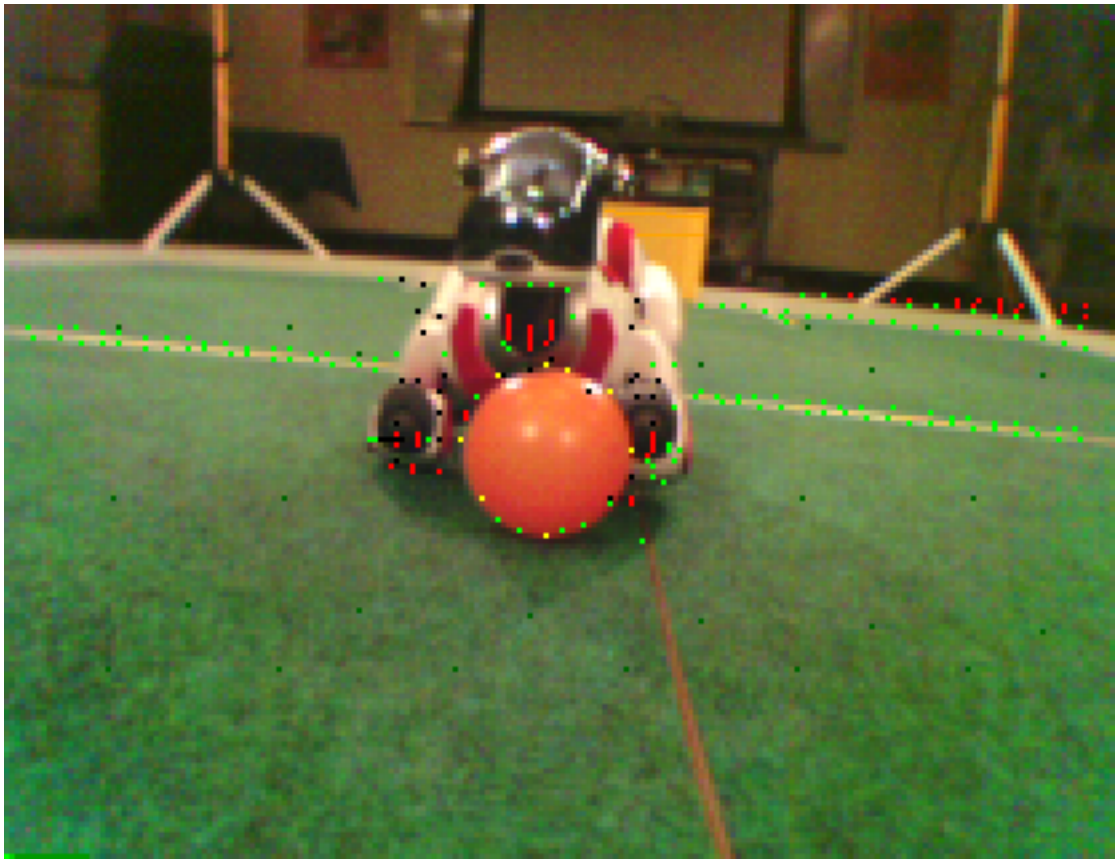


Figure 3.10: The results of feature detection over a sample image. Ball edge features are displayed as yellow points, line edge features as light green points, obstacles as red points and field-green features as dark green points. Black points represent features that have been discarded due to checks applied after initial detection.

The results of ball, line and obstacle detection are shown in Figures 3.8 and 3.10. C++ code for the detection of ball, field line and obstacle features is presented in Appendix A.2.

3.2.5 Symbolic colour detection

As noted earlier, colour remains a very important part of the four-legged league domain, and this approach continues to use symbolic colour classification for detection of beacons and goals. This landmark detection takes place over the upper horizontal scan lines described in section 3.2.3. These scan lines are scanned left to right and each pixel in turn classified into one symbolic colour via a static colour segmentation created using the methods outlined in section 3.2.2.

Colour remains an appropriate indicator for the landmark objects as they are far less subject to variations in lighting during a four-legged league match than on-field objects such as the ball. Lighting remains reasonably constant during a match but while the ball and field lines are subject to shadowing, reflections, blurring and other distortions the beacons and goals lie outside many of these effects. Although the colour segmentation must be tuned for a specified environment, the perceived colour of beacons and goals changes little during the course of a match.

Detecting landmarks

A state machine tracks the number of consecutive pixels found of each of pink, yellow and blue along horizontal scan lines, allowing for up to one pixel of "noise" (any other colour), along with the start and end points of these runs of colour. Beacons are detected by the pink square that appears on all beacons. A run of five consecutive pink pixels (plus one pixel of noise) in a scan line creates a *beacon feature*. These features are passed to the object recognition system outlined below for further processing. Goals are detected by their uniform colour of pale blue or yellow. A run of twelve blue or yellow pixels (plus one pixel of noise) creates a blue or yellow *goal feature*. Again these features are passed to the object recognition system for further processing.

A slight modification to the thresholds is made when the robot's head is held low and the horizon coincides with the top of the image, such as when the robot's head is controlling the ball. Pixels near the top border of the image are subject to significantly more noise (mainly due to chromatic "ring" distortion and its correction) than those near the centre of the image. Pixels near the borders of the image are therefore more likely to be misclassified. Thus when the robot holds its head down over the ball and is searching for the goal an additional "noise" pixel is allowed in goal features.

Detecting the wall

While the area outside the green carpeted field is undefined by the four-legged league rules there is often a low wall or region of uniform colour surrounding the field. It is advantageous to detect this to allow filtering of features and objects appearing off the field. The *rUNSWift* laboratory field has a 7cm high white wall around the green carpet, while the field at the RoboCup World Championships had a metre of grey carpet before a low white wall. The following method is applicable to both environments.

A state machine tracks the number of green, grey or white, and other-coloured pixels encountered during the scanning of each vertical scan line. A wall feature is detected by a series of four green classified pixels (allowing one pixel of noise) followed by a a series of at least five white or grey classified pixels, followed by a series of two pixels of other colours (allowing two pixels of noise). The requirement for non-green above the white pixels prevents close field lines being detected as walls. A wall feature is created midway between the start and end of the white/grey pixel series.

Detecting faint edges

There are a number of cases where edges in the image become blurred so the ball and field line feature detection methods outlined above become less effective. The most common cause is motion blur; when either the camera or objects in the environment move quickly the result is a blurred image with indistinct edges. In such images the thresholds for change required for feature detection may not be met, since the transition is spread over many pixels.

In these cases an alternative feature detection method is used, based on segmented colour. While symbolic colour classification is susceptible to changes in lighting it is fairly robust to blurring; edge detection exhibits the opposite tendencies. The ball is the only object for which these methods are used: field line detection is not important enough to warrant extra processing in the case of blurred images, although similar methods could be applied. Since the colour

segmentation is weighted away from orange in order to avoid orange classifications appearing on objects other than the ball, this ball feature detection must be robust to a large amount of non-orange colours detected in the ball, most commonly pink and red. Hence this method is also effective in very dark situations where no orange classified pixels appear near the lower edge of the ball.

A state machine keeps track of the number of orange, maybe-orange (i.e. pink, red and yellow) and non-orange (the remainder) pixels detected along a scan line. A transition is detected between non-orange pixels (usually green but sometimes white or black) and orange pixels, possibly with a number of intervening maybe-orange pixels. Three consecutive orange pixels are required to satisfy as an orange region, although the number of maybe-orange pixels before this is unbounded. If the transition is into an orange region a feature is created at a point midway between the last detected non-orange pixel and the first orange or maybe-orange pixel. If the transition is away from orange a feature is created at a point midway between the last detected orange pixel and the first non-orange pixel, so long as these two points are within six pixels of each other. On transitions away from orange the maybe-orange pixels are ignored since such pixels usually occur on the lower half of a valid ball.

A small number of additional checks are added to ensure that orange in the red team uniform does not cause false ball features to be recognised. C++ code for the state machine implementing this symbolic colour edge detection is presented in Appendix A.3.

3.2.6 Object recognition

Object recognition takes place over the features as recognised in the previous section. Object recognition involves grouping features related to the same real-world object and extracting the important attributes of these objects such as position in the image, position in the environment, heading, elevation, orientation and variances over these attributes. Although a sense of elegance might suggest benefits to a clean distinction between feature detection and object recognition it is convenient for the object recognition module to have direct image access. In fact, feedback

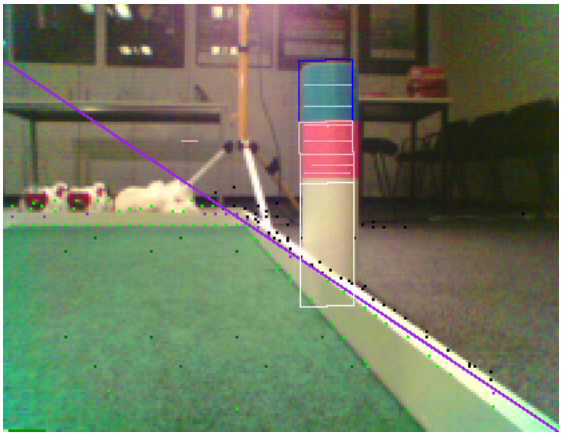


Figure 3.11: A recognised beacon. The beacon features are displayed as horizontal pink lines. Note a false beacon feature caused by pink in the background has been ignored. The pale blue lines in the beacon are not used for beacon detection. The white field wall has also been detected, displayed as a purple line; point features above the wall are ignored.

from the early stages of object recognition are necessary to focus computational effort when few features are initially detected.

