rUNSWift 2011 Vision System:
A Foveated Vision System for Robotic Soccer

Carl Chatfield (z3255311)
Bachelor of Computer Science

Supervisor: Bernhard Hengst
Assessor: Maurice Pagnucco

August 25, 2011

School of Computer Science & Engineering
University of New South Wales
Sydney 2052, Australia

Abstract

The vision module used by the 2011 rUNSWift team acts as the primary source of input for its robots during a match of robotic soccer. Like all real time vision systems, the algorithms employed must be fast and robust, however designing such a system for use with the Aldebran Nao used by RoboCup Standard Platform League poses considerable challenges. The severe limitations of the available hardware must be overcome using innovative methods that may have otherwise remained unexplored. In particular, we achieved a major speed-up using an intelligent ball tracking "fovea", allowing us to reduce the number of processed pixels whilst preserving detection accuracy. In this context, the concept of a "fovea" is defined as a rectangle of the image sampled at an arbitrary resolution and is an underpinning feature of our design philosophy. The performance of the foveated ball detector has been statistically analysed and compared to that of last year. Such detailed analysis is a first for rUNSWift, and has revealed an unnoticed fruit in the previous year's system.

Contents

Chapter 1

Introduction

RoboCup is an international robotics competition with a stated ultimate goal:

> "By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup." [11]

To those involved, robotic soccer represents the next major research milestone in the field of artificial intelligence after IBM's Deep Blue famously defeated Garry Kasparov, the then reigning world chess champion. [10] [12]

rUNSWift, the University of New South Wales' (UNSW) entrant into this competition, competes in the Standard Platform League (SPL). The league is unique in that all SPL teams compete using teams of identical Aldebran Naos, leaving only software to distinguish the competition's victor.

In 2010, the team deemed it necessary to execute a complete rewrite of the rUNSWift code base [14], granting themselves complete freedom when redesigning their system's architecture. Their efforts culminated in a fresh code base that was able to prove its metal on the international stage, placing second in the world at the Singapore competition. Although impressive, the system in its youth still lacked several desirable features that could not be implemented simply due to a lack of time. The responsibility of filling these gaps fell to the 2011 team.

The vision module, with three dedicated members assigned to it, received the most attention over 2011. In order to remain competitive with the leading teams, two major features where the required: field line detection [9] and robot detection [13]. Both are non-trivial problems and require significant image processing to solve. With the 2010 system already running maximising its CPU utilisation, it became necessary to rethink and optimise existing components of the vision pipeline before these additions of this scale could be made. The ball detector, for example, has been rewritten to rely on significantly less image preprocessing, thus freeing up resources for vision components.

This thesis explores several issues pertaining to the design of the 2011 rUNSWift vision system.

- Chapter 2 provides a high level overview of the vision system as a whole. We describe a system based on the concept of a fovea: a rectangular region of an image sampled at an arbitrary resolution. Foveae not only reduce the required computation needed to process an image, but also mandate that our algorithms are generic and easily substitutable. These traits are especially desirable for a UNSW RoboCup team which must rapidly develop, sometimes incomprehensible, code only to then experience an almost complete student turnover at the end of the year.

- Chapter 3 discusses the vision testing framework in detail. Although the framework itself was completed, time ran out before it could be incorporated into the continuous integration environment. We had hoped to generate performance metrics to chart progress made over the course of the year, but instead we use the testing framework to statistically compare the performance of the foveated ball detector against the detector of last year. It will also be used to assess the benefits yielded by foveated ball tracking.
- Having discussed the testing framework, Chapter 4 now describes and assesses the foveated ball detector. The ball detector was necessarily rewritten as part of the effort to reduce CPU consumption. The new detector boasts improved detection accuracy whilst requiring significantly less computation. The improvements are backed by statistical evidence, however the analysis also discovered that last year's detector performs surprisingly well on blurry images. Before now, the properties of the 2010 detector had never been compared to the properties of more conventional algorithms, and as a result this resilience to blur was never touted.

Chapter 2

A RoboCup Vision System

2.1 Problem Definition

Identify, in the image space, objects of interest to playing the game of soccer. These include, but are not at all limited to, the following: the ball, goals, field markings, the field itself, and both friendly and opponent robots. The 2011 Vision system attempts to identify all of the aforementioned objects.

The SPL domain in which rUNSWift competes has strict rules that place constraints on our system. In particular, teams are tied to a single hardware platform and must efficiently utilise the robots on board hardware. The rules also strictly define the physical appearance of all objects on the field in terms of both shape and colour. In turn, this allows us to strictly define the demands of our vision system.

2.1.1 Hardware: Aldebaran Nao

SPL soccer is played by teams of competing Naos, robots produced by a French company Aldebaran. As the name *Standard Platform* suggests, the Nao is standard hardware used by all teams and may not be modified.

Of specific interest to the vision module are the two cameras situated on the robot's head, horizontally centred with a vertical offset. One camera is positioned to look directly forwards whilst the other points downwards towards the ground and the robot's feet. Both cameras are capable of snapping 640x480 images at 30 frames per second and have a 58 degree diagonal field of view [2]. Compared to a human, who has a diagonal field of view of about 225 degrees [3], the Nao's field of view is comparatively narrow. The cameras can not be used simultaneously, and therefore the robots field of view is limited to a tiny window that often lacks visual context.

The Nao is also hurdled by its limited processing capabilities. In starch contrast to other real-time vision systems, which may often use dedicated specialised hardware [4], the rUNSWift vision system must run on a 500 megahertz AMD Geode processor [1]. Our system must therefore be extremely economical with its available processing resources.

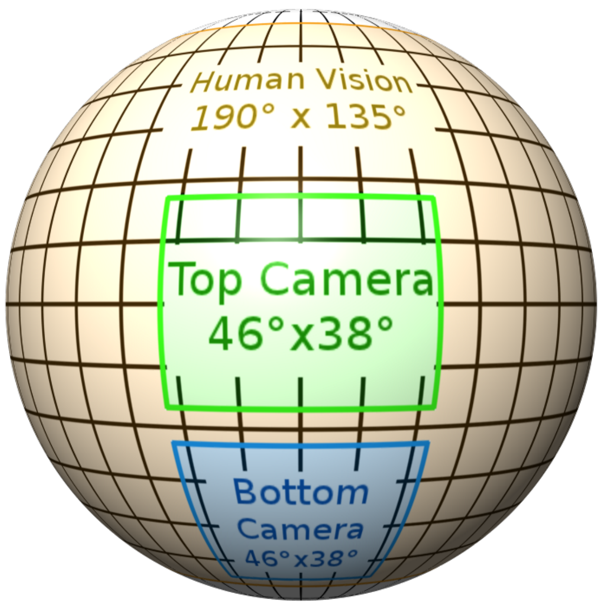


Figure 2.1: Field of View comparison: Human versus Nao

2.1.2 The Field: Colours and Lighting

Playing a competitive game of soccer against a human team is still far beyond the realm of today's computers. To compensate, SPL soccer introduces many simplifications over real soccer; the most significant is the colouring of the environment. Goal posts are colour coded either bright blue or yellow depending on which side of the field they stand. The field itself is a solid matte green, and the ball is a distinctive orange colour. Artificial colouring greatly simplifies the task of writing an SPL vision system, but there remain challenges. Between venues, the shades and reflectivity of materials used vary. The field colour at the Turkey competition, for example, was a bright pale green, whilst the colour used at the Singapore competition was considerably darker.

The other variant is the lighting used to illuminate the field. The SPL rule book states "The lighting conditions depend on the actual competition site. Only ceiling lights may be used [6]." In practice the luminosity of the field is usually around 500 lumen; the types of lighting can range from uniform flood lighting to highly specular point lighting as was the case in Turkey. Whilst not usually noticeable to humans, specular lighting can cause interesting phenomena that change the appearance of objects in an image. Any vision system must be robust enough to handle varying lighting conditions.

2.2 Related Work: rUNSWift 2010 at a glance

Placing second in the world at the 2010 Singapore world championship once again affirmed UNSW's position as a leading university in the competition. Such success can be partially attributed to a well thought out design that has principally remained intact as the basis of the 2011 system. Within the vision module, many ideas pioneered by the 2010 team have been reused, but are now more formally defined in code. A brief overview of each vision component is provided for context.

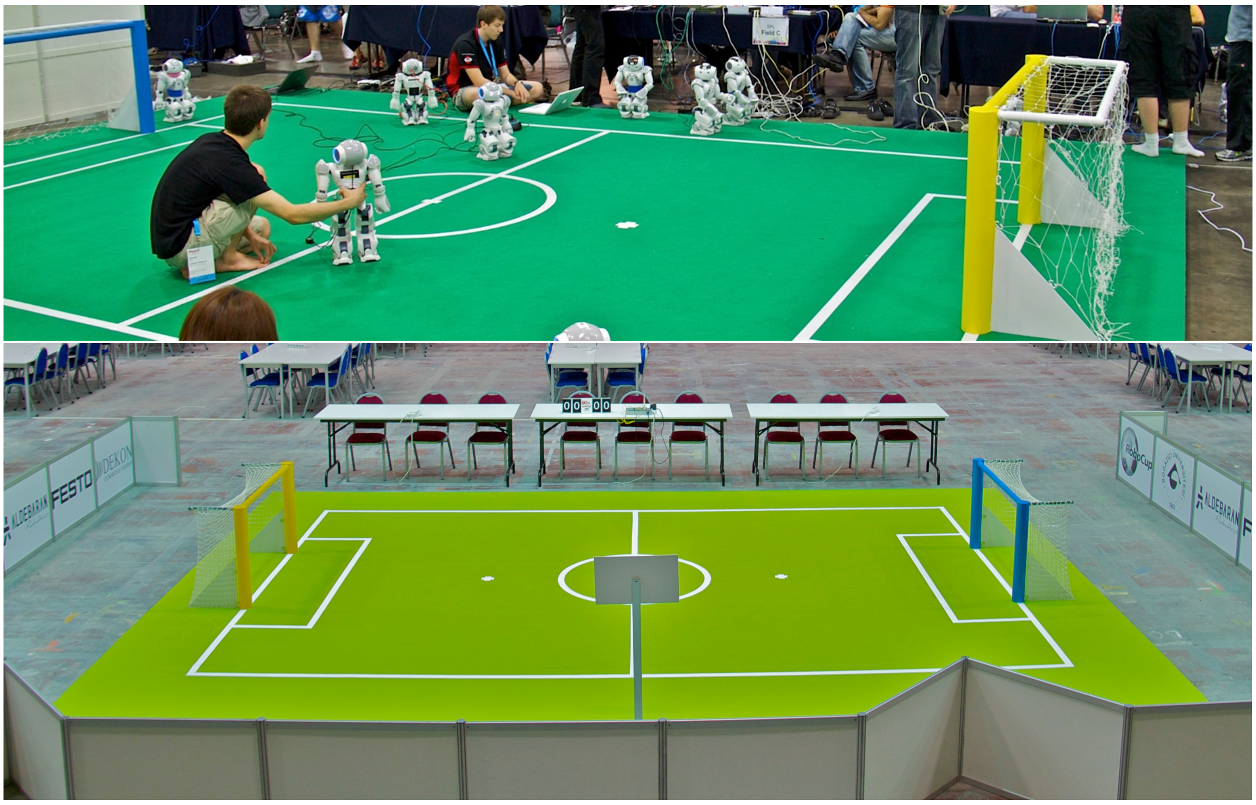


Figure 2.2: Top: Field colour in Singapore.
Bottom: Field colour in Istanbul.

Saliency Image

Image processing begins by creating a colour saliency image: a reduced scale image where each pixel colour has been classified using a predefined colour table. The image is down scaled by four to one, i.e. only one in sixteen image pixels is classified. During this stage, horizontal and vertical histograms representing the colour distribution across the image are also generated.

Region Building

The saliency image is then passed along to the region builder. Its function is to identify regions that are likely to contain detectable objects of interest. It works by vertically scanning over the saliency image whilst heuristically attempting to determine what each column of pixels may represent. Adjacent columns with similar properties are then grouped together to form rectangular regions that can later be used to seed the various object detectors.

Field Edge Detector

The field edge detector identifies the edge of the field, allowing downstream detectors to ignore the noisy region outside of the playing area. First, each saliency image column is traversed downwards until a field coloured pixel is encountered. These pixels are assumed to lie on the field edge, and lines representing these edges are then fitted using RANSAC [8].

Robot Detector

Relatively crude robot detection exists in the 2010 vision system. The region builder reports any otherwise unclassified non-green regions that contain at least one robot coloured pixel as a possible robot region. The robot detector performs several sanity checks on these regions before reporting a robot. These robot regions are used by downstream detectors to help rule out false positives.

Ball Detector

The ball detector is seeded using ball candidates reported by the region builder. Ball edges are found by scanning over the region both horizontally and vertically whilst searching for fluctuations in colour. Both classified colour changes and strong edges in the red image channel are considered. Circle fitting is achieved by selecting three of the detected edge points at random and constructing a circle. Several circles are constructed in the same manner using a different sampling of points. Finally all circles are sorted by their radius and the median result is reported as the detected ball.