Beacon recognition

Beacon recognition comprises grouping the detected pink features into candidate beacons then searching above and below the pink region for the other characteristic beacon colour (pale blue or yellow). Blue or yellow features are not used for beacon recognition, although it is likely that the following methods could be improved to take advantage of the information they carry.

Beacon features (which are horizon-parallel line segments) are grouped by merging adjacent "overlapping" features. Two features overlap if a line perpendicular to the horizon can be drawn that intersects with both features. A simple merge algorithm is applied: each feature is placed into a group of its own, then groups are merged if they contain features that overlap and are on adjacent scan lines. Thus one or more pink features are merged into a group if it is likely they are part of the same beacon, as shown in Figure 3.11.

A local search is then performed to classify each group of beacon features as one of the four possible beacons, or as falsely detected features. The centroid of the pink section, which appears ideally as a square, is estimated by an average of the midpoints of each feature in the group. This rough estimate is used only to locate other regions of the beacon later on. For each feature in the group a line is constructed perpendicular to the horizon and this line is searched linearly to determine the height of the pink region. The height of the entire beacon is initially estimated to be four times the maximum calculated height of the pink region, although this is later improved if more information is available. Occasionally the height of the pink region is artificially low, such as when the beacon lies partially off the top or bottom edge of the image. If the length of the longest feature is significantly greater than the calculated height then this length is used instead.

Given the estimated centroid of the pink region, the size of one side and the slope of the horizon, the centroid of the identifying patch of colour (blue or yellow) may be estimated as this side length either below or above the pink centroid, with reference to the horizon. The lower region is searched first for either blue or yellow classified pixels since, if this beacon has pink as the upper segment, the region above the pink segment may be off-field background information. An eight-point asterisk comprising seventeen pixels is searched and if at least six pixels of either blue or yellow are found then the beacon is uniquely identified as the pink-on-[blue|yellow] beacon. If no such colour is found then the upper region is searched in the same way, possibly giving a classification for the [blue|yellow]-on-pink beacon. If still no colour is found then this group of pink features is discarded.

Once the beacon is uniquely classified another line is searched to determine the height of the beacon, this time taking into account the expected colour above or below the pink region. If the height estimated from both coloured regions is similar to the initially estimated height then the new estimate is used instead. A further line is searched for the white bottom half of the beacon, and a new height estimate is made. Again if this estimate is similar to the previous height estimate it is used instead. Estimates over longer distances are likely to be more accurate as the error induced by discretisation of the image is reduced; however, it is often the case that

the entire beacon is not visible, lying partially outside the image frame or occluded by another robot.

The important properties for a beacon object may then be derived from its position and size in the image. Distance is calculated via an inverse-linear function of perceived height (in pixels), as in Equation 3.2. The constants for this function are calculated manually by fitting a curve to a number of observed function values. Heading and elevation within the image are determined from knowledge of the camera's field of view. These values may then be translated and rotated through the robot's stance and camera geometry to give a world position of the beacon.

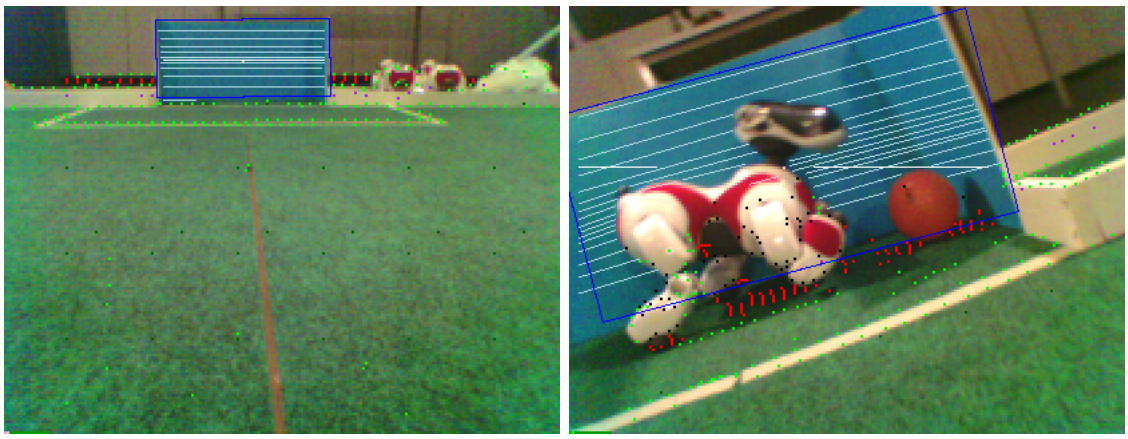
$$dist = \frac{8995.2}{height + 2.45167} \tag{3.2}$$

Only a single check is performed to confirm the validity of a candidate beacon (c.f. the sixteen checks listed in [13]). The centroid of the beacon must not be below the horizon by more than 25 pixels. This check rules out some invalid beacon features that might be detected by excessive pink occurring in the ball or red team uniform.

Goal recognition

Goal recognition comprises grouping the detected blue and yellow features into candidate goals. Goal features are grouped by merging adjacent "overlapping" features in the same way as beacon features, relaxed to allow up to one scan line separating features to be merged. However, it is possible for two distinct regions of the one goal to be visible as shown in Figure 3.12. Thus goal feature groups are also merged if they contain features on the same scan line.

The centroid of the goal is estimated by the centre of a bounding box drawn around the features. For each feature in the group a line is constructed perpendicular to the horizon, and this is searched to determine the height of the blue or yellow region. The goal's height is taken as the maximum of the heights at each feature.



(a) (b)

Figure 3.12: (a) A recognised goal. (b) The goal is often occluded by a robot, but grouping of features leads to correct recognition, even if the goal is divided in half. The two possible gaps for shooting are indicated by horizontal white lines. The recognised goal is assumed to be aligned with the horizon, so no attempt is made to detect the goal outline.

The important properties for a goal object may then be derived from its position and size in the image. Similarly as for beacons, distance is calculated via an inverse-linear function of perceived height, as in Equation 3.3. Heading and elevation within the image are determined from knowledge of the camera's field of view. These values may then be translated and rotated through the robot's stance and camera geometry to give a world position of the goal.

$$dist = \frac{5433.4}{height + 0.641} \tag{3.3}$$

A few checks are made to confirm the validity of a candidate goal. Firstly, the aspect ratio of the goal is checked to make sure it forms a sensible shape. This filters false goal candidates formed by only a small number of features and goals that lie largely outside the image frame, where reliable information cannot be obtained. Secondly, a goal must not appear further than twenty pixels above the horizon. Finally, a number of pixels underneath the candidate goal are tested for colour. If very few are found to be green the goal is rejected. The goal is also rejected if many are found to be white, as might occur in the blue or yellow patch of a beacon. The aspect ratio and colour checks are ignored if the robot holds its head down low while controlling

the ball: both are likely to trigger falsely, and goal detection is of utmost importance in these cases.

Wall recognition

Wall recognition was implemented by Nobuyuki Morioka, but is outlined here for completeness. Wall recognition comprises constructing a single straight line approximating the wall features as detected in section 3.2.5. A line is estimated with a modified random sample consensus (RANSAC) [10] algorithm. A random subset of half of the wall features (with a minimum of ten) is selected and the average line approximating these features is calculated. The aggregate error of all features from this line is calculated and if this falls within a threshold value the approximation is accepted, else the process is repeated. The process fails if fewer than ten wall points are detected or a suitable approximation is not found after five iterations.

Ball recognition

Ball recognition is performed after beacon, goal and wall recognition has completed. This ordering allows ball features detected above the wall, goal or beacons to be ignored. After this filtering, if fewer than seven ball features have been detected additional scan lines are created and scanned near existing ball features as outlined in section 3.2.3.

Ball recognition comprises estimating the outline of the ball from the detected features. This approach assumes that there is at most one ball in view. A circle is fitted to the ball edge features using a generalisation of Siegel's repeated median line fitting algorithm to circles, as described in [14]. Under the assumption that the majority of the points to fit lie on a circle, this algorithm claims robustness to up to 50% outlying data. Slight modifications were made to account for the fact that the ball frequently does not appear perfectly circular.

Given the parameterised equation of a circle $(x-a)^2 + (y-b)^2 = r^2$ all triplets of features (i, j, k) are considered, from a minimum of four features. Each triplet determines a circle by the

intersection point of perpendicular bisectors constructed to the chords formed by the triplet, similar to the close ball recognition method used in [6]. The parameters (a, b) are calculated separately as in Equation 3.4: for each pair (i, j) take the median of the parameter over all choices for the third point, k; for each i take the median parameter over all choices for the second point, j; and take the result as the median over i.

$$a = \underset{i}{\text{med}} \underset{j \neq i}{\text{med}} \underset{k \neq i,j}{\text{med}} a_{i,j,k} \qquad b = \underset{i}{\text{med}} \underset{j \neq i}{\text{med}} \underset{k \neq i,j}{\text{med}} b_{i,j,k} \tag{3.4}$$

In contrast to the algorithm presented in [14] the radius is calculated from a single median after the positional parameters have been calculated (Equation 3.5). This aids stability in the presence of a large number of outliers. If at least seven features are present then the middle three are averaged to give the radius. This averaging helps to reduce jitter induced by image noise.

$$r = \underset{i}{\text{med}}\ r_i \tag{3.5}$$

The important properties of the ball may be derived from its position and size. Distance is calculated via an inverse linear function in a similar fashion to that of beacons and goals, based on the perceived radius of the ball in pixels (Equation 3.6). The constants are again calculated by fitting a curve to a number of measured samples. Heading and elevation within the image are translated and rotated through the robot's stance and camera geometry to give a position for the ball.

$$dist = \frac{1103.94}{radius + 0.847627} \tag{3.6}$$

Three checks are performed to ensure that the recognised ball is valid. If it is deemed invalid then no attempt is made to correct it; it is difficult to distinguish between cases were the ball is visible but is misrecognised and cases where the ball is not visible. Two geometry tests are applied: the ball is discarded if it appears above the horizon by more than ten pixels; the ball is also discarded if its radius is unreasonably large (two thousand pixels).