Field Line Detection

Whilst present in the code base, the field line detector was not used at the 2010 competition. It attempted to match visible field line points against a world field model using an iterative hill climbing algorithm.

Goal Detector

Goals are detected by first analysing the colour histograms collected during the saliency image creation phase to locate large goal coloured regions. The detector then searches the original image for strong vertical edges in the vicinity and then fits a bounding box around the goal. The approach is rather novel in that the histograms will respond to sparsely classified goal regions, allowing the detector to locate goal posts even if they are poorly defined in the saliency image.

2.3 Design Overview

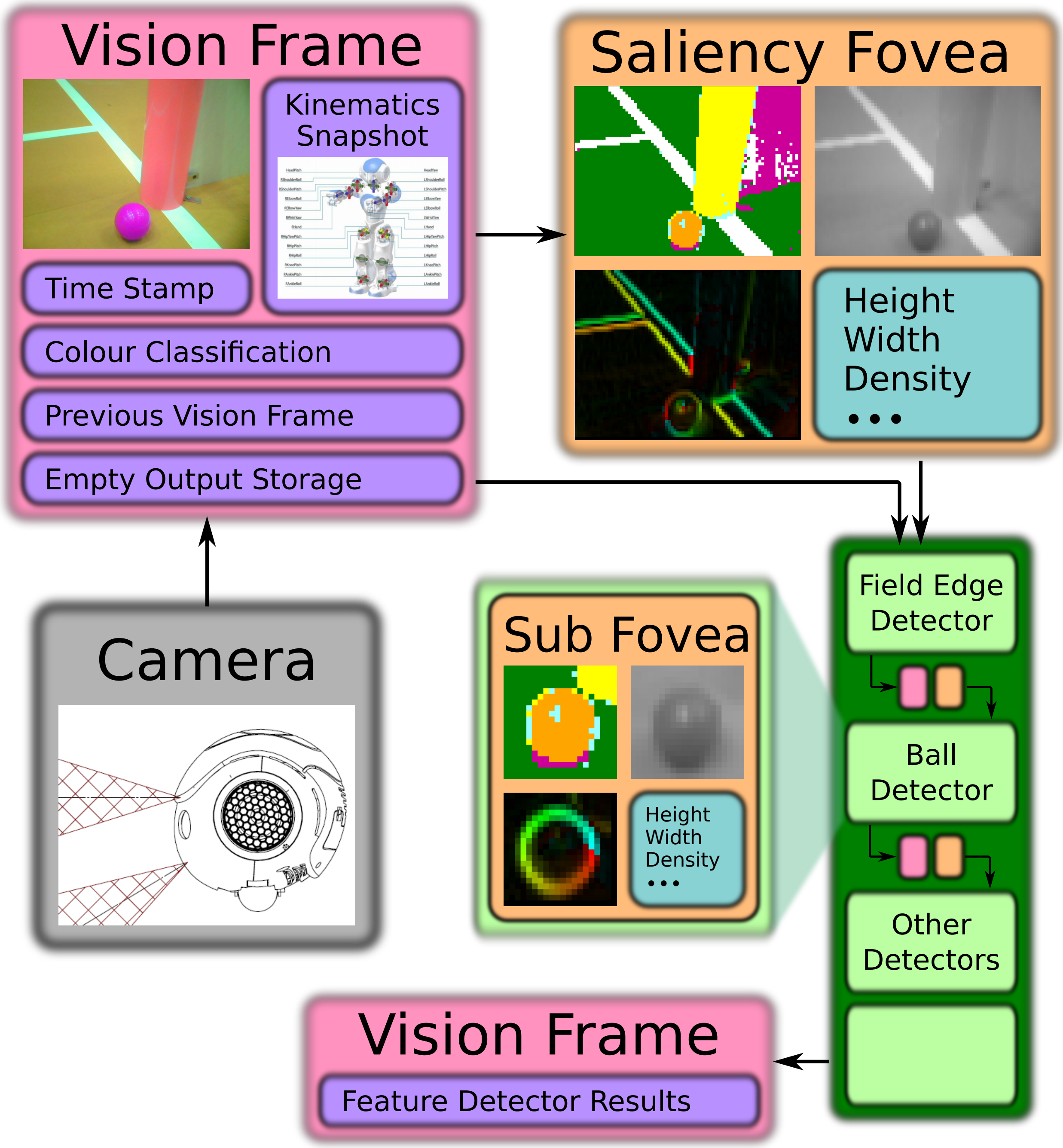


Figure 2.3: Flow of information through the 2011 system.

As previously alluded to in section 2.2, the 2010 vision pipeline has survived in its fundamental form, although information is better structured and defined in the new code. Beginning from a macroscopic perspective, consider the vision system to a simple black box that takes a set of inputs and produces a set of outputs. All inputs are grouped into a single referee frame call a *vision frame*. Conveniently, the vision frame also provides a buffer for vision output to be stored.

2.3.1 VisionFrame

The vision frame provides a well defined interface for invoking the black box vision system. The input fields represent a snapshot of the robots present state, the state that the vision system will process. After execution, detected objects are also stored in the frame, creating a mapping from robot state to perceived world state within a single context.

```
struct VisionFrame
{
   /* Constant input members */
   const struct
   {
      const uint8_t *image;
      const NNMC &nnmc;
      const CameraToRR &cameraToRR;
   };

   /* Input members that should be constant,
    * however cannot be due to limitations of our code */

   int64_t timestamp;
   WhichCamera whichCamera;

   /* Link to previous frame */
   boost::shared_ptr<VisionFrame> last;

   /* Output members */

   std::vector<BallInfo>          balls;
   std::vector<PostInfo>          posts;
   std::vector<FootInfo>          feet;
   std::vector<RobotInfo>         robots;
   std::vector<FieldEdgeInfo>     fieldEdges;
   std::vector<FieldFeatureInfo>  fieldFeatures;

   int *startScanCoords;
};
```

Input members, wrapped within the anonymous `const struct`, are as follows:

`const uint8_t *image`
: Pointer to the image that will be processed.

`const NNMC &nnmc`
: Colour classification table associated with the image. A separate table is used for both the top and bottom cameras, and therefore the active colour table must be declared along side the image.

`const CameraToRR &cameraToRR`
: The kinematic chain of the robot's current pose that allows points in the image space to be projected onto the field plane.

`int64_t timestamp`
: The time, in microseconds, that the image was captured.

`WhichCamera whichCamera`
: The current camera being used, either top or bottom.

`boost::shared_ptr<VisionFrame> last`
The previous frame processed by the vision system.

Chapter 4 will explain the purpose of the `last` member at greater length, however for now it is sufficient to understand that it links each frame into a list representing the robots chronological history. [1]

The output fields are self explanatory, except for the externally irrelevant `startScanCoords`. Visual inspection of the code should suggest that all output format is well defined and consistent in nature. Disastrously complex interfaces were one of the reasons the 2010 team rewrote their system from scratch, and it is hoped that keeping the new interfaces clean will improve the longevity of the code.

2.3.2 Image Format

Before delving into the vision module's internals, it is important to understand the exact format of the image. The native operation mode of the Nao's cameras produces images in the yuv422 format. The three y, u, and v channels define the images colour. The y channel represents the luminance across the image, and when viewed independently produces a grey scale image. Colour is defined by the u and v channels which represent the blue and red levels respectively. These channels tend to be less sharply defined and when viewed alone appear blurry. Because of this, the camera conserves bus bandwidth by sharing u and v values between pairs of adjacent pixels. This is reflected by the 422 in the formats name, in every 8 bytes of data, there are 4 bytes that represent the y channel and 2 that represent each of the colour channels.

2.3.3 Saliency Images

Processing of the vision frame begins with the creation of several saliency images, an idea taken from the 2010 design. These images are a form of preprocessing that produce an intermediate output applicable to several lower level object detectors. Although constructing the saliency images is a costly operation, this cost is subsidised over the multiple detectors. The 2010 code also had a second stage of image preprocessing, namely the region builder. This stage has been removed to recover its heavy computational demands, but in its place two new types of saliency images have been added making for a total of three: a colour classified image, a grey scale image, and an edge image. An example of each can be seen in Figure 2.3.

2.3.3.1 Colour Classified Image

The colour classified image is analogous to the 2010 saliency image. It simply maps the true colour values from the image to their respective classifications. The classification table used is the one associated with the input vision frame.

[1]Images are never copied from the internal kernel buffer; as such, the `image` pointer will only be valid if the kernel buffer is valid. There are 32 kernel buffers available, however we only protect one buffer during processing. Older frames are not guaranteed to reference their original images.

2.3.3.2 Grey Scale Image

Perhaps better named the "single channel image", the grey scale image is created by taking the three true colour values, y u and v, and flattening them into a single value. First, each channel is and multiplied by a scaler weighting and then summed into a single value. Customisable weightings allow this image to serve different purposes under different circumstances. For example, the ball detector is interested in the blue (u) channel of the image, whilst the field line detector operates on the luminance (y) channel. Finally, coarseness introduced into the image by the low resolution sampling grid is smoothed by applying a Gaussian blurring. The resulting image is not usually of direct interest to the object detectors; it is an intermediary image required for the construction of the edge image.

$$value = \left| \begin{pmatrix} y \\ u \\ v \end{pmatrix} \times \begin{pmatrix} weight_y & weight_u & weight_v \end{pmatrix} \right|$$

2.3.3.3 Edge Image

The edge image captures the intensity and direction of edges in the grey scale image. Intensity at a given pixel is determined by inspecting the value differences between it and the surrounding pixels. Let the images be two dimensional matrices indexed by x and y.

$$GreyImage = \begin{pmatrix} G_{(0,0)} & G_{(1,0)} & \cdots & G_{(x,0)} \\ G_{(0,1)} & G_{(1,1)} & & \vdots \\ \vdots & & \ddots & \vdots \\ G_{(0,y)} & \cdots & \cdots & G_{(x,y)} \end{pmatrix}$$

As the images have two dimensions, each edge will also have two components: $edge_x$ and $edge_y$. Together, they form a vector whose magnitude represents the edge's intensity and the argument its direction. A simple way of calculating the vector would be to simply compare the pixel $G_{(x,y)}$ with the two adjacent pixels $G_{(x+1,y)}$ and $G_{(x,y+1)}$.

$$\begin{aligned} edge_x &= G_{(x+1,y)} - G_{(x,y)} \\ edge_y &= G_{(x,y+1)} - G_{(x,y)} \end{aligned}$$

In the above calculation, the pixel $G_{(x,y)}$ has been included twice and therefore has been given an unfair weighting. A better solution is to include the pixel $G_{(x+1,y+1)}$ by applying the Robert's Cross. Unfortunately, the result calculated by the Robert's Cross represents the edge weightings along the 45 degree diagonal axes. To compensate, a rotation is applied to realign the values to the x and y axes.

$$\begin{aligned} a &= G_{(x+1,y+1)} - G_{(x,y)} \\ b &= G_{(x+1,y)} - G_{(x,y+1)} \\ edge_x &= \sqrt{2} * (a+b) \\ edge_y &= \sqrt{2} * (a-b) \end{aligned}$$

The above procedure can also be represented using matrix form.

$$\begin{pmatrix} edge_x \\ edge_y \end{pmatrix} = \begin{pmatrix} \sin\frac{\pi}{2} & \cos\frac{\pi}{2} \\ \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} \end{pmatrix} \times \left(\begin{pmatrix} G_{(x+1,y+1)} \\ G_{(x+1,y)} \end{pmatrix} - \begin{pmatrix} G_{(x,y)} \\ G_{(x,y+1)} \end{pmatrix} \right)$$

2.3.3.4 Sampling Density

Saliency images are sampled sparsely resulting into smaller images than the original. The 2010 system sampled every fourth pixel horizontally and vertically, whilst we sample only one pixel in eight. Lower level detectors can process these smaller images relatively quickly, and in many cases can extract all the necessary information from the saliency images alone. If greater precision is required, the original image can always be accessed.

2.3.3.5 Colour Histograms

Colour concentrations are profiled using two histograms constructed from the colour saliency image. These histograms tally the counts of classified pixels along both the x and y axes, and can later used to locate large coloured regions. The histograms were part of the 2010 vision system and a fuller description is available in the team report [14].

2.3.4 Foveae

Fovea are a generalisation of the saliency images. We have observed that in many situations that all information required to process a frame is captured by the saliency images alone. Unfortunately, there remain situations where it would be desirable to have a larger, higher resolution set of saliency images available. This would of course come at a greater cost in both terms of creating the images and processing them; however, most of the extra processing would be wasted as the finer details of interest are usually confined to a small region. If the approximate location of the desired details are known in advance, an obvious optimisation is to only sample the interesting regions. The foveated infrastructure provides a simple way for detectors to achieve this.

In our vision system, a fovea is a set of saliency images sampled at an arbitrary resolution representing a rectangular region of the image. As it happens, this definition also applies to the aforementioned saliency images when the entire image is the region of interest. Indeed the 2010 saliency image has been replaced by a full image fovea, but the true benefits of the foveated framework arise only when sub-sampling smaller rectangles within the image.

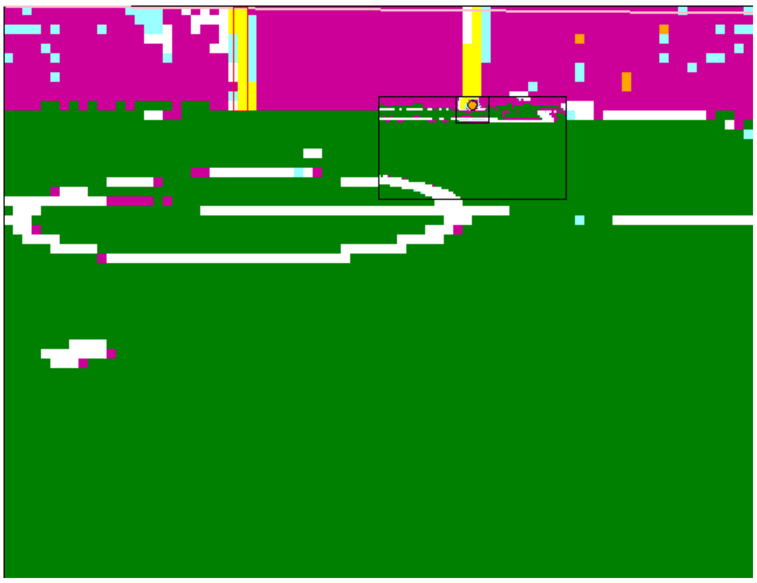


Figure 2.4: Zooming in using sub-foveae to locate the ball.

Fovea also function as a coupling between the heart of the vision system and the individual object detectors. Instead of processing an image, object detectors now operate on a fovea leading to two major advantages. The first is consistency, all information needed to invoke a detector is jointly contained by a vision frame and a fovea. Compare the function prototypes for the current ball and goal detectors with those of last year.

```
/* unsigned int *s is the seed for the random number generator */
/* 2011 */
void findBalls(VisionFrame &frame, const Fovea &fovea, unsigned int *seed);
void findGoals(VisionFrame &frame, const Fovea &fovea, unsigned int *seed);

/* 2010 */
void findBalls(ImageRegion **ballRegions, uint16_t numBallRegions,
  CameraToRR *convRR, Vision *vision, unsigned int *seed,
  RobotDetection *robotDetection, uint32_t *startOfScan,
  const std::pair<int, int> &horizon);

void findGoals(
  XHistogram xhistogram[IMAGE_COLS/SALIENCY_DENSITY][cNUM_COLOURS],
  YHistogram yhistogram[IMAGE_ROWS/SALIENCY_DENSITY][cNUM_COLOURS],
  CameraToRR *convRR, uint32_t *startScanCoords, Vision *vision,
  int *endOfScan, uint16_t numRobotRegions,
  RobotRegion **robotRegions, const::std::pair<int, int> &horizon);
```

The second benefit is that by definition, if a function is able to operate on a fovea, then transitively that function is also able to process any sub-region of the image at any resolution. Consider the ball detector, which is optimised to detect circles with a radius of 12 pixels. In image space, the balls radius can vary from 6 to 60 pixel depending on its distance from the camera. Rather than developing and optimising several detectors for balls of various sizes, a clean and elegant solution is to simply use the same ball detector, but have it operate using foveae of varying densities.

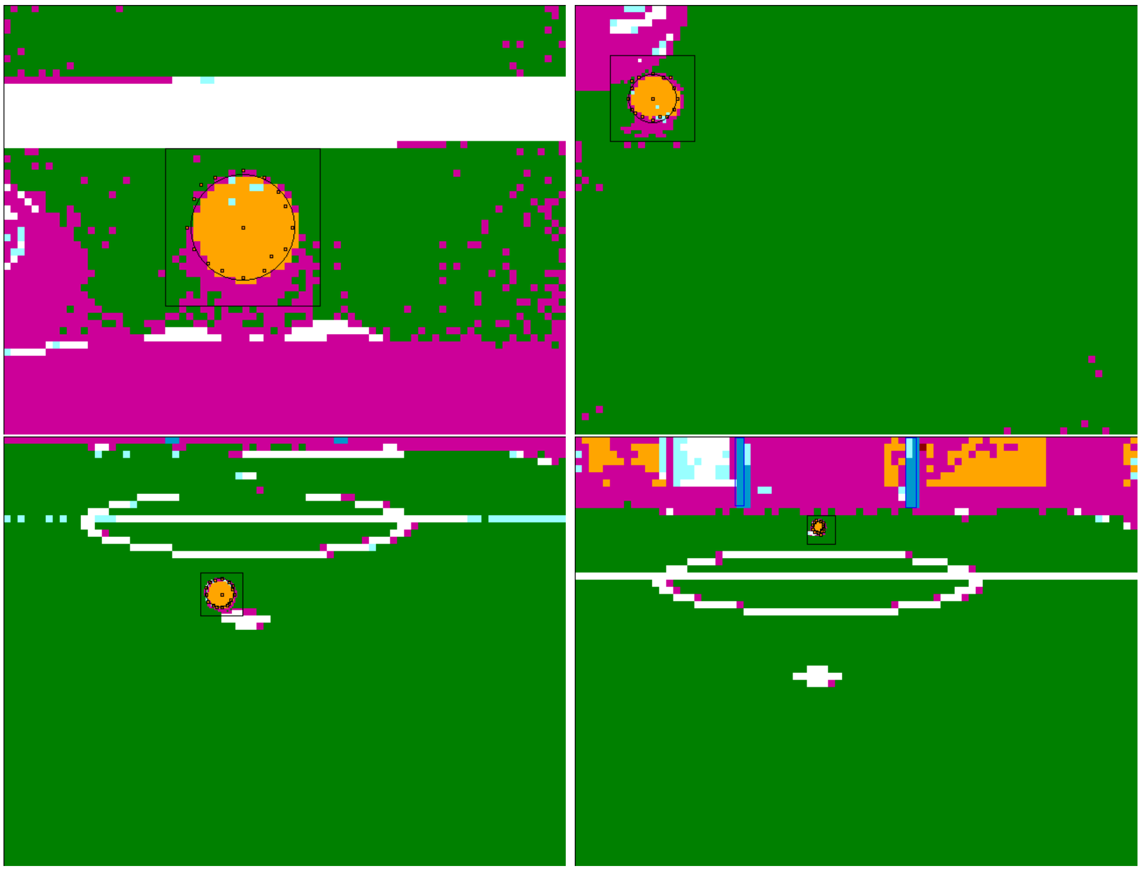


Figure 2.5: Using foveae of varying densities.

2.4 Off-line Vision Debugging and Testing

OffNao is the tool used by the rUNSWift team to connect to their robots and provide live feedback. Several other functions are also consolidated into this tool, including the ability to run the vision system off-line by replaying previously collected logs. Vision systems are notoriously difficult to test and debug; the prime culprit being reproducibility. OffNao allows any issue captured by a log to be reproduced and mended accordingly. Another inherent difficulty of vision systems is evaluating spacial data masked in a numeric form. A visualiser built into OffNao insightfully displays output data as a series of overlays over the image, enabling the vision team to interpret the systems output. These two features alone make OffNao an invaluable debugging tool, however many other features also exist including the vision specific colour classifier and camera calibrator. From a users perspective, these features remain unchanged from 2010, and a full description can be found in the 2010 report. [14]

2.4.1 Logging

Obtaining a complete log from all the robots sensory inputs is a deceptively difficult task. A single 640 by 480 pixel image is just over 600 kilobytes in size. Multiply that by the thirty frames per second processed by the vision system and suddenly the bandwidth required to stream a real-time video from the robot becomes 18.5 megabytes per second. Even if the Nao's networking hardware was able to sustain this bandwidth, it can not, the extra CPU time required by the kernel to transmit the data would cut into the vision thread's time slice.

The solution introduced this year is to create the log files locally on the robot. File system latency is minimised by writing the logs to RAM disk, a region of RAM masquerading as a high performance writeable disk. In this configuration the robot is able to capture 10 frames per second, however the log size is limited to the amount of available RAM, around 100 megabytes.

2.4.1.1 Log Formats

The 2010 version of OffNao supported two types of log files: OffNao records (.ofn) and yuv dumps (.yuv). OffNao records contain a log of all information collected during a live debugging session conducted over the network. They usually contain a low resolution saliency image along side a copy of the current blackboard, the central information store for the entire rUNSWift system. Although an OffNao record can be configured to stream raw images, network speeds makes this prohibitively slow. To obtain a high frame rate log, a yuv dump must be created locally on the robot. A yuv dump is simply a concatenation of raw image frames collected as the robot runs. They are useful for colour classification, however they lack vital information stored on the blackboard needed to actuate the vision system. Another weakness of both formats is that they are not synchronised with the vision thread, making it impossible to exactly reproduce the behaviour of the on-line system.

We have introduced an additional logging format, the BlackBoard dump. Like a yuv dump, it is a simple format created locally on the robot by concatenating a series of frames. Importantly, like the OffNao records, this format can be configured to include a snapshot of the blackboard. Additionally, the dumping processes is now synchronised with the vision thread. These two enhancements allow the entire vision processes to be faithfully reproduced off-line. [2]

2.4.2 Visualisation

The vision debugging visualiser has also been revamped to support the new features of the vision system. The single saliency image used in 2010 has been replaced with an option to view any of the colour classified, grey, or edge saliency image. Additionally, a field view has been added to help visualise where points projected from the image lie on the ground plane.

[2] Because the BlackBoard dump is a raw format, it lacks the meta data stored in an OffNao Record. This problem is remedied by opening a BlackBoard dump using OffNao, and then saving the file as an OffNao Record.

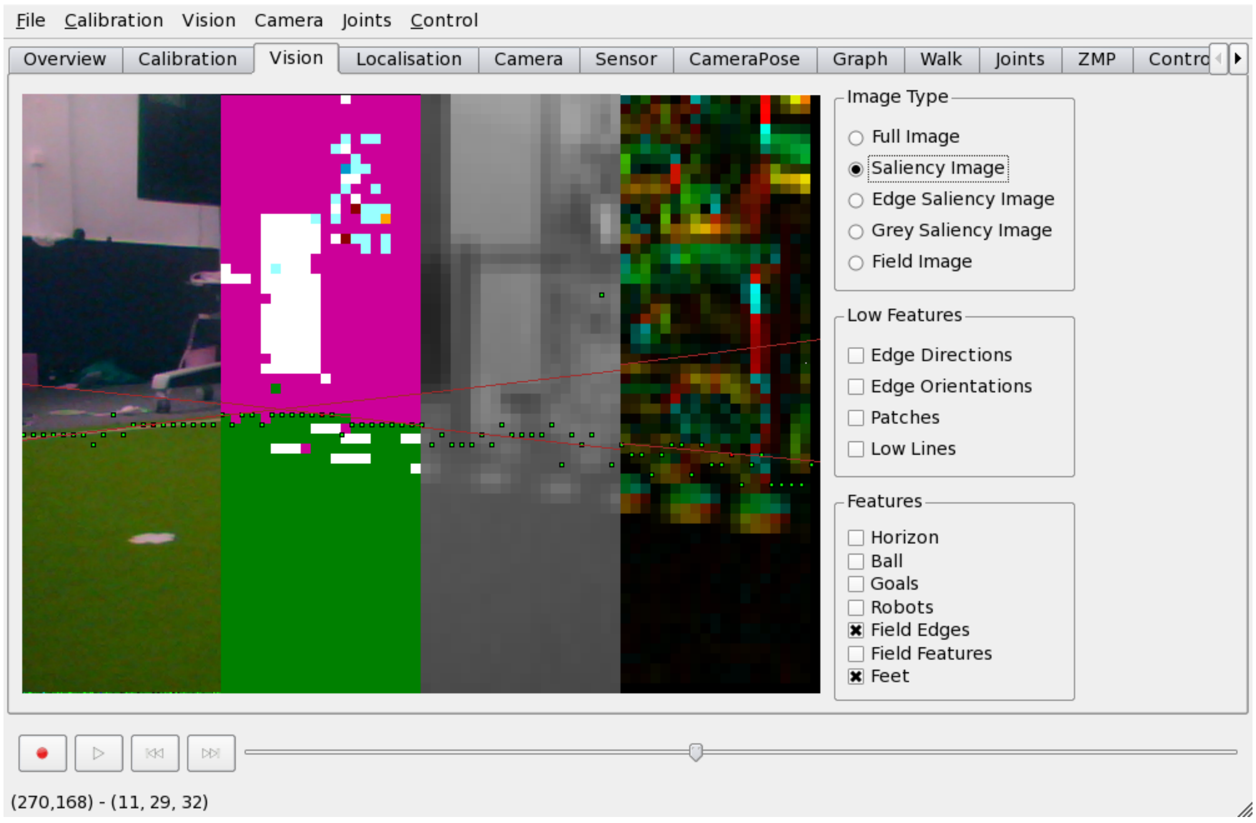

File Calibration Vision Camera Joints Control
Overview Calibration Vision Localisation Camera Sensor CameraPose Graph Walk Joints ZMP Contro