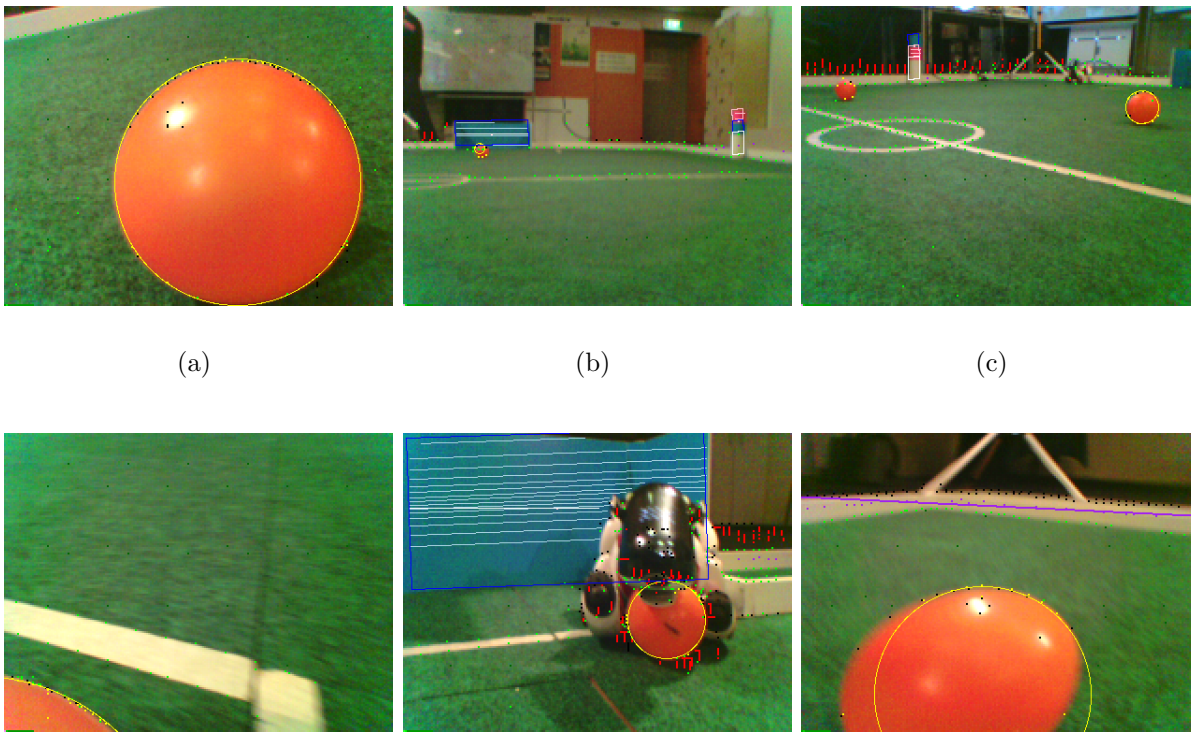
Finally, a symbolic colour based test is applied. A valid ball should contain orange pixels within its circumference. Although it might be occluded or shadowed and consistent orange is not necessary, a minimum amount is required. If the ball is small (a radius smaller than ten pixels) then a square of side length equal to the ball's radius is constructed around the centre of the ball, and all pixels in this square are colour-classified. Otherwise, features that lie within ten pixels of the circumference are randomly chosen and a line segment is constructed between the feature and the ball centroid. The pixels along this line segment are classified, up to a total of one hundred pixels over all features. In both cases counts are maintained of the number of orange, red, pink and green classified pixels encountered.

If fewer than three orange pixels are encountered the ball is discarded. If the radius of the ball is greater than ten pixels and fewer than twelve orange pixels are found the ball is discarded. Experimentation was also performed with checks relating the amount of detected orange to the amounts of pink and red; however, these checks were discarded in favour of robustness to varied lighting conditions. This colour checking is displayed as a cross shape of accessed pixels over the ball in Figure 4.1.

This approach allows accurate recognition of the ball under a range of conditions. While it is limited by an assumption that there is only one ball present in the image, the ball may be detected when blurred or skewed, occluded or only partially in frame. Figure 3.13 shows recognition in a number of these cases. The repeated median algorithm exhibits $\Theta(n^3)$ computational complexity in the number of features. Since n is usually small this remains appropriate; this implementation limits the number of features used to a maximum of seventeen, more than enough to achieve an accurate fit.

Mutual consistency

Once object recognition is complete a number of checks are made to ensure that the perceived objects are mutually consistent. These checks are outlined below, in order of application.



(a) (b) (c)

(d) (e) (f)

Figure 3.13: Ball recognition in a number of different situations. Ample information available in (a) allows for a very tight fit. The repeated median estimator continues to be accurate at long range in (b), although at such distances precise information is less important. Noise features caused by a second ball in (c) are ignored. A ball lying mainly outside the image frame in (d) is still detected accurately, as is a ball partially occluded by a robot in (e). A combination of motion blur and skewing in (f) lead to a questionable fit.

- The two goals cannot be simultaneously perceived. If they are, the one comprising the fewest features is discarded. If they have the same number of features the goal with the highest elevation is discarded.
- A goal centroid cannot appear inside a beacon. If it does, the goal is discarded.
- A beacon centroid cannot appear inside a goal. If it does, the beacon is discarded.
- A goal cannot appear above a beacon by more than ten degrees. If it does, the goal is discarded.
- Diagonally opposite beacons cannot simultaneously be observed. If they are, both are discarded.
- Beacons at the same end of the field cannot appear within thirty degrees of each other. If they are, both are discarded.
- The ball cannot appear above a goal. If it is, the ball is discarded.
- The ball cannot appear above a beacon. If it is, the ball is discarded.

These checks conclude visual processing for one frame. The information extracted from the recognised features and objects is passed to the localisation and behaviour modules.

3.3 Summary

This chapter has outlined in detail the theory and implementation of a new approach to robot vision. This implementation is based on colour-space relationships between neighbouring pixels from high information image areas, and object recognition from these sparsely detected features. While advantage is taken of the information colour segmentation has to offer, the focus is on colour gradients between pixels, which remain stable under variations in lighting. The sub-sampling approach focusses primarily on areas in each image likely to contain useful information,

sampling low information areas only sparsely. This approach has led to methods for object recognition from a discrete set of features. These methods rely on more detailed information about an object's expected appearance and utilise far fewer of the validity tests common to blob-based object recognition approaches.

Chapter 4

Evaluation

As discussed in section 2.3, quantitative evaluation of a robot vision system is difficult. This approach, like many others, is designed to perform as well as possible in one general environment, here the competitive environment of the RoboCup four-legged league. In one sense this real-world performance is the most important evaluation measure.

The *rUNSWift* team using this system placed first in the RoboCup 2005 Australian Open and third in the RoboCup 2005 four-legged league World Championships. The world champion GermanTeam also use elements of sub-sampling and colour relationships [18], so these methods are clearly a valid and successful approach to four-legged league vision. While it is extremely difficult to directly compare the accuracy and robustness of different vision systems, this chapter presents some results pertinent to this approach.

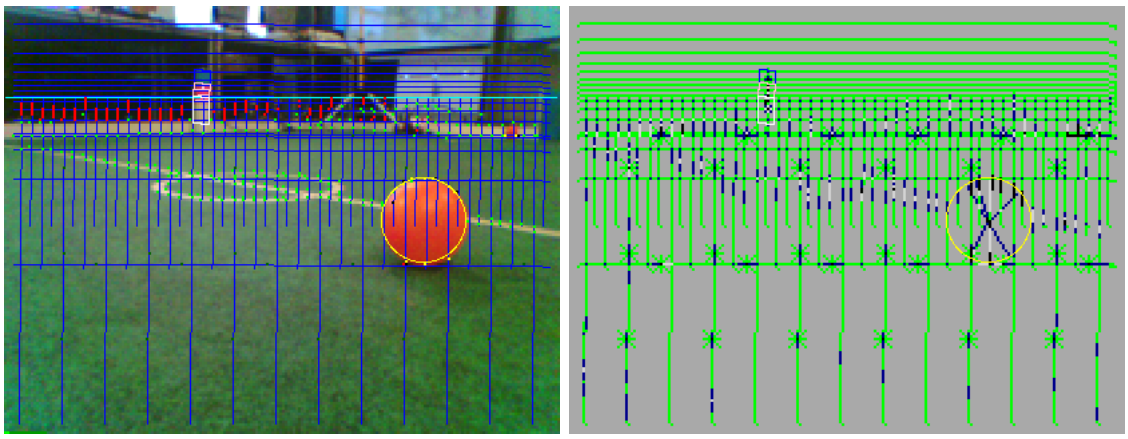
4.1 Colour relationships

A focus on the relationships between neighbouring pixels, rather than symbolically classified colour, leads to a great improvement in robustness. As demonstrated in Figures 3.1 and 3.2, variations in lighting quickly lead to the degradation of static colour classifications, while colour-

space relationships remain far more constant. This was confirmed by the ability of the system presented here to perceive the ball and field lines in a range of environments without re-calibration. In fact, the colour-difference vectors in this implementation were not modified after their initial calculation despite large changes in lighting intensity, colour temperature and even camera settings across four different field environments in which the implementation was tested.