Image Type
- ○ Full Image
- ● Saliency Image
- ○ Edge Saliency Image
- ○ Grey Saliency Image
- ○ Field Image

Low Features
- ☐ Edge Directions
- ☐ Edge Orientations
- ☐ Patches
- ☐ Low Lines

Features
- ☐ Horizon
- ☐ Ball
- ☐ Goals
- ☐ Robots
- ☒ Field Edges
- ☐ Field Features
- ☒ Feet

(270,168) - (11, 29, 32)

Figure 2.6: A doctored image showing all available views. From left to right: True Colour, Colour Classified, Grey Image, Edge Image

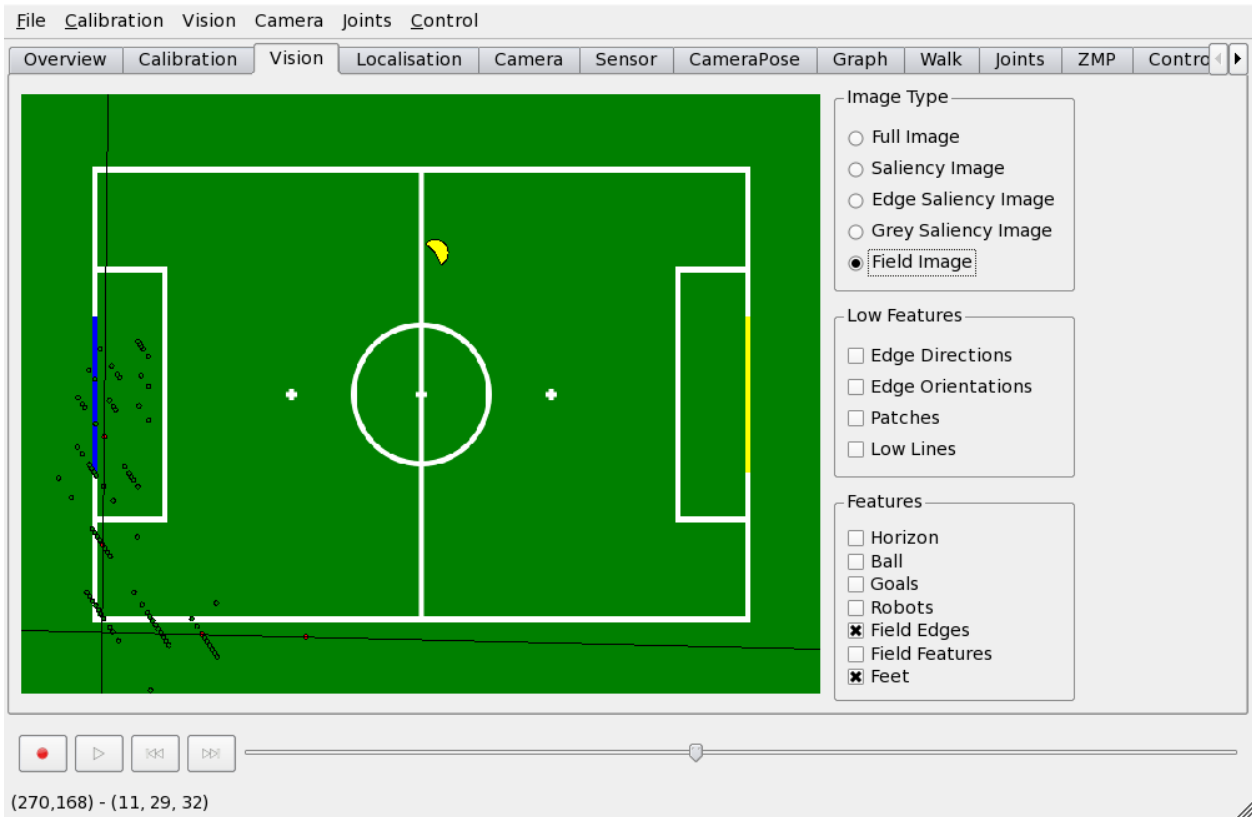

File Calibration Vision Camera Joints Control
Overview Calibration Vision Localisation Camera Sensor CameraPose Graph Walk Joints ZMP Contro

Image Type
- ○ Full Image
- ○ Saliency Image
- ○ Edge Saliency Image
- ○ Grey Saliency Image
- ● Field Image

Low Features
- ☐ Edge Directions
- ☐ Edge Orientations
- ☐ Patches
- ☐ Low Lines

Features
- ☐ Horizon
- ☐ Ball
- ☐ Goals
- ☐ Robots
- ☒ Field Edges
- ☐ Field Features
- ☒ Feet

(270,168) - (11, 29, 32)

Figure 2.7: The field edges as detected above, but projected into the ground plane and visualised using the Field View.

Chapter 3

Vision Test Suite

The vision test suite is an automated program for statistically testing the performance of the rUNSWift vision system. Vision testing was discussed earlier along side OffNao in section 2.4, which allows allows us to visually inspect the results of the vision system. It does not, however, provide statistical evaluation of system's performance. The vision test suite was intended to fill this void. Whilst work on the actual test suite itself was completed, the team's original ambitions to integrate vision statistics into our continuous integration environment were never realised. None the less, it can still be used to analyse the performance of the vision system and will be used later to evaluate the ball detector.

3.1 Image Classifier

Automated testing requires a set of inputs paired with a set of outputs assumed to be correct. The vision test suite accepts as input OffNao Records containing the expected output stored as special meta data. The meta data is added using the image classifier that has been added to OffNao. It allows a human to define visual objects within the image that can later be analysed by an external tool. Refer to the image classifier Manual located in the appendix for a full description of all functionality.

3.2 Vision Tester

Classified frames require a tool for analysis. Currently, the only such tool is the vision tester, a command line tool that opens a classified OffNao Record, analyses each frame, and then prints out a performance summary. In addition to the input record, a configuration file is required. This configuration file is of exactly the same format as the one used by the robot, meaning all configuration options available to the robot are also available to the vision tester. Most importantly, the configuration file allows for an appropriate colour classification to be loaded and used by the vision system.

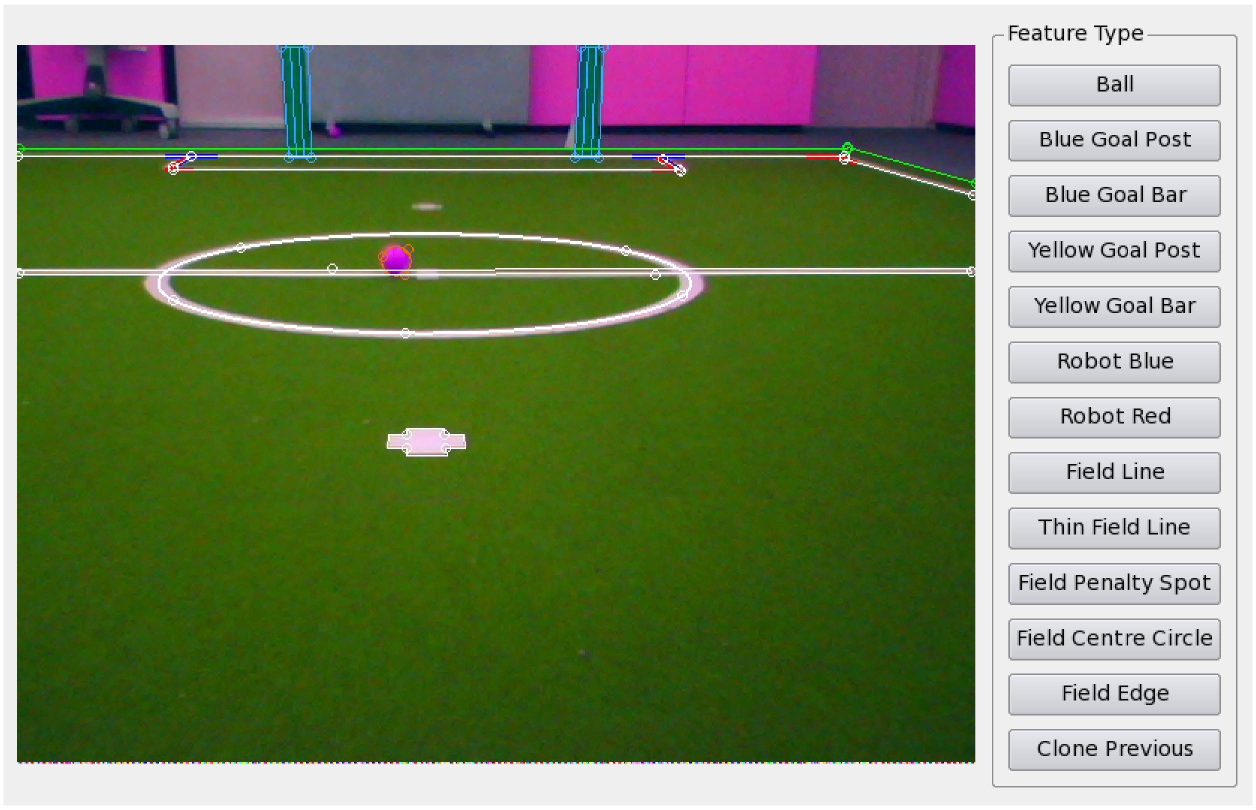


Figure 3.1: An image classified using the image classifier.

3.3 libvisiontest

All core functionality of the vision test suite is built into the shared library libvisiontest. Both the image classifier and the Vision Tester require access to common code responsible for manipulating and comparing shapes, so as a design decision it makes sense to bundle the shared functionality into a library. It is also hoped that future rUNSWift teams may find the vision test architecture useful and possibly introduced additional features, such as automatically locating and reporting frames within OffNao where the vision system has performed poorly. The functionally contained by libvisiontest should theoretically make the development of such features trivial.

```
Total (0.80) {
        Balls (1.00) {
                true positives : 1/1
        }
        Blue Posts (1.00) {
                true positives : 2/2
        }
        Field Edges (0.50) {
                true positives : 1/2
        }
}
```

Figure 3.2: Sample output snippet from the vision tester.

Chapter 4

Ball Detection

The most fundamental aspect of playing soccer is to simply get to the ball, and part of that challenge is simply being able to see it. The ball detector was rewritten from scratch this year, partially as a result of the region builder being gutting out, and partially to cope with the lower resolution saliency scan. The approach taken is fundamentally very simple: cheaply find a list of candidate regions likely to contain a ball, then search those regions using a more intensive algorithm. The new detector will be analysed using the vision tester for both the percentage of balls detected and the accuracy of the reported measurements. For comparison, results will also be compared against those of last years detector.

4.1 Locating Candidates

The SPL RoboCup competition uses the distinctly coloured orange ball. This provides a very simple colour based method for classifying pixels likely to belong to the ball. Grouping these pixels into clusters is slightly more complicated.

At a distance of six metres, a full field length, the ball can appear as a single orange pixel in the saliency image. On the other extreme, a ball appearing at the robots feet may be fifteen by fifteen saliency pixels large, occupying approximately 5 percent of the entire image. The grouping algorithm must therefore be suitable to balls of all sizes. One common approach is to build disjoint sets of adjacent pixels [7]. This approach, however, does not handle cases where pixels belong to the same ball but are somehow separated, either due to blur, specular lighting, poor colour calibration, or some other phenomena. A much more simple approach is to locate the top, left, bottom, and right most orange pixels in the image and assume the bound the ball. This approach was used in early prototyping, but is vulnerable to even a single misclassified orange pixel. We instead use an approach inspired by the success of the goal detector, a remarkably simple solution that scales to goal posts of all sizes.