This system was also used without significant modification for the four-legged league Variable Lighting Challenge, in which *rUNSWift* placed third in the world. The *rUNSWift* agent was able to perceive the ball for the majority of the challenge. However, when lighting was made *brighter* than normal game conditions the agent mistook a robot in red uniform for the ball, and became fixated upon this. Excessively bright lighting had not been extensively tested and the distinction between the ball and red team uniform is stable under darker lighting. During a trial run in the *rUNSWift* laboratory (where lighting varied only darker than normal game conditions), Michael Quinlan, leader of the Newcastle University team *NUbots*, commented "this is the first system I've seen that can really chase a ball under variable lighting conditions". These performances demonstrate robustness to dynamically variable lighting.

The reduced reliance on colour segmentation has also led to reduced complexity of and effort required for colour segmentation. Coupled with the interactive classification tool presented by [15] and the efficiency and flexibility of the kernel classifier, colour segmentation can be performed with more sample data and with less effort than previous approaches. The sub-sampling system is robust to a sloppy segmentation, although in a competitive environment effort is always made to achieve a stable classification. The final rounds of the RoboCup 2005 four-legged league competition were performed with a segmentation based on over one thousand images, trained in a few hours of human expert time and tested only in the few hours before the competition games.



(a) (b)

Figure 4.1: Pixel access profiling for a typical image. (a) shows the original YCbCr image with scan lines, features and recognised objects displayed. A pixel access profile is shown in (b). Dark grey represents pixels that have not been accessed, green represents pixels that have been accessed once, other colours represent pixels accessed multiple times. Note how access is clustered around high-information areas of the image such as the ball, field lines and beacon.

4.2 Sub-sampling

Focussing image access on regions of high informational value leads to efficiency gains, as time is not spent processing redundant information within uniformly coloured regions. Domain knowledge allows targeting of high information areas for dense sampling, while regions of typically low information may be sampled more sparsely. A dynamic element to image sampling allows even more directed processing based on the information gained in earlier stages. Pixel access to a typical image is shown in Figure 4.1, where it can be seen that areas containing useful information are sampled with higher density than less informative regions. The scan line pattern typically covers only twenty percent of the pixels in an image, with dynamic access processing a little more depending on the information gained in the initial pass.

Actual processor time consumed by this approach is similar to previous methods, typically ranging from 10ms - 15ms, depending on image content. There are a number of reasons why this is not significantly lower. Firstly, the scan line pattern describes essentially random access

to the image. While approaches that sample the entire image may make good use of processor caching and similar optimisations, these are lost when the image data is accessed in this pattern; every pixel access potentially generates a cache miss. Similarly, image correction and colour classification are applied on the fly to pixels as they are accessed, giving random access to correction and classification tables. Disabling chromatic distortion correction leads to a small gain in efficiency. Removing all colour classification-based access leads to large gains: earlier versions of this system that eschewed colour segmentation altogether processed images in under 5ms, but lacked good beacon and goal detection.

4.3 Object recognition

Moving to a sub-sampled approach makes traditional blobbing approaches to object recognition impossible. Instead, more sophisticated object recognition procedures are required to form objects from sparsely detected features. A significant advantage to this approach is the reduction in complex, hand-coded validity tests for objects. [13] explicitly listed thirty such checks involved in blob-based detection of the beacons, goals and ball for *rUNSWift* in 2004, but the code used at the 2004 RoboCup competition contained many, many more and spanned thousands of lines of C++. The fifteen validity tests outlined in section 3.2.6 represent the entirety of such checks in this implementation; in general this approach requires far fewer domain-specific tests. This leads to more efficient object recognition, allowing for more sophisticated and computationally expensive approaches to obtaining accurate information.

4.4 Shortcomings

As noted above, while the efficiency of the sub-sampling system as implemented is well within acceptable ranges for the four-legged league, improvements may be made by optimisations to

the image access pattern or a reduction in colour-based tests. Random access to both the image data and correction and classification tables make poor use of processor caching features.

The beacon detection implemented in this approach leaves room for improvement. Being colour-based, it has all the deficiencies associated with reliance on a finely tuned static colour segmentation, but makes use of less information than other approaches. While the accuracy was adequate for our purposes there are gains to be made in more accurately fitting the beacon model to the available data. However, it is likely that the coloured beacons will be removed from the four-legged league field definition in the near future as the league attempts to move towards yet more realistic environments.

Edge-based methods in general respond poorly to blurred images, which are not uncommon for legged robots. Significantly blurred images, such as those obtained while contesting positions with robots from the opposing team, are poorly processed. Section 5.1.1 suggests some possible improvements.

4.5 Discussion

The successful results obtained by the sub-sampled approach presented in this report outline a path to more robust, lighting-independent robot vision. While there is still much work to be done, significant improvements in robustness have been gained by a shift of focus away from statically classified colour towards detection of colour relationships and transitions. Unlike approaches based on selection between multiple colour tables this approach gracefully caters for unexpected conditions without the need for additional calibration efforts. In contrast to existing dynamic classification approaches, this implementation allows for potentially arbitrary complexity in colour and colour-gradient classification without the need for adjustments calculated from past observations.

A change in the image access method from processing the entire image and grouping related pixels, to sampling only a small number in important regions, has also met with success. Whilst

performance gains are limited by the remaining colour based methods and poor use of hardware optimisations such as caching, this approach leads to more sophisticated object recognition techniques and robustness to much of the background information that necessitates extensive validity checks in blob-based approaches.

These changes in focus are likely to be applicable to other robotic vision domains where uniform colour is a primary differentiator for important objects. It is immediately applicable to other RoboCup leagues, and to other domains requiring robust object recognition under tight constraints on efficiency.

Chapter 5

Conclusion

Vision is an important sense for robots, but the high level of detail and dependence upon environmental conditions makes creation of an accurate, robust and efficient robot vision system a complex task. The goal of this report is to describe and justify the implementation of a new vision system for the RoboCup four-legged league. Rather than a traditional focus on colour segmentation of entire images, this approach moves towards detection of local relationships between a subset of the image pixels. While systems based on colour segmentation are brittle and respond poorly to variations in lighting, the relationships between colours exhibit independence to lighting conditions, leading to a more robust system.

5.1 Achievements

This report has demonstrated the stability of colour relationships under variations in lighting, particularly intensity. Unlike approaches that focus on object recognition from colour segmented images, this implementation continues to provide reliable information under a range of environments and variations in lighting. This leads to a more robust feature detection system. A

reduced dependence on colour segmentation also leads to reduced calibration time and eases the transition between different environments.

This report has also demonstrated the effectiveness of sub-sampled image processing approaches. The information contained in a typical image is not uniformly distributed over its area; neighbouring pixels are highly correlated. In order to more effectively make use of constrained resources, regions of the image typically high in information content are sampled more densely than areas of typically low informational value. A dynamic element to the sampling allows an even tighter focus on useful regions in any given image. The sub-sampled approach leads to efficiency gains and implicit rejection of much unwanted data.

Given the information provided by a sub-sampled, edge-oriented approach this report has also described robust object recognition from relatively sparse features. In contrast to approaches based on blobbing of segmented colours, object recognition is performed over a discrete set of features corresponding to particular features in the sampled areas of each image. Domain knowledge allows accurate recognition under a range of conditions.

The system described in this report was used in the UNSW/NICTA *rUNSWift* four-legged league team competing in RoboCup 2005. The team placed first in the Australian Open and third in the World Championships of the RoboCup four-legged league.

5.1.1 Further research

While the system as implemented in this report has been successful, a number of areas that may provide fruitful future research have been identified and are outlined here.

In the area of colour segmentation, the major shortcoming of most previous approaches is the limitation of a single classification for each pixel value. Multiple classifications would serve to "loosen" a colour classification procedure, allowing a pixel value to be classified as belonging to more than one colour class. This would somewhat reduce the environmental specificity of traditional colour segmentation methods. A method to assign probabilities to membership of

various classes would provide more information to higher level processing, although this would also re-introduce dependence on the original classification data.

The image access patterns described in this report focus on areas of typically high information, but the concept can be taken further. A focus on dynamic processing, where image access is determined by information obtained by previous processing, could lead to even further gains in efficiency. The scan lines themselves could be sub-sampled, performing some variation of an interval search for transitions, sampling every pixel only around areas containing edges. In addition, temporal awareness might be used to further hone access patterns; areas of recently high information value might be sampled first and more densely than areas of previously little value.

Blurring remains a problem for transition-sensitive techniques. However, consulting the relationships between pixels at greater intervals than immediate neighbours might allow detection of softer edges. A blurred transition between two colours will have a similar profile to a sharp transition if viewed over more widely spaced pixels. Conversely, on sharp transitions more detail is available from the ERS-7 camera than is currently used. The Y channel is sampled at twice the resolution of the chroma channels, and this information might be used to improve the accuracy of the location of detected features.

Perhaps the most clearly beneficial direction for future work is in generalisation of the colour gradient space regions used for feature classification. Strong parallels may be drawn between colour segmentation and the classification of transitions, the major difference being that the transitions are invariant under linear shifts in the colour-space. The classification procedure for colour-gradient vectors used in this implementation is of a similar complexity to early colour segmentation routines. Applying present-day colour segmentation methods to gradient classification would likely lead to a great improvement in the accuracy of detected features and further reduce reliance on colour segmentation.

Finally, there are some possibilities for improvement upon the object recognition methods presented in this report. The assumption that there is only one ball, and that it appears

as a circle, limits both the flexibility and robustness of ball recognition. Some variation on feature clustering would serve the dual purposes of allowing recognition of multiple balls and rejection of gross outliers. The repeated median circle estimator is effective but its computational complexity prevents use of abundant features. A combination of a random sample consensus algorithm, similar to that used for wall detection, and an optimisation step such as least squares approximation may prove more efficient and allow fitting of more general ellipsoids.

5.2 Conclusions

The robot vision approach presented in this report validates a shift of focus from colour recognition to detection and classification of relationships between neighbouring image regions. The gains in robustness by this approach are significant, leading to improved performance under variable conditions and a reduction of dependence on environment-specific calibrations.

This report also confirms the validity of sub-sampled approaches to image processing. Directing processing time to areas of dense information rather than uniformity gives rise to efficiency gains through reduced processing of redundant data.

Finally, given sparse features detected by a sub-sampled approach, this report details a robust object recognition system based on domain knowledge. The approach is simpler conceptually and in implementation than the complex checks and balances required by blob-based object recognition systems.

Bibliography

[1] J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '00)*, volume 3, pages 2061–2066, October 2000. 31

[2] J. Brusey and L. Padgham. Techniques for obtaining robust, real-time, colour-based vision for robotics. In *IJCAI-99 Proceedings of the Third International Workshop on Robocup*, Stockholm, Sweden, July 1999. 31, 49

[3] Jared Bunting, Stephan Chalup, Michaela Freeston, Will McMahan, Rick Middleton, Craig Murch, Michael Quinlan, Christopher Seysener, and Graham Shanks. Return of the NUbots: The 2003 NUbots team report. Technical report, Newcastle Robotics Laboratory, The University of Newcastle, 2003. 52

[4] Jared W. Bunting and William C. McMahan. Vision processing for RoboCup 2003. Technical report, The University of Newcastle, 2002. http://murray.newcastle.edu.au/users/students/2002/c3012299/NUbotVision_FYP2002.pdf. 23, 24

[5] Peter J. Burt and Edward H. Adelson. The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, COM-31,4:532–540, 1983. 44

[6] Jin Chen, Eric Chung, Ross Edwards, Nathan Wong, Bernard Hengst, Claude Sammut, and Will Uther. Rise of the AIBOs III - AIBO revolutions. Technical report, The University of New South Wales, 2003. 32, 72

[7] Wei Ming Chen. Honours thesis, The University of New South Wales, 2005. 52

[8] Larry Don Colton. *Confidence and Rejection in Automatic Speech Recognition*. PhD thesis, Oregon Graduate Institute of Science and Technology, 1997. 21

[9] J. Czyz. *Decision fusion for identity verification using facial images*. PhD thesis, Université Catholique de Louvain, 2003. 21

[10] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, 1981. 71

[11] D. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 2003. 19

[12] Matthias Jüngel, Jan Hoffmann, and Martin Lötzsch. A real-time auto-adjusting vision system for robotic soccer. In *7th International Workshop on RoboCup*. Springer-Verlag, 2003. 33, 35

[13] Daniel Lam. Visual object recognition using Sony four-legged robots. Honours thesis, The University of New South Wales, 2004. 23, 25, 34, 50, 69, 80

[14] David M. Mount and Nathan S. Netanyahu. Efficient randomized algorithms for robust estimation of circular arcs and aligned ellipses. *Comput. Geom. Theory Appl.*, 19(1):1–33, 2001. 71, 72

[15] Kim Cuong Pham. Incremental learning of vision recognition using ripple down rules. Honours thesis, The University of New South Wales, 2005. 15, 34, 50, 78

[16] Michael J. Quinlan, Craig L. Murch, Timothy G. Moore, Richard H. Middleton, Lee Andy Yung Li, Robert King, and Stephan K. Chalup. The 2004 NUbots team report. Technical report, Newcastle Robotics Laboratory, The University of Newcastle, 2004. 33

[17] L. G. Roberts. Machine perception of three-dimensional solids. *Optical and Electro-Optical Information Processing*, pages 159–197, 1965. 39

[18] Thoman Röfer, Tim Laue, Hans-Dieter Burkhard, Jan Hoffmann, Matthias Jüngel abd Daniel Göhring, Martin Lötzsch, Uwe Düffert, Michael Spranger, Benjamin Altmeyer, Viviana Goetzke, Oskar von Stryk, Ronnie Brunn, Marc Dassler, Michael Kunz, Max Risler, Maximilian Stelzer, Dirk Thomas, Stefan Uhrig, Uwe Schweigelshohn, Ingo Dahm, Matthias Hebbel, Walter Nistico, Carsten Schumann, and Michael Wachter. GermanTeam 2004. Technical report, Universität Bremen, Humboldt-Universität zu Berlin, Technische Universität Darmstadt, University of Dortmund, 2004. 15, 24, 32, 35, 52, 62, 77

[19] Joshua Shammay. Real-time shared obstacle probability grid mapping and avoidance for mobile swarms. Honours thesis, The University of New South Wales, 2005. 62

[20] Andrew Sianty. Honours thesis, The University of New South Wales, 2005. 61

[21] Mohan Sridharan and Peter Stone. Towards illumination invariance in the legged league. *Lecture Notes in Computer Science*, 3276:196–208, January 2005. 33

[22] Peter Stone, Kurt Dresner, Peggy Fidelman, Nicholas K. Jong, Nate Kohl, Gregory Kuhlmann, Mohan Sridharan, and Daniel Stronger. The UT Austin Villa 2004 RoboCup four-legged team: Coming of age. Technical report, Department of Computer Sciences, The University of Texas at Austin, 2004. 35, 49

[23] Matthew Turk. Computer vision in the interface. *Commun. ACM*, 47(1):60–67, 2004. 19

[24] Manuela Veloso, Sonia Chernova, Douglas Vail, Scott Lenser, Colin McMillen, James Bruce, Francesco Tamburrino, Juan Fasola, Matt Carson, and Alex Trevor. CMPack'04: Robust modeling through vision and learning. Technical report, Carnegie Mellon University, 2004. 15, 32, 35

[25] Zbigniew Wasik and Alessandro Saffiotti. Robust color segmentation for the RoboCup domain. In *16th International Conference on Pattern Recognition*, volume 2, 2002. 33

[26] Derrick Whaite. Thesis report. Honours thesis, The University of New South Wales, 2004. 35

[27] Jing Xu. 2004 rUNSWift thesis – low level vision. Honours thesis, The University of New South Wales, 2004. 23, 31, 33, 50

[28] Technical challenges for the RoboCup 2005 legged league competition. http://www.tzi.de/4legged/pub/Website/Downloads/Challenges2005.pdf. 28

[29] TecRams team report, RoboCup 2004, 2004. 31

Appendix A

Code listings

Code listings for three critical portions of the implementation described in Chapter 3 are given here. Appendix A.1 presents code used for the construction of the scan lines defining the initial image access pattern. Appendix A.2 presents the code used to detect ball, field line and obstacle features on each pixel along a scan line. Appendix A.3 presents the code implementing the state machine which finds ball edge features based on colour segmentation.

Minor changes have been made to the code as used at RoboCup 2005 in order to provide context or improve readability.