In a similar fashion to the goal detector, the ball detector builds histograms of orange pixels along both the x and y axis of the image. To eliminate noise present in the irregularity coloured background, only the region of the image bounded by the field edge is considered. Peaks in the histograms are then found and used to estimate the bounds of potential ball candidates. There is one major caveat worth noting: by compressing two dimensional spacial information into one dimensional histograms we have discarded information. When reconstructing the two dimensional regions, artefact regions containing no ball pixels may appear, as illustrated by figure 4.2. An extra sanity check is therefore required to test for the presence of orange before reporting the region as

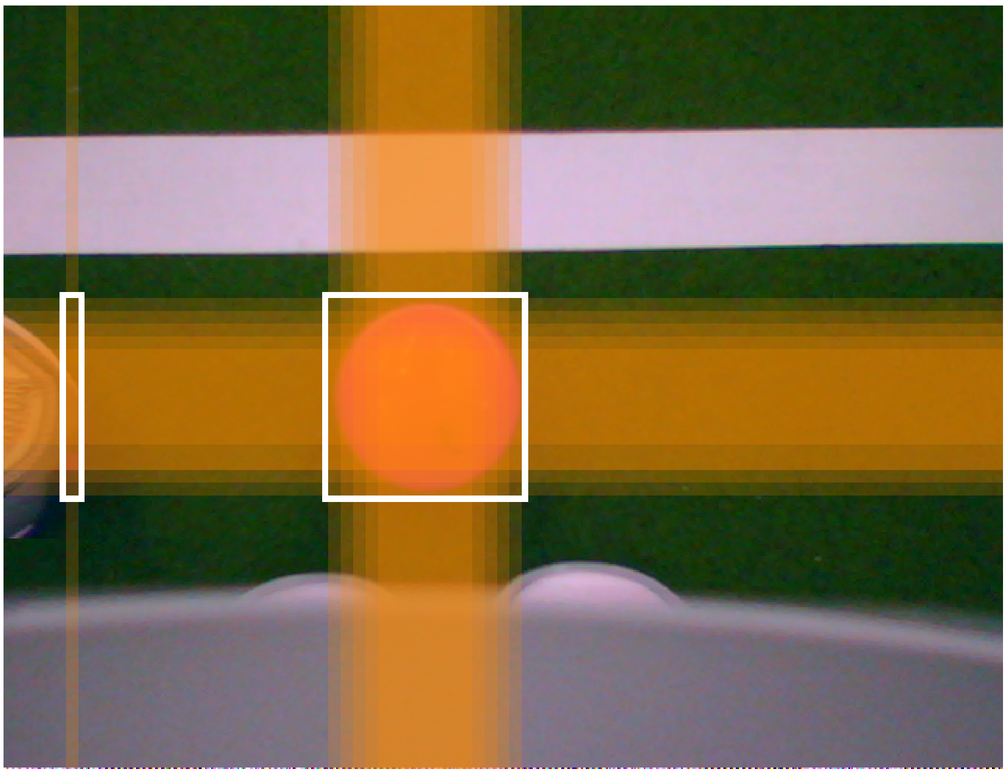


Figure 4.1: Histograms used to locate candidate regions, outlined in white. Note the falsely classified pixel in the shoe that has created a strangely shaped candidate region.

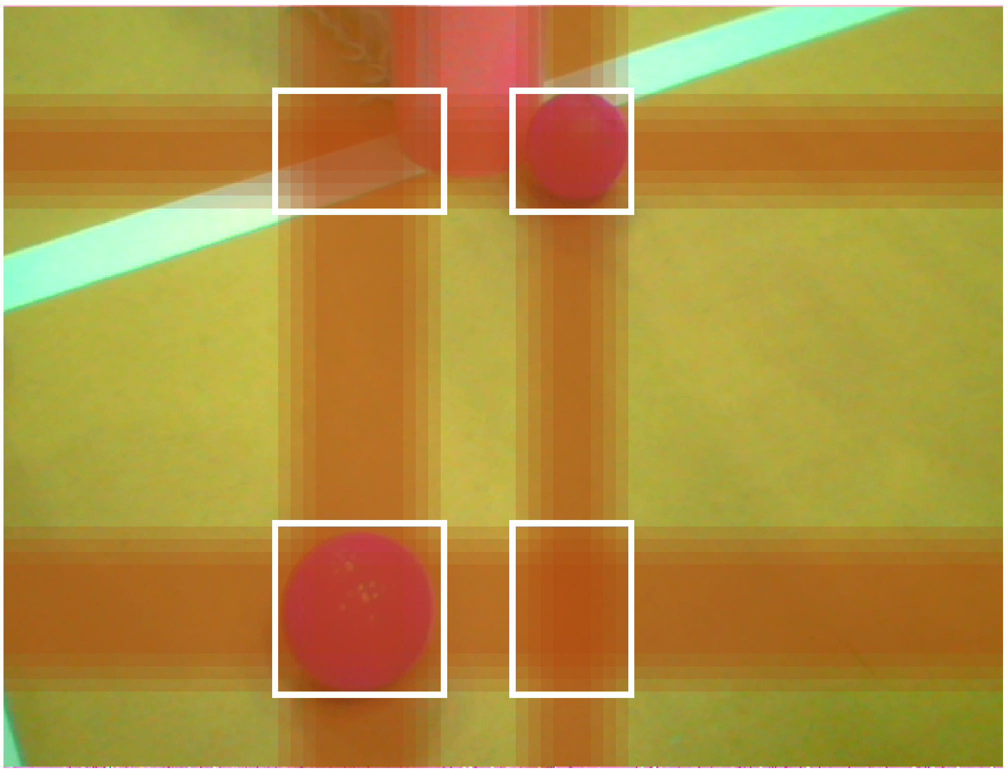


Figure 4.2: The introduction of a second ball causes two artefact regions to appear.

a possible ball candidate.

4.2 Foveating In

For each ball candidate, the ball detector next constructs a fovea covering the related region of the image. The candidate fovea must satisfy two requirements to be of use:

1. The ball radius in the resulting fovea should be normalised between 6 and 12 pixels.
2. The edge intensities present in the edge image must be highest around the boundary of the ball.

4.2.1 Choosing a Sampling Density

Normalising the ball radius has simplified the design of the ball detector by allowing us to only focus on detecting balls with a specific range of radii. An appropriate ball radius can be obtained by selecting an appropriate sampling density for the fovea. If the would be ball radius is less than 6 pixels, it can be doubled by doubling the sampling resolution. Likewise, if the ball radius is greater 12 pixels sampling at half the resolution will then half the resulting ball radius.

4.2.2 Choosing Appropriate Edge Weightings

The ball detector also relies on strong edges being reported around the ball's edge. Unfortunately, the luminance channel used to generate the saliency fovea is highly responsive to specular lighting reflected from the ball's surface. This can lead to false edges being detected in reflections on the ball. As an alternative to using the luminance, it may seem logical that the edges of an orange ball would be well defined in the red channel. Whilst edges in the red channel found to work very well in most cases, edges were lost when the ball was set against a yellow goal post. Experimentation with the blue channel on the other hand exceeded all expectations. Perhaps at first a little surprising, we found the strongest variations between the ball colour and all other colours in the environment to be present in the blue channel. Therefore, the foveae created by the 2011 ball detector use an edge weighting that highlights the edges in this channel.

4.3 Revising the Candidate

A good estimate of the ball's centre is required for the circle fitting stage of the algorithm. Therefore, if a candidate fovea is of a higher resolution than the saliency fovea, then the bounds of the candidate region ought to be refined. Fortunately, the exact same code that was used to locate the original candidate can be reused for this purpose, except in this case the code will use the candidate fovea in place of the original saliency fovea.

4.4 Locating Edge Points

By this stage the Ball Detector should have a fairly accurate estimate of the balls location and dimensions, though so far we have only used statistical hints provided by the histograms. Points

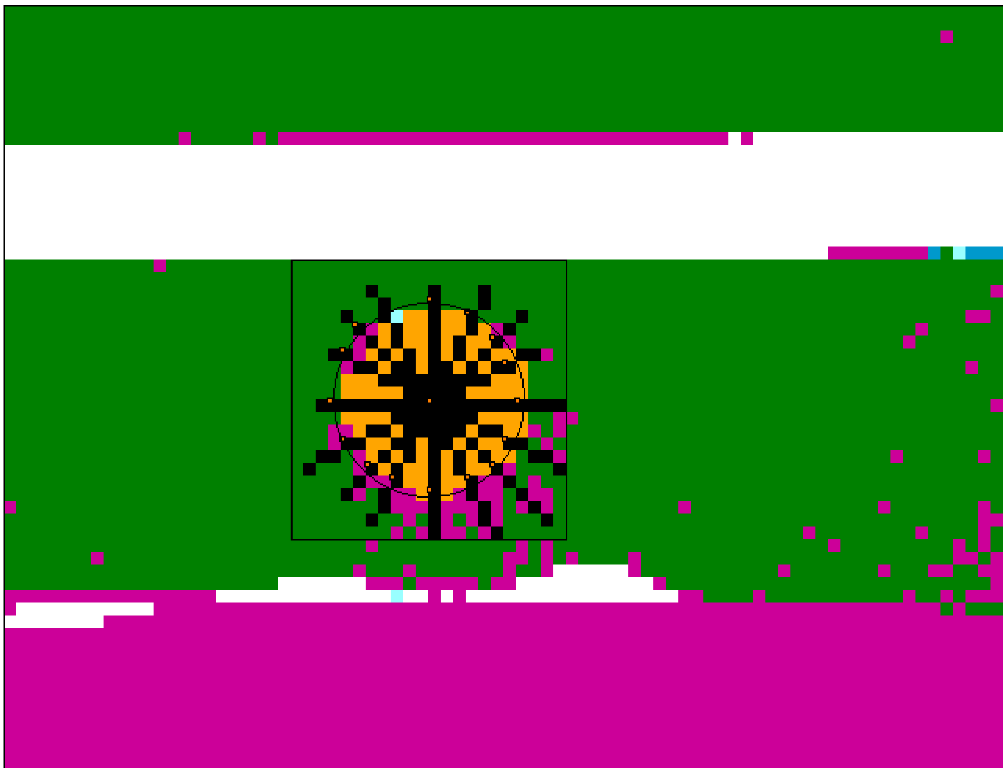


Figure 4.3: Scanning radially outwards from the balls centre.

lying along the ball's boundary are the first concrete features the ball detector searches for. The process is relatively straight forward: start from the centre of the candidate fovea and scan radially outwards until an edge is encountered. Defining an "edge" is a little bit more complicated and relies on a somewhat fiddly heuristic.

4.4.1 Radially Scanning

In total, sixteen radial scans are performed from the centre of the fovea at angles that evenly divide the circle. Traversing a two dimensional image in an arbitrary direction is somewhat non-trivial, but we provide a utility called a BresenhamPtr that hides most of the involved complexity. As the name implies, Bresenham's famous integer line plotting algorithm is used internally [5]. Once a BresenhamPtr has been correctly initialised, the traversal can be performed using the familiar c++ iterator idiom.

4.4.2 Defining Edges

The simplest definition of the ball's boundary would be the point with the strongest edge intensity encountered during a radial scan. Unfortunately, this does distinguish between ball edges and other types of edges. To prevent stray edges lying outside of the actual ball from being detected, we prematurely terminate a scan if we encounter green or white pixels. Edges are also discarded if an insufficient number of ball coloured pixels were encountered. The actual algorithm used can be best summarised in the following pseudo-code.

```
$ball\_count \leftarrow 0$
```

```
$green\_white\_count \leftarrow 0$
$total\_count \leftarrow 0$
$current\_point \leftarrow centre$
while $current\_point$ is inside fovea and $green\_white\_count < 2$ do
  if $current\_point$ is green or white then
    $green\_white\_count \leftarrow green\_white\_count + 1$
  end if
  if $current\_point$ is ball coloured then
    $ball\_count \leftarrow ball\_count + 1$
  end if
  $total\_count \leftarrow total\_count + 1$
  if $current\_point$ is strongest edge so far then
    $strongest\_edge \leftarrow current\_point$
  end if
  $current\_point \leftarrow next\_point$
end while
if $ball\_count/total\_count > 70\%$ then
  $edge \leftarrow strongest\_edge$
else
  discard edge
end if
```

4.5 Fitting a Circle

Circles are fit to the detected edge points using RANSAC [8]. First, three edge points are chosen at random and then used to construct a circle. If several of the remaining edge points also lie near to the circle, they are said to form a consensus. The strength of the consensus is defined by two factors: the number of points in the consensus, and the summed squared distance from each point to the circle. The strongest consensus is the one with the most points close to the circle. RANSAC attempts to find multiple consensus sets by varying the initial three points and assumes the best fitting circle is the one with the strongest consensus set.

The algorithm we use makes one simple improvement over a vanilla RANSAC implementation. Because the radius of the circle can be roughly deduced from the candidate region's dimensions, we only search for circle sizes that fit the candidate. If three points construct an unrealistically large or small circle, it is immediately discarded.

The algorithm is resilient to outliers, which makes it very good at detecting occluded balls. If an occlusion has deformed the ball's boundary, the points lying along the deformation will not fall into the stronger consensus sets. Similarity, if a circle is constructed from one or more invalid points, it will likely be void of other points needed to form a consensus. The result is that the ball detector will only detect balls if a circular segment of the boundary is visible. This prevents circles from being found in random point clouds that could potentially lead to false positives.

4.6 Ball Tracking

The major innovation introduced into the 2011 ball detector is the use of historical information to track the ball when no candidate regions are found. There are many possible situations that could cause a ball to be detected in one frame and not the next, but the most tragic is a ball not being detected simply because the saliency resolution was too low. This years saliency resolution of one in eight gives a distant ball ample wriggle room to hide in between the saliency grid. Therefore, it stands to reason that if a small ball suddenly disappears, there is a possibility that it hasn't actually moved very far. If such a disappearance occurs, the new ball detector will scan the ball's last known location at a higher resolution.