A.1 Scan line construction

```
/* Scan line construction. Note: the origin is the top left corner of the image,
 * so "down" is usually positive y
 */

#include <cmath>
#include <utility> // for std:pair
#include <vector>

using namespace std;
```

```
typedef pair<int, int> point;

// constants
enum {
    // pixels from the edge of the image that are too distorted to use
    IMAGE_EDGE = 3,
    // pixels horizontal space between full scan lines (power of 2)
    FULL_SCANLINE_SPACING = 16,
    // max length of the half-size scanlines
    HALF_SCANLINE_LENGTH = 4 * FULL_SCANLINE_SPACING,
    // max length of the quarter-size scanlines
    QTR_SCANLINE_LENGTH = 3 * FULL_SCANLINE_SPACING,
    // minimum scan line length (pixels)
    MIN_SCANLINE = 3,
    // max distance above horizon to put scanlines. We ignore goals
    // much lower down in sanity checks anyway.
    MAX_HORZ_SCANLINE_GAP = 50, // gives 10 scanlines above horizon
    // Max number of horizontal scanlines (when grabbed)
    MAX_HORZ_SCANLINES = 30,
    // dimensions of image
    CPLANE_WIDTH = 208, CPLANE_HEIGHT = 160,
};

/* Given a straight line in point-gradient form and a y value, returns the
 * x value of that intercept, rounded inwards where necessary.
 */
template<typename T>
inline T solveForY(const pair<T, T>& p, double gradient, T yIntercept) {
    return static_cast<T>((yIntercept - p.second) / gradient + p.first);
}

/* Given a straight line in point-gradient form and an x value, returns the
 * y value of that intercept, rounded inwards where necessary.
 */
template<typename T>
inline T solveForX(const pair<T, T>& p, double gradient, T xIntercept) {
    return static_cast<T>(gradient * (xIntercept - p.first) + p.second);
}

/* Returns the direct distance between two points */
inline double length(const point& p, const point& q) {
    return sqrt((double)(p.first - q.first)*(p.first - q.first)
            + (p.second - q.second)*(p.second - q.second));
}
```

```
/* Returns the -1 if x is negative, else 1 */
template <class T>
inline static int SIGN(const T &x) {
        return (x < 0 ? -1 : 1);
}

// Function declarations
point makeScanline(point hIntercept, int len = INT_MAX);
pair<point, point> makeHorzScanline(point centrePoint);

// These are calculated earlier, representing the intercepts of the
// horizon with the image border
extern point horizonPoint, otherHorizonPoint;

// slope of the horizon
extern double horizonGradient;

// true up in the image is actually down in the world
extern bool horizonUpsideDown;

// "vertical" scan lines, pairs of two (x,y) points, the first of each pair
// being on the horizon
vector<pair<point, point> > scanlines;
// low "horizontal" scanlines used for ball/line detection
vector<pair<point, point> > leftScanlines;

// "horizontal" scan lines used for beacons and goals
vector<pair<point, point> > horzScanlines;

/* Constructs scan lines over the image based on the current horizon estimate.
 * Scan lines are created as pairs of points in scanlines, leftScanLines
 * and horzScanLines
 */
void createScanLines(void) {
    // Drop a scan line from the centre of the horizon
    point horizonCentre;
    if (fabs(horizonGradient) < 1) {
        horizonCentre.first = horizonPoint.first +
                    (otherHorizonPoint.first - horizonPoint.first) / 2;
        horizonCentre.second = solveForX(horizonPoint, horizonGradient,
                                         horizonCentre.first);
    } else {
        horizonCentre.second = horizonPoint.second +
```

```
                (otherHorizonPoint.second - horizonPoint.second) / 2;
    horizonCentre.first = solveForY(horizonPoint, horizonGradient,
                                    horizonCentre.second);
}
pair<int, int> scanIntercept = makeScanline(horizonCentre);
if (length(horizonCentre, scanIntercept) > 2)
    scanlines.push_back(make_pair(horizonCentre, scanIntercept));

// Drop some more scan lines progressively out from the centre,
// first moving out to the left, then when we've gone too far
// go back and move out to the right
int gap;
int direction = -1; // -1 for left, 1 for right
bool stop = false; // should we stop dropping in this dir
for (gap = FULL_SCANLINE_SPACING/4; ; gap += FULL_SCANLINE_SPACING/4) {
    point horizonIntercept;
    if (fabs(horizonGradient) < 1) {
        horizonIntercept.first = horizonCentre.first + direction*gap;
        horizonIntercept.second = solveForX(horizonPoint, horizonGradient,
                                            horizonIntercept.first);
    } else {
        horizonIntercept.second = horizonCentre.second + direction*gap;
        horizonIntercept.first = solveForY(horizonPoint, horizonGradient,
                                            horizonIntercept.second);
    }
    if (gap % FULL_SCANLINE_SPACING == 0)
        scanIntercept = makeScanline(horizonIntercept);
    else if (gap % (FULL_SCANLINE_SPACING/2) == 0)
        scanIntercept = makeScanline(horizonIntercept,HALF_SCANLINE_LENGTH);
    else
        scanIntercept = makeScanline(horizonIntercept,QTR_SCANLINE_LENGTH);

    // if the line's upside down and shouldn't be, stop
    if (! horizonUpsideDown) {
        if (scanIntercept.second < horizonIntercept.second)
            stop = true;
    } else {
        if (horizonIntercept.second < scanIntercept.second)
            stop = true;
    }
    // if the line is too short, stop
    if (length(horizonIntercept, scanIntercept) < MIN_SCANLINE) {
        stop = true;
    }
```

```
    // if both points of a line are on the border or off screen, ignore it.
    if ((horizonIntercept == scanIntercept
            || (horizonIntercept.first < IMAGE_EDGE
            || horizonIntercept.first > CPLANE_WIDTH - 1 - IMAGE_EDGE
            || horizonIntercept.second < IMAGE_EDGE
            || horizonIntercept.second > CPLANE_HEIGHT - 1 - IMAGE_EDGE))
           && (scanIntercept.first < IMAGE_EDGE
                || scanIntercept.first > CPLANE_WIDTH - 1 - IMAGE_EDGE
                || scanIntercept.second < IMAGE_EDGE
                || scanIntercept.second > CPLANE_HEIGHT - 1 - IMAGE_EDGE)) {
        // if it's a long line, stop
       if (gap % FULL_SCANLINE_SPACING == 0) {
            stop = true;
       } else {
           continue;
       }
    }
    if (stop) {
        if (direction == -1) { // back to centre and change direction
            gap = 0;
            direction = 1;
            stop = false;
            continue;
        } else {
            break; // finished
        }
    }
    if (length(horizonIntercept, scanIntercept) > 2)
        scanlines.push_back(make_pair(horizonIntercept, scanIntercept));
}

// Create some sparse horizontal scan lines parallel to the horizon below it
// Note that despite being horizontal these lines are not in the
// horzScanLines list and are not used for beacon/goal detection; they are
// for picking up extra ball points and lines that run away from the robot.
gap = FULL_SCANLINE_SPACING;
while (true) {
    point centrePoint = horizonCentre; // centre of scanline
    if (fabs(horizonGradient) < 1) {
        centrePoint.second = centrePoint.second + gap;
        centrePoint.first = solveForY(horizonCentre, -1/horizonGradient,
                                      centrePoint.second);
        if (centrePoint.second > CPLANE_HEIGHT - 1)
            break;
```

```
    } else if ((horizonGradient < 0 && ! horizonUpsideDown)
            || (horizonGradient > 0 && horizonUpsideDown)) {
        centrePoint.first = centrePoint.first + gap;
        centrePoint.second = solveForX(horizonCentre, -1/horizonGradient,
                                       centrePoint.first);
        if (centrePoint.first > CPLANE_WIDTH - 1)
            break;
    } else {
        centrePoint.first = centrePoint.first - gap;
        centrePoint.second = solveForX(horizonCentre, -1/horizonGradient,
                                       centrePoint.first);
        if (centrePoint.first < 0)
            break;

    }

    pair<point, point> line = makeHorzScanline(centrePoint);
    if (length(line.first, line.second) > 2)
        leftScanlines.push_back(line);
    gap *= 2; // exponential spacing
}

// create some tight horizontal scanlines from just below out to well
// above the horizon. These are for beacons/goals.

// start this far below horizon
gap = (int)(-FULL_SCANLINE_SPACING * 1.35);
int nHorzLines = 0;
while (true) {
    if (nHorzLines++ > MAX_HORZ_SCANLINES)
        break;
    point centrePoint = horizonCentre; // centre of scanline
    if (gap > MAX_HORZ_SCANLINE_GAP) {
        //rarely anything useful above here, so don't process
        break;
    }
    if (fabs(horizonGradient) < 1) {
        centrePoint.second = centrePoint.second - gap;
        centrePoint.first = solveForY(horizonCentre, -1/horizonGradient,
                                      centrePoint.second);
        if (centrePoint.second < 0)
            break;
    } else if ((horizonGradient < 0 && ! horizonUpsideDown)
            || (horizonGradient > 0 && horizonUpsideDown)) {
```

```
            centrePoint.first = centrePoint.first - gap;
            centrePoint.second = solveForX(horizonCentre, -1/horizonGradient,
                                           centrePoint.first);
            if (centrePoint.first < 0)
                break;

        } else {
            centrePoint.first = centrePoint.first + gap;
            centrePoint.second = solveForX(horizonCentre, -1/horizonGradient,
                                           centrePoint.first);
            if (centrePoint.first > CPLANE_WIDTH - 1)
                break;
        }
        pair<point, point> line = makeHorzScanline(centrePoint);
        if (length(line.first, line.second) > 2)
            horzScanlines.push_back(line);
        // slow exponential spacing
        gap += FULL_SCANLINE_SPACING/8 + abs(gap/5);
    }
}

/* Constructs a scan line from the specified point on the horizon "down"
 * perpendicular to the horizon for length pixels, or until it intercepts with
 * an image border. Returns the (x,y) coordinates of the second point.
 */
point makeScanline(point hIntercept, int len) {
    point p;
    double scanGradient;

    scanGradient = -1.0 / horizonGradient;
    // first try solving for y = CPLANE_HEIGHT (or y = 0 if image is upside
    // down). If that's within the image then it's our intercept
    if (horizonUpsideDown)
        p.second = IMAGE_EDGE;
    else
        p.second = CPLANE_HEIGHT - 1 - IMAGE_EDGE;

    p.first = solveForY(hIntercept, scanGradient, p.second);
    if (p.first < IMAGE_EDGE) {
        // no, intercept is with left border
        p.first = IMAGE_EDGE;
        p.second = solveForX(hIntercept, scanGradient, p.first);
    } else if (p.first > CPLANE_WIDTH - 1 - IMAGE_EDGE) {
        // no, intercept is with right border
```

```
        p.first = CPLANE_WIDTH - 1 - IMAGE_EDGE;
        p.second = solveForX(hIntercept, scanGradient, p.first);
    } else {
        // yes, intercept is with bottom/top edge
    }

    // Clip to desired length
    if (length(hIntercept, p) > len) {
        if (fabs(scanGradient) < 1) {
            p.first = hIntercept.first + SIGN(p.first - hIntercept.first) * len;
            p.second = solveForX(hIntercept, scanGradient, p.first);
        } else {
            p.second = hIntercept.second +
                SIGN(p.second - hIntercept.second) * len;
            p.first = solveForY(hIntercept, scanGradient, p.second);
        }
    }
    return p;
}

/* Constructs a scan line from the specified point parallel to the horizon
 * until it intercepts the image borders on either side. Returns the (x,y)
 * coordinates of both points.
 */
pair<point, point> makeHorzScanline(point centrePoint) {
    point p, q;

    // p (leftmost) first. Try solving for x = 0
    if (! horizonUpsideDown) {
        p.first = IMAGE_EDGE;
    } else {
        p.first = CPLANE_WIDTH - 1 - IMAGE_EDGE;
    }
    p.second = solveForX(centrePoint, horizonGradient, p.first);
    if (p.second < IMAGE_EDGE) { // intercept with top edge
        p.second = IMAGE_EDGE;
        p.first = solveForY(centrePoint, horizonGradient, p.second);
    } else if (p.second > CPLANE_HEIGHT - 1 - IMAGE_EDGE) {
        // intercept bottom edge
        p.second = CPLANE_HEIGHT - 1 - IMAGE_EDGE;
        p.first = solveForY(centrePoint, horizonGradient, p.second);
    } // else intercept with left edge

    // now q (rightmost). Try x = CPLANE_WIDTH - 1
```

```
    if (! horizonUpsideDown) {
        q.first = CPLANE_WIDTH - 1 - IMAGE_EDGE;
    } else {
        q.first = IMAGE_EDGE;
    }
    q.second = solveForX(centrePoint, horizonGradient, q.first);
    if (q.second < IMAGE_EDGE) { // intercept with top edge
        q.second = IMAGE_EDGE;
        q.first = solveForY(centrePoint, horizonGradient, q.second);
    } else if (q.second > CPLANE_HEIGHT - 1 - IMAGE_EDGE) {
        // intercept bottom edge
        q.second = CPLANE_HEIGHT - 1 - IMAGE_EDGE;
        q.first = solveForY(centrePoint, horizonGradient, q.second);
    } // else intercept with left edge

    return make_pair(p, q);
}
```

A.2 Feature detection

```
/* Feature detection */

#include <utility> // for std:pair
#include <vector>

using namespace std;

typedef pair<int, int> point;

// constants
enum {
    // pixels from the edge of the image that are too distorted to use
    IMAGE_EDGE = 3,
    // distance to search for orange inwards from a ball feature
    BALL_COLOUR_SEARCH = 5,
    // min pixels of orange in BALL_COLOUR_SEARCH to classify as ballfeature
    MIN_BALL_COLOUR = 3,
    // max pixels of non-orange in BALL_COLOUR_SEARCH to classify as ballfeature
    MAX_NONBALL_COLOUR = 1,
    // dimensions of image
    CPLANE_WIDTH = 208, CPLANE_HEIGHT = 160,
};
```

```
// Visual feature thresholds and constants
enum {
    TH_ANYTHING = 20,          // minimum gradient sum to be at all interesting
    TH_FIELDLINE_DY = 15,      // minimum Y gradient for green/white edge
    TH_FIELDLINE_DUV = 40,     // maximum U/Y gradient for green/white edge
    TH_BALL_DU = 15,           // was 20, min du on orange edge
    TH_BALL_DUDV = 4,          // max fraction of similar du/dv for ball edge

    TH_OBSTACLE_Y = 35,        // min y of non-obstacle/shadow
    TH_SPECULAR_Y = 180,       // y above this on objects is probably specular
    TH_FUZZY = 4,              // a small delta allowing for noise
};

enum visualfeature_type {
    VF_NONE = 0,           // used for pruned features
    VF_FIELDLINE,          // a green/white or white/green transition
    VF_BALL,               // a ball edge (x/orange transition)
    VF_OBSTACLE,           // an obstacle or noise
    VF_FIELD,              // pure field green
    VF_PINK,               // beacon pink run of pixels
    VF_BLUE,               // beacon blue run of pixels
    VF_YELLOW,             // beacon yellow run of pixels
    VF_WALL,               // white wall run of pixels
    VF_WALL_OBJECT,        // wall object hacked as visual feature
    VF_UNKNOWN,            // something we can't classify
};

/* A visual feature is something recognised by feature detection, usually an
 * edge or colour transition. Features have information about the colours
 * involved and their location in the frame.
 */
class VisualFeature {
    public:
        /* Create a new VisualFeature with coords x, y */
        VisualFeature(visualfeature_type tt = VF_NONE, double xx = 0.0,
                      double yy = 0.0, int ddy = 0, int ddu = 0,
                      int ddv = 0, int dd = 0)
            : x(xx), y(yy), dy(ddy), du(ddu), dv(ddv), dsum(ddy + ddu + ddv),
              dir(dd), type(tt) {scanline = 0; vertical = 0;}

        ~VisualFeature() {}

        /* Members. These are all public for convenience since this class
         * is only for storing related information.
```

```
     */
    double x, y;          // position in image (subpixel)
    double endx, endy;    // position of the end of a run (beacons and goals)
    double dy, du, dv;    // colour gradients
    double dsum;          // sum of colour gradients
    int scanline;         // index of scanline (beacons/goals)
    // direction of scan line (DIR_UP/DOWN/LEFT/RIGHT). In the case of ball
    // and line features it is the direction of the ball/line.
    int dir;
    visualfeature_type type;
    int vertical;         // vertical distance from the bottom the scanline
};

/* Tests if this pixel -- at coordinates given by p and with YUV values
 * of yy, uu, vv -- is interesting relative to the previous one. Returns
 * a VisualFeature, with type VF_NONE if it's not interesting.
 * Interesting things are field line edges, ball edges and
 * obstacles (shadows). The direction of scanning is given by dir,
 * either DIR_UP or DIR_RIGHT. This should be executed once for each
 * pixel in a scan line.
 *
 * If initialiseOnly is true VF_NONE is returned; use this for the first pixel
 * on each scan line.
 */
VisualFeature testPixel(point p, unsigned char yy, unsigned char uu,
                        unsigned char vv, int dir, bool initialiseOnly) {

    visualfeature_type type = VF_NONE;
    static int prevY = 0, prevU = 0, prevV = 0; // previous YUV

    int x = p.first, y = p.second;
    if (x < 0 || x >= CPLANE_WIDTH || y < 0 || y >= CPLANE_HEIGHT)
        initialiseOnly = true;

    // We don't use pixels in the corners of the image
    // because they are too noisy
    int cornerDist = 3 * IMAGE_EDGE; // pix
    if ((x < cornerDist && y < cornerDist)
            || (x < cornerDist && y > CPLANE_HEIGHT - 1 - cornerDist)
            || (x > CPLANE_WIDTH - 1 - cornerDist && y < cornerDist)
            || (x > CPLANE_WIDTH - 1 - cornerDist
                    && y > CPLANE_HEIGHT - 1 - cornerDist)) {
        initialiseOnly = true;
    }
```

```
if (initialiseOnly) {
    prevY = -1; prevU = -1; prevV = -1;
    return VF_NONE;
}

// Y, U and V point gradients
int dy = yy - prevY;
int du = uu - prevU;
int dv = vv - prevV;
int sum = abs(dy) + abs(du) + abs(dv);

// This unusual control structure functions like a goto without
// actually using one. The reason is that there is some finalisation
// that needs to happen at the end of each call in setting the prevX
// values, regardless of the return value, so we can't return
// immediately. So once a point has been classified (or failed to be)
// we break from the loop. The loop only ever executes once.
while (true) {
    // The pixel tested is the beginning of the scanline,
    // so ignore it. Just update prevY, prevU, prevV.
    if (prevY == -1) {
        break;
    }

    // For obstacles we search for the low-Y level of shadows...
    if (yy < TH_OBSTACLE_Y) {
        type = VF_OBSTACLE;
        // but don't break. Allow the point to be classified an edge
        // point below
    }

    // If the sum of changes is too small, quit now
    if (sum < TH_ANYTHING) {
        break;
    }

    // Ball feature transitions. A something/orange transition has a large
    // du but not necessarily much dy
    if (abs(du) > TH_BALL_DU                    // minimum du
            && (dv == 0 || abs(du/dv) < TH_BALL_DUDV
                || SIGN(du) == -SIGN(dv)) // dv opp du
            // not specular reflection
            && yy < TH_SPECULAR_Y && prevY < TH_SPECULAR_Y) {
```

```
    // If this edge is leaving a ball, reverse dir to point back
    // towards the orange
    if (du < 0) {
        if (dir == DIR_UP)
            dir = DIR_DOWN;
        else if (dir == DIR_DOWN)
            dir = DIR_UP;
        if (dir == DIR_LEFT)
            dir = DIR_RIGHT;
        else if (dir == DIR_RIGHT)
            dir = DIR_LEFT;
    }

    // Check that there are some orange pixels
    // towards the ball centre
    int ballColour;
    ballColour = dirColour(x, y, dir, cBALL, BALL_COLOUR_SEARCH);
    if (ballColour >= MIN_BALL_COLOUR
            && dirColour(x, y, dir, cROBOT_RED, BALL_COLOUR_SEARCH)
                <= MAX_NONBALL_COLOUR
            && dirColour(x, y, dir, cBEACON_PINK,BALL_COLOUR_SEARCH)
                <= MAX_NONBALL_COLOUR
            && dirColour(x, y, dir, cBEACON_YELLOW,
                        BALL_COLOUR_SEARCH) < MAX_NONBALL_COLOUR) {

        type = VF_BALL;
        break;
    }
}

// Field lines. A white/green transition has most of its change in Y
if (abs(dy) > TH_FIELDLINE_DY                // minimum dy
        && abs(du) < TH_FIELDLINE_DUV        // maximum du and v
        && abs(dv) < TH_FIELDLINE_DUV
        && (abs(du) < TH_FUZZY || SIGN(du) == SIGN(dy)))// du follows dy
{
    // If we don't already think this is an obstacle (e.g. robot
    // leg) then it's a line
    if (type != VF_OBSTACLE)
        type = VF_FIELDLINE;
    // If this edge is leaving a line, reverse dir to point back
    // towards the white
    if (dy < 0) {
        if (dir == DIR_UP)
```

```
                    dir = DIR_DOWN;
                else if (dir == DIR_DOWN)
                    dir = DIR_UP;
                if (dir == DIR_LEFT)
                    dir = DIR_RIGHT;
                else if (dir == DIR_RIGHT)
                    dir = DIR_LEFT;
            }
            break;
        }
        type = VF_UNKNOWN;
        break;
    } // while(true)

    prevY = yy;
    prevU = uu;
    prevV = vv;

    // The feature coordinates are in the middle of the edge between pixels
    // that gave the biggest change. This assumes vertical lines are
    // scanned upwards (decreasing y), horizontal scanned left to right
    // (increasing x), and image coords refer to the centre of their pixel
    if (dir == DIR_UP) {
        return VisualFeature(type, x, y + 0.5, dy, du, dv, dir);
    } else { // DIR_RIGHT
        return VisualFeature(type, x - 0.5, y, dy, du, dv, dir);
    }

    return VisualFeature(); // default VF_NONE
}
```