The implementation of this feature is truly elegant. First, a higher resolution fovea is constructed from the region of the image in which the ball was last detected. Constructing a new saliency fovea, especially at high resolution, is expensive, but a major part of this expense is generating the edge image. The edge image is not required by the ball detector, and fortunately its creation can be disabled by specifying a zero edge weighting during the fovea's construction. The rest of the process is simple. Because the vision system is designed to work on foveae, not images, the ball detector can simply be called recursively using the new fovea and the rest of the algorithm will run exactly as it would have otherwise, except in higher resolution.

Chapter 5

Results

5.1 Ball Detection Results

5.1.1 Comparison of Detectors

We compare the performance of three detectors.

1. 2010 Ball Detector
2. 2011 Ball Detector without ball tracking
3. 2011 Ball Detector with ball tracking

As a disclaimer, these results are likely biased towards the 2011 detector variants, as the intricate knowledge of the 2010 detector was lost with the departure of its original author, Adrian Ratter. Although best efforts have been made to optimise its performance for testing, there likely remain tweaks that would improve its results.

5.1.2 Test Sets

Each detector will be tested using three different test sequences, each designed to stress a different aspect of the ball detector.

Name	**Description**	**Game Scenario**
Close Balls	An assorted set of close range balls.	General game play.
Far Balls	A chronological sequence of a ball moving in the range of 3-6m.	Distant ball location. Goalie ball tracking.
Sprint	A chronological sequence of the robot moving towards the ball at high speed. Set is extremely blurred.	General game play.

Thumbnails of the test sequences have been provided in the appendix "Ball Test Sets".

5.1.3 Detection Ratio

The simplest performance metric is simply the number of balls detected in the test sets. The ratios below have been calculated by dividing the number of percepts reported by the detector by the actual number of human classified balls. The percepts are only considered valid if they overlap one of the human classify balls, otherwise the system reports a false positive. However, there were no false positives reported in any of the test sets.

Table 5.1: Ball Detection Results

	Close Close	**Far Ball**	**Sprint**
2010	17/22 (0.77)	15/34 (0.44)	42/45 (0.93)
2011 Tracking	22/22 (1.00)	34/34 (1.00)	39/45 (0.87)
2011 No Tracking	n/a	32/34 (0.94)	n/a

Although the 2011 ball detector reports a perfect score on both the Close and Far Ball sets, this is a slightly biased result. These frames were captured in a highly controlled fashion for the purpose of testing, and any issue found within these frames was promptly fixed during our development cycle. A realistic game dump is unfortunately extremely difficult to obtain, as dumping at even ten frames per slows the other modules running in parallel to a crawl. None the less, it is fair to conclude that the ball detector is highly robust to the many situations represented by our tests. In fact, in searching through all recent dump files created both in the lab and at competition, the only undetected balls were in highly blurry frames.

In the Close Ball Set, the 2010 detector had difficulties detecting balls placed in front of a yellow goal post, likely due to the lack of an edge in the red image channel. It also falsely detected a field edge that in turn caused a ball, now appearing to lie outside of the field boundary, to be ignored. Various improvements made to the 2011 field edge detector addressed these falsely detected field edges, thus the new detector does not suffer the same problem.

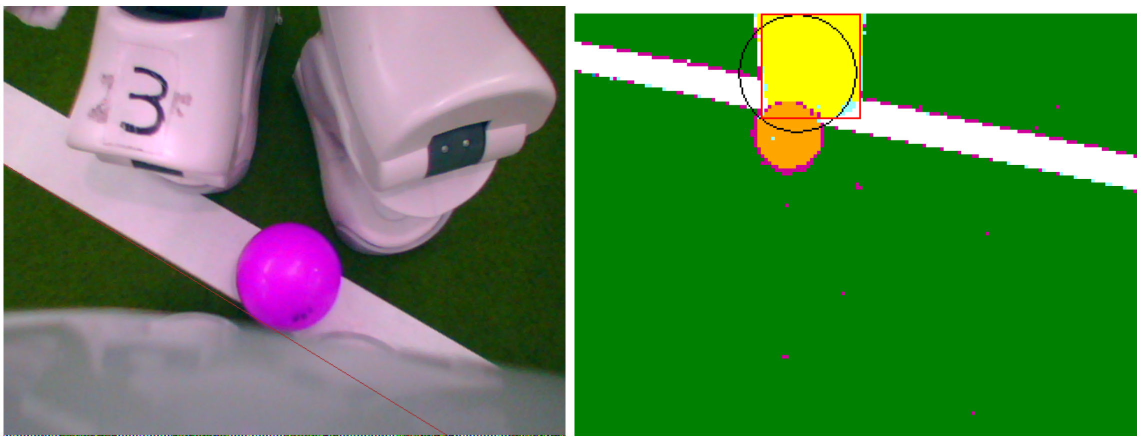


Figure 5.1: Left: A falsely detected field edge resulted in the ball not being detected. Right: The lack of a strong edge between the yellow and the orange regions causes an oversized ball to be detected.

The 2010 Detector also had difficulty detecting long range balls, despite operating at twice the resolution of the new detector. This is not a huge problem as long as the ball is at least occasionally detected, as once the direction to the ball is known the robot can move towards it. The closer the robot gets, the more reliably it can detect the ball. However, the development of an advanced ball [15] model has increased the importance of reliable long distant ball tracking, especially for the goalie. The 2011 detector was developed with this in mind, and is thereby able to reliably locate and track distant balls. On the subject of distant balls, disabling the tracking fovea has only resulted in two balls being missed. Despite operating at half the resolution, the 2011 detector is still substantially better at locating distant balls even without any form of tracking.

The Sprint set is perhaps the most interesting of all, as the 2010 detector out performed the 2011 detector. The new method of circle fitting using RANSAC strictly requires a circular arrangement of points. RANSAC is effective at ruling out oddly shaped regions of orange as false positives, however blurred balls tend to be just that. Interestingly, the old detector seems able to robustly detect these oddly shaped balls despite substantial blur in the image.

5.1.4 Positional Accuracy

Positional accuracy measures how closely the centre of a percept matches the centre of a human classified ball. The position of the ball is required to project it from image space into the ground plane, and is therefore extremely important. As the ball decreases in size, high positional accuracy becomes even more important due to the inverse relationship between a ball's size and its distance from the camera. The positional offset is measured in terms of the ball's radius. For example, a score of 0.2 implies that the reported centre was offset by 0.2 radii from the human classified centre.

The scores have been aggregated into buckets that are 0.05 units in width and are then plotted as a frequency histogram. The accuracy and consistency of the measurements is represented by a bell curve fitted over the data and is listed in the form $k * norm(mean\ offset, standard\ deviation)$. Ideally, the mean offset should be zero and the standard deviation as low as possible. Note that the distinction between the tracking and non-tracking versions of the 2011 detector has not been made in the below plots as they yield equivalent results.

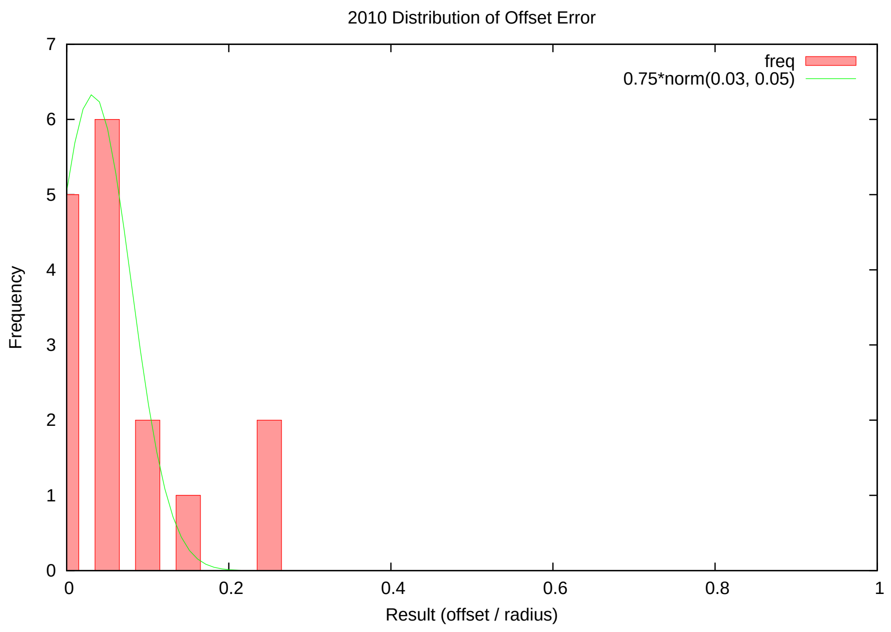
Both the 2010 and 2011 detectors produced impressive results, with offsets averaging no more than 0.2 times the length of the radius. In terms of consistency, the 2011 detector produced less variant results, but the 2010 detector produced fewer extreme outliers.

A particularity surprising outlier was present in the results of the Far Ball test set. The 2011 detector detected 6 balls within 0.025 radii of their human classified position, whilst all other offsets were centred around 0.2 radii. One possible explanation is that these balls were processed by a higher resolution fovea than the others, and thus the properties of the detector. changed. A blue curve that ignores this outlier has also been fitted to better represent the 2011 detectors performance.

Another result worth noting is the accuracy of the 2010 detector over the sprint data set. Although the scores are relatively variant, all measurements are within 0.5 radii of the balls actual centre. This is very impressive given the blurriness of the test set. On the other hand, the 2011 detector on average reported better and more consistent percepts, however several extreme outliers were also reported. These results reiterate the robustness of the 2010 detector to blurry images, which not only detects blurry balls but is also able to accurately define them.

Finally, somewhat expectedly, we see that over close ball set both detectors performed more or less equally.

Figure 5.2: Close Ball Test Set: Offset Error Distribution.



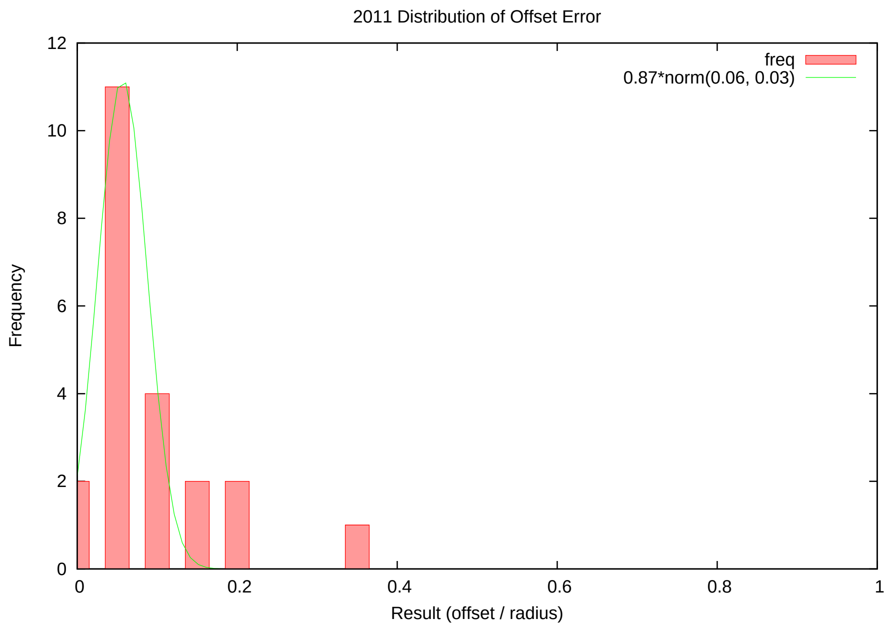
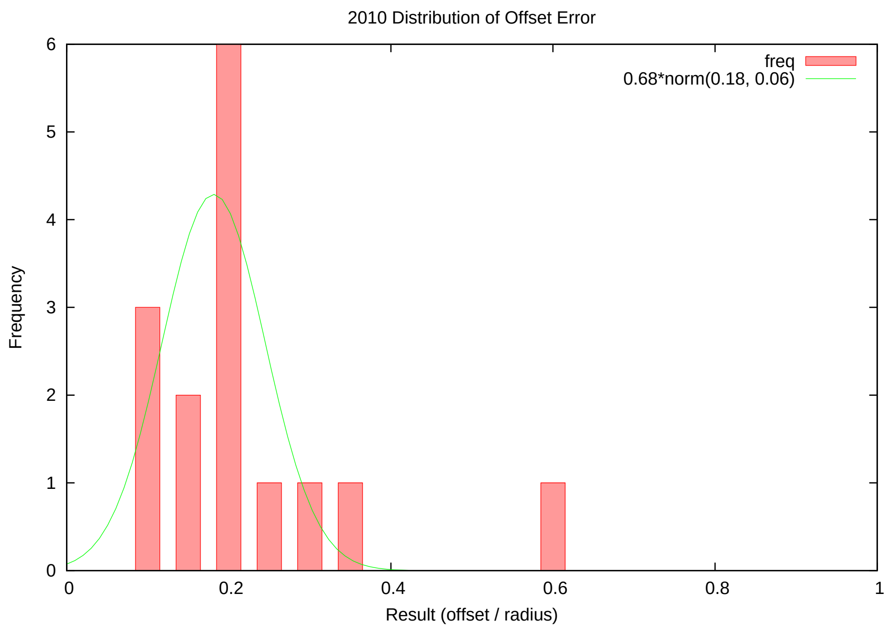


Figure 5.3: Far Ball Test Set: Offset Error Distribution.