A.3 Symbolic colour transition detection

```
/* Symbolic colour ball feature detection */

#include <utility> // for std::pair

using namespace std;

typedef pair<int, int> point;

// constants
enum {
```

```
    // number of (not consec) orange pix required for testBallColourTransition
    MIN_ORANGE_TRANSITION = 6,
    // min number of consecutive orange pixels to count as orange found
    // in testBallColourTransition
    MIN_CONSEC_ORANGE_TRANSITION = 3,
    // number of non-orange pix required to reset testBallColourTransition
    MIN_NOTORANGE_TRANSITION = 3,
};

typedef enum {
        cBALL = 0,
        cBEACON_BLUE = 1,
        cBEACON_GREEN = 2,
        cBEACON_YELLOW = 3,
        cBEACON_PINK = 4,
        cROBOT_BLUE = 5,
        cROBOT_RED = 6,
        cFIELD_GREEN = 7,
        cROBOT_GREY = 8,
        cWHITE = 9,
        cBLACK = 10,
        cFIELD_LINE = 11,       // marker, not a real colour
        cFIELD_BORDER = 12,     // ditto
        cNUM_COLOR = 13,
        cNONE = 127
}
Color;

/* An image point fo caching associated colour and classification information.*/
struct classifiedPoint {
    int x; // image position
    int y;
    unsigned char yy; // YUV values
    unsigned char uu;
    unsigned char vv;
    Color cc; // classified colour value

    classifiedPoint(int ax, int ay) {
        x = ax;
        y = ay;
    }

    void set(int ax, int ay, unsigned char ayy, unsigned char auu,
            unsigned char avv, Color acc) {
```

```
        x = ax;
        y = ay;
        yy = ayy;
        uu = auu;
        vv = avv;
        cc = acc;
    }
};

// true if the robot's head is currently down low while grabbing the ball
extern bool isGrabbing;

/* This state machine detects transitions from runs of orange classified points
 * to runs of non-orange classified points, and vice-versa. These are likely
 * ball edges that are possibly too blury or red to pick up in edge detection.
 * The state is reset if initaliseOnly is true.
 */
void testBallColourTransition(point p, unsigned char yy, unsigned char uu,
                              unsigned char vv, Color cc,
                              int dir, bool initialiseOnly) {
    // point start non-orange
    static classifiedPoint startNotOrange = classifiedPoint(-1,-1);
    static classifiedPoint endNotOrange = classifiedPoint(-1,-1);
    // point start and end orange
    static classifiedPoint startOrange = classifiedPoint(-1,-1);
    static classifiedPoint endOrange = classifiedPoint(-1,-1);
    // point start and end maybe-orange
    static classifiedPoint startMaybeOrange = classifiedPoint(-1,-1);
    static classifiedPoint endMaybeOrange = classifiedPoint(-1,-1);

    // number of detected transitions in this scanline
    static int countDetected = 0;
    // pointer to last transition feature
    static VisualFeature* lastTransitionFeature = 0;

    // count of white pix
    static int countWhite = 0;
    // count of orange pix
    static int countOrange = 0;
    // count of non-orange pix
    static int countNotOrange = 0;
    // count of maybe-orange pix
    static int countMaybeOrange = 0;
    // count of consecutive orange pix
```

```
static int countConsecOrange = 0;

// count since the last white and green pix
static int sinceGreen = 0, sinceWhite = -1;

// true when we get a run of orange (possibly separated by maybeOrange)
static bool orangeFound = false;
// true when we get a run of non-orange
static bool notOrangeFound = false;
// true when we get a run of maybe-orange
static bool maybeOrangeFound = false;
// true when we get a run of consecutive orange
static bool consecOrangeFound = false;
// true when we find a transition
static bool transitionFound = false;

// true if orange was found after non-orange, else false
static bool transitionToOrange = false;

if (initialiseOnly) {
    countDetected = 0;
    lastTransitionFeature = 0;

    orangeFound = false;
    maybeOrangeFound = false;
    notOrangeFound = false;
    consecOrangeFound = false;
    transitionFound = false;

    countWhite = 0;
    countOrange = 0;
    countNotOrange = 0;
    countMaybeOrange = 0;
    countConsecOrange = 0;
    sinceGreen = 0;
    sinceWhite = -1;
    return;
}

int x = p.first, y = p.second;

++sinceWhite;
++sinceGreen;
```

```
if (cc == cBALL) {
    if (countOrange == 0) {
        startOrange.set(x,y,yy,uu,vv,cc);
    }

    endOrange.set(x,y,yy,uu,vv,cc);
    countOrange++;
    countConsecOrange++;
    if (countConsecOrange >= MIN_CONSEC_ORANGE_TRANSITION) {
        consecOrangeFound = true;
    }
    if (countOrange >= MIN_ORANGE_TRANSITION && consecOrangeFound) {
        orangeFound = true;
        if (notOrangeFound) { // a transition *into* orange
            transitionFound = true;
            transitionToOrange = true;
            notOrangeFound = false;
        }
        // on a transition into ball (but not out of) we use the start of
        // the maybe-orange, which will be earlier than the orange
        if (countMaybeOrange >= countOrange && dir == DIR_UP) {
            maybeOrangeFound = true;
        }
    }
    // if we found orange and already have two points then at least the
    // last one is decidedly dodgy
    if (countOrange >= MIN_ORANGE_TRANSITION
            && countDetected >= 2 && lastTransitionFeature != 0) {
        lastTransitionFeature->type = VF_NONE;
        --countDetected;
    }

    countNotOrange = countWhite = 0;
    // don't reset countMaybeOrange
} else if (cc == cROBOT_RED || cc == cBEACON_PINK
        || cc == cBEACON_YELLOW) {
    if (countMaybeOrange == 0) {
        startMaybeOrange.set(x,y,yy,uu,vv,cc);
    }
    endMaybeOrange.set(x, y, yy, uu, vv, cc);
    countMaybeOrange++;
    countConsecOrange = 0;
    countNotOrange = countWhite = 0;
    // don't reset countOrange
```

```
} else {
    //cBEACON_BLUE cROBOT_BLUE cFIELD_GREEN cWHITE cBLACK cNONE cROBOT_GREY
    if (cc == cWHITE) {
        countWhite++;
        if (countWhite > 1)
            sinceWhite = 0;
    } else {
        countWhite = 0;
    }
    if (cc == cFIELD_GREEN) sinceGreen = 0;

    if (countNotOrange == 0) {
        startNotOrange.set(x, y, yy, uu, vv, cc);
    }
    endNotOrange.set(x, y, yy, uu, vv, cc);

    countNotOrange++;
    countConsecOrange = 0;
    if (countNotOrange >= MIN_NOTORANGE_TRANSITION) {
        // if this is the end of and orange run remember as such
        notOrangeFound = true;
        if (orangeFound) {  // transition *out of* orange
            transitionFound = true;
            transitionToOrange = false;
            orangeFound = false;
        }
        // we don't use maybeOrange on a transition out of the ball
        // since that is frequently confused with red robot etc

        // reset maybe/orange counts
        countOrange = countMaybeOrange = 0;
        // Need to find another consec orange to get another point
        consecOrangeFound = false;
    }
}

// If the transition distance was less than 6 pixels, then consider mid
// point as a ball edge.
if (transitionFound) {
    classifiedPoint start(-1, -1), end(-1, -1);

    if (transitionToOrange) {
        start = endNotOrange;
        if (maybeOrangeFound) {
```

```
            end = startMaybeOrange;
        } else {
            end = startOrange;
        }
    } else {
        start = endOrange;
        end = startNotOrange;
    }
    double distsqr = DISTANCE_SQR(start.x, start.y, end.x, end.y);

    if (distsqr >= SQUARE(6)) {
        return;
    }

    int dy, du, dv;
    double midx, midy;

    dy = end.yy - start.yy;
    du = end.uu - start.uu;
    dv = end.vv - start.vv;
    midx = (end.x + start.x) / 2.0;
    midy = (end.y + start.y) / 2.0;

    // reverse dir to point towards orange if necessary
    if (! transitionToOrange) {
        if (dir == DIR_UP)
            dir = DIR_DOWN;
        else if (dir == DIR_DOWN)
            dir = DIR_UP;
        if (dir == DIR_LEFT)
            dir = DIR_RIGHT;
        else if (dir == DIR_RIGHT)
            dir = DIR_LEFT;
    }

    // Transitions above white without intervening green are highly
    // spurious and likely robots or beacons. Ignore this check when
    // grabbing
    if (!isGrabbing && sinceWhite < sinceGreen) {
        // Mark this edge point as void, for debugging
        VisualFeature vf = VisualFeature(VF_NONE, midx, midy, dy, du, dv, dir);
        features.push_back(vf);
        countDetected++;
    } else {
```

```
            VisualFeature vf = VisualFeature(VF_BALL, midx, midy, dy, du, dv, dir);
            features.push_back(vf);
            lastTransitionFeature = &features.back();
            countDetected++;
        }

        // reset the state.
        orangeFound = false;
        maybeOrangeFound = false;
        notOrangeFound = false;
        consecOrangeFound = false;
        transitionFound = false;

        countOrange = 0;
        countMaybeOrange = 0;
        countNotOrange = 0;
        countWhite = 0;
        countConsecOrange = 0;
        sinceGreen = 0;
        sinceWhite = -1;
    }
}
```