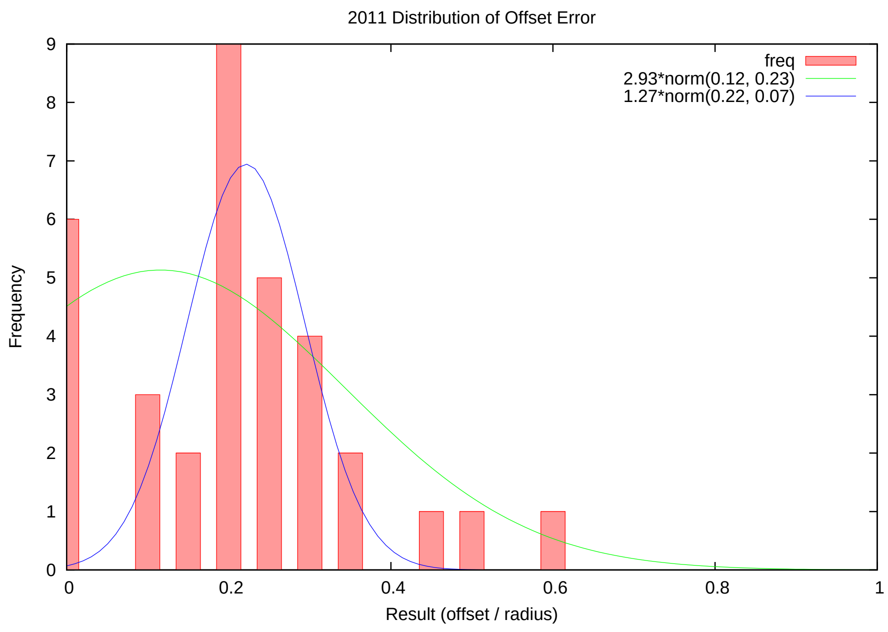
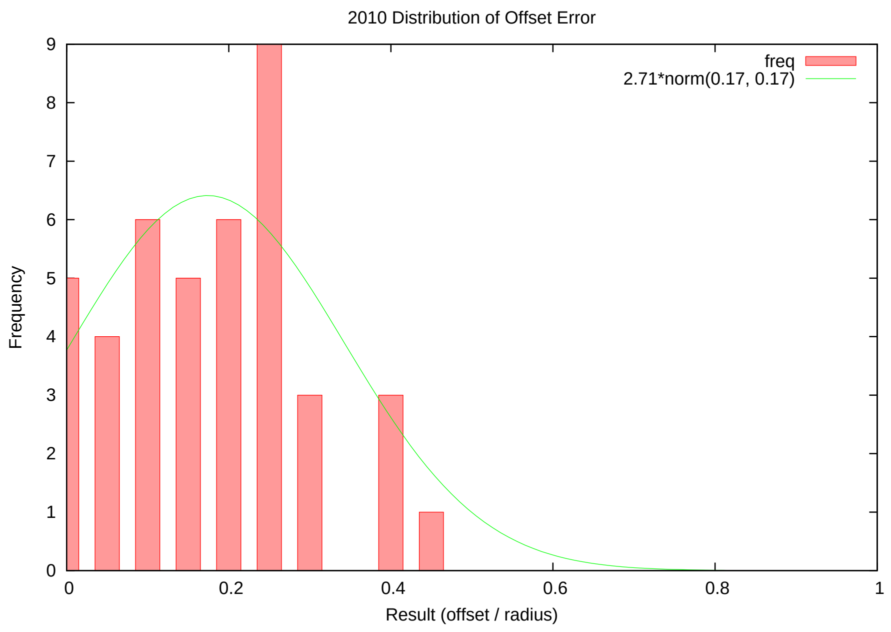
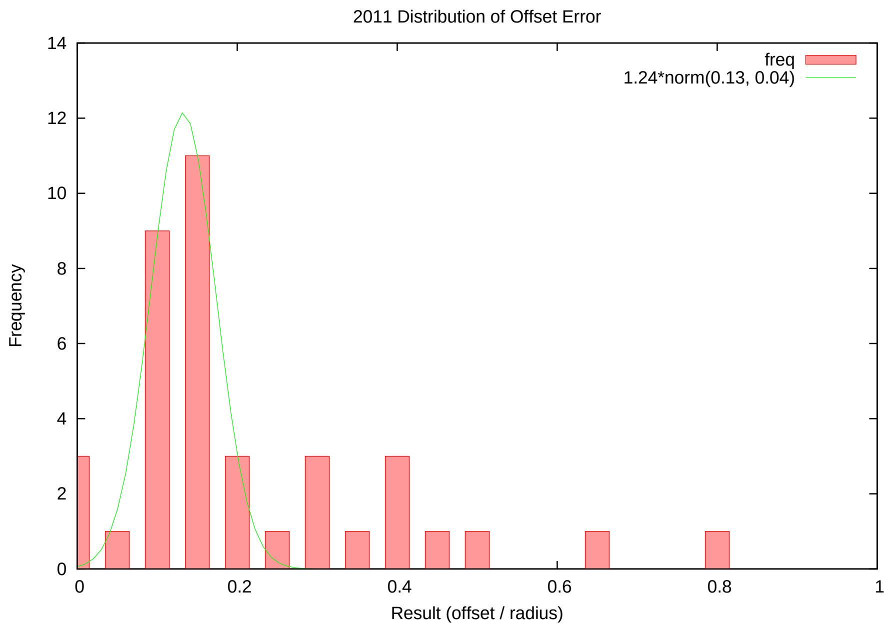


Figure 5.4: Sprint Test Set: Offset Error Distribution.





5.1.5 Radial Accuracy

The ball's radius is also a feature that can used to calculate its distance from the camera. Although rUNSWift relies solely on positional information to make this calculation, it is likely that the calculation could be refined by taking the ball's radius into account. Therefore, our tests will also measure the error in the radius as a ratio of reported length and actual length. Again, the below bell curves are listed in the form $k * norm(mean\ deformation, standard\ deviation)$. Ideally, the mean deformation should be 1, and the standard deviation should be low.

The radial accuracy graphs tell much the same story that the was told by the positional measurements. The new detector yielded more consistent results, whilst last year's detector yielded fewer extreme outliers. In maintaining this sense of deja vu, both detectors performed almost identically on the close ball set.

The strange outlier that was present in the far ball set has resurfaced, except that this time it almost appears as though the data could be best represented using the sum of two distinct bell curves. Perhaps this supports the theory that the properties of the new detector vary with the resolution of the fovea.

Finally, the impressive resilience to blur displayed by the 2010 detector has again resulted in impressive accuracy. Compared to the 2011 detector, results were equally variant, but lacked any extreme outliers.

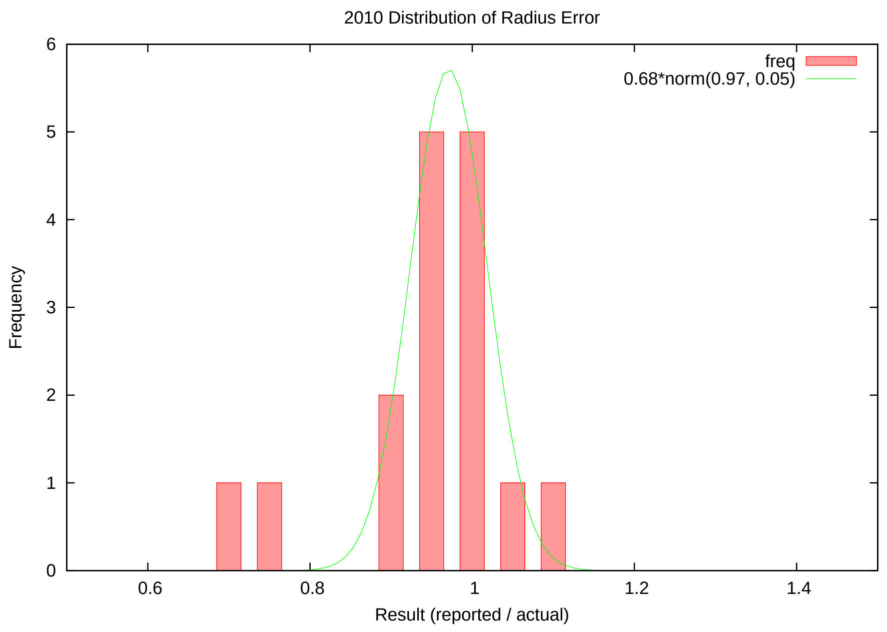
5.2 Team Results

With rUNSWift only placing among the top eight finalists, the teams aspirations were left somewhat unfulfilled. None the less, several key gains were made during the year preceding the 2011 competition. The first is that all visible features key to playing the game of SPL soccer are now represented by a detector present in the vision pipeline. Secondly, the underlying vision framework has been substantially reworked to allow individual detectors to be easily substituted and experimented with. Finally, perhaps most importantly, a method now exists for evaluating any modifications made to the system, whether they be incremental improvements or a completely substituted component. The combination of these improvements exposes many low lying fruits that can hopefully be exploited by future rUNSWift teams.

Two voids that had always been present until 2011 were the lack of a field line detector and a reliable robot detector. Both of these have been added to the vision pipeline, however at a substantial cost. The field line detector alone consumes roughly half of all processing time available to the rUNSWift system, and with the 2010 vision system already struggling to maintain the maximum frame rate, making available the resources for two new detectors was no trivial task. None the less, the required speed up has been achieved and there no longer exist any fundamental gaps in the vision pipeline.

In contrast to the 2010 team where Adrian Ratter by in large authored the entire vision system as a solo effort, the 2011 team had three members dedicated to vision alone. Whilst this extra man power undoubtedly helped rUNSWift compete with the larger teams, the state of the 2010 code was not one that fostered collaborative development. The scope of the tangled web of dependencies between the vision components was only truly revealed after a key component, the region builder, needed to be gutted. Newly defined standards and interfaces have resulted in a much cleaner system that allows for individual components to be modified in isolation without needing to redefine the way information flows throughout the entire vision system. It is a fundamental requirement for any team that its members can work without trampling on each others toes, and the requirements of

Figure 5.5: Close Ball Test Set: Radius Error Distribution.



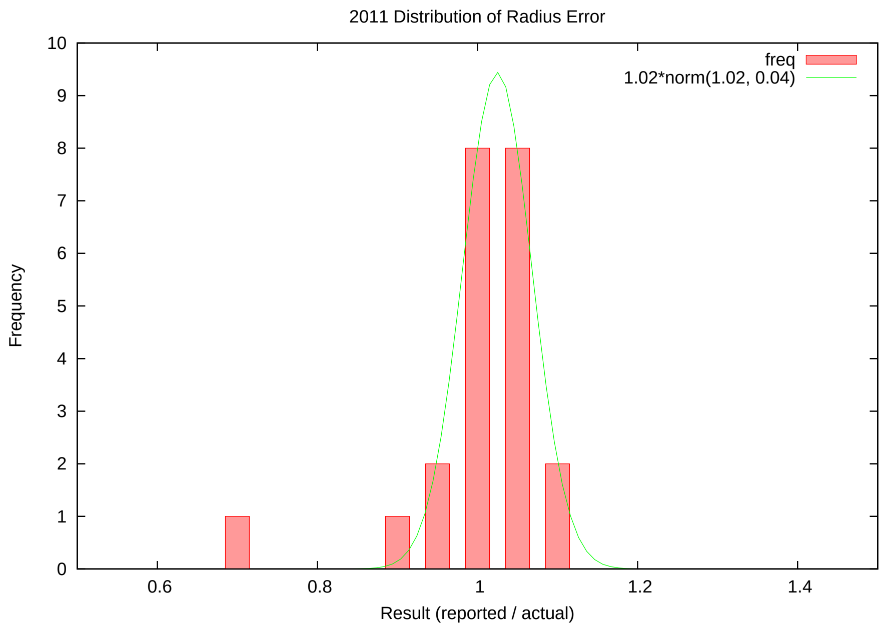
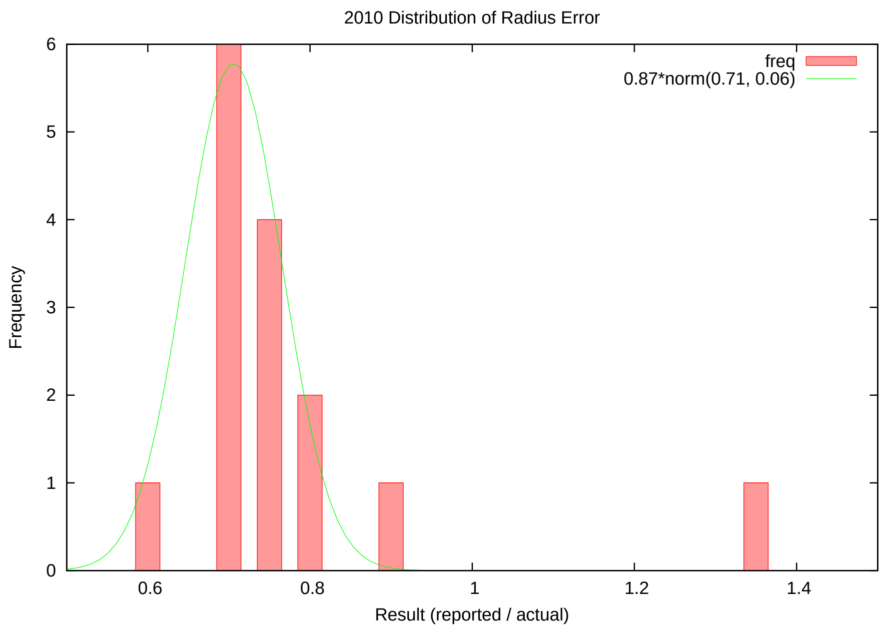


Figure 5.6: Far Ball Test Set: Radius Error Distribution.



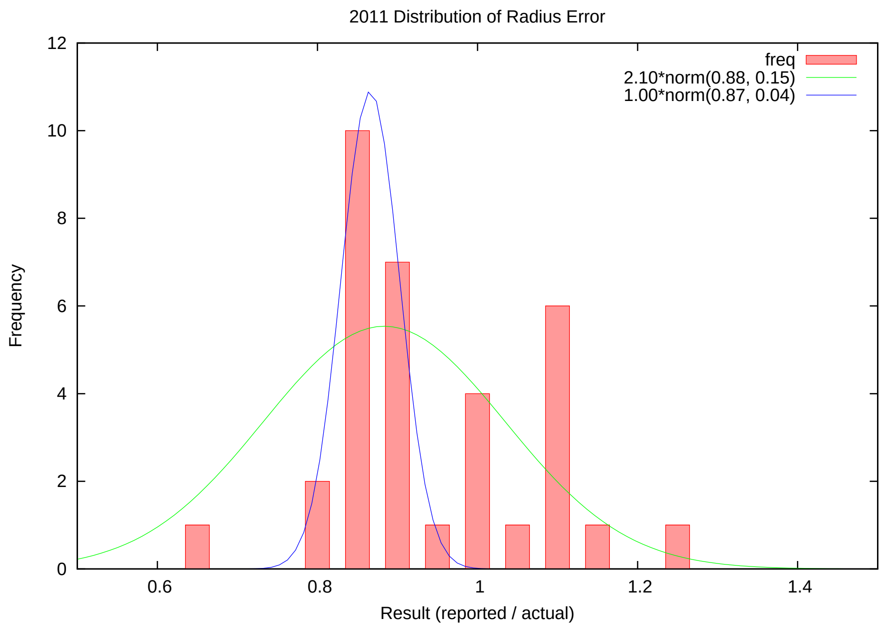
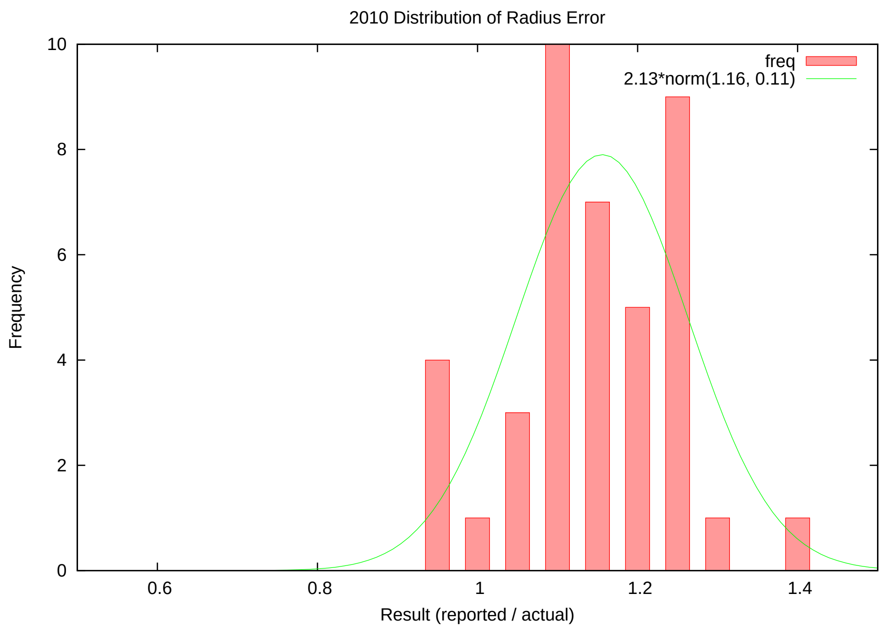
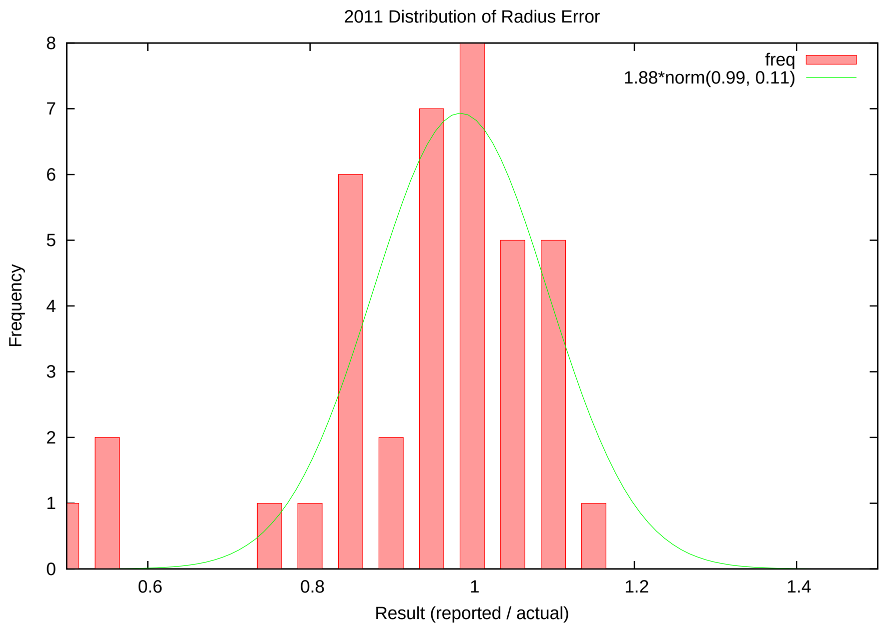


Figure 5.7: Sprint Test Set: Radius Error Distribution.





our team has now been addressed through careful code design.

Even with careful design, the vision system remains highly intricate system that will unavoidably require more and more comprehensive testing as it evolves. Testing a vision system is by nature a difficult problem, however we have provided for the first time a system for statistically measuring our system's performance.

Despite these improvements, making only the quarter finals was a disappointing result. Even so, our vision system has already proved its metal against the other top teams, and this year's infrastructure upgrades and additions make it one of the most flexible and advance systems around.

5.3 Future Work

The immediate experiment that comes to mind upon viewing the ball detection results is to create a hybrid detector and measure its properties. Both the 2010 detector and 2011 detector work by first identifying a ring of ball edge points, and then fitting a circle. The superior detection rate of the 2011 detector suggests that it does a better job of finding these edge points, whilst the 2010 method of circle fitting seems more robust to blur and the resulting shape deformation. Ideally, a best of both worlds solution may be achievable.

Prioritising work to be done on the overall vision module is not as straight forward. RoboCup, almost by definition, is a never ending torrent of future work. The improved hardware introduced into the Nao version 4 will for the first time in the Nao's history provide a reasonable amount of processing power, and effectively utilising this extra processing processing will no doubt be a challenge in itself. However, there remains one key hole in the vision infrastructure that was never completed: fully automated vision metrics and testing.

Whilst a comprehensive testing system will not directly improve the vision module by itself, statistically metering progress is a requirement for proving that modifications made to the system have actually improved its performance. The fundamental functionality for such a system already exists within the current vision test suite, however generating meaningful graphs and metrics remains a manual process. Automating this process would be by no means a trivial task, but has the potential to pay dividends.

5.4 Conclusion

This thesis has focused mainly the enhancements made to the existing system that were necessary before additional features could be added. A huge amount of work was done to reduce the CPU time consumed by the 2010 vision module, and these reductions have freed up the necessary resources needed by the new robot and field line detectors. In this regard, the efforts were successful.

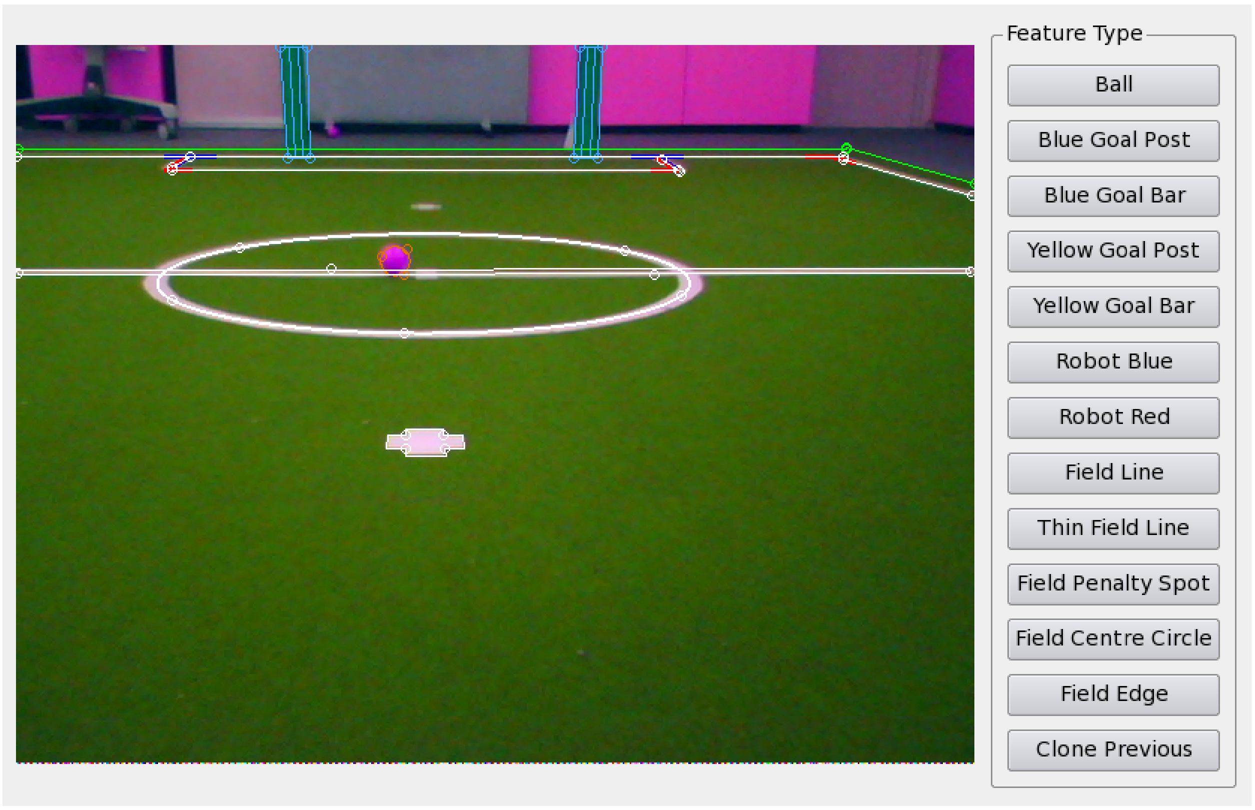
The more interesting results arose from the statistical analysis of the ball detector. Never before has rUNSWift analysed a detector in such detail, and the results were somewhat enlightening. The underpinning conclusion learnt is not that foveated ball tracking improves distant ball detection or that last years circle fitting method is resilient to blur; the deeper lesson is that rUNSWift could potentially gain an advantage over its competition by applying scientific principles when evaluating its results. At face value the 2011 ball detector seems to detect more balls in more situations and is therefore better. Had the statistical analysis not been carried out, the surprising properties of the circle fitting method used by Ratter et al. may not have been realised. A potentially valuable contribution to the rUNSWift system would have been thoughtlessly discarded, and I would have been at fault. In conclusion, the use of scientific method could potentially result in discoveries that are both applicable to the game of SPL soccer and also worthy of the title “science”.

Appendix A

Image Classifier Manual

A.1 Object Classification

The image classifier provides a simple interface for classifying regions of the image as objects.



Object Creation

To add a new object, simply click the corresponding button from the menu on the right of the image. The newly created object will appear in the centre of the image.

Control Points

Each shape is defined by a set of control points. These points can be selected and dragged to match the classified objects shape.

Movement
: Sometimes it is more convenient to move the object as a whole instead of dragging each control point individually. Clicking anywhere within an object and dragging the mouse will translate all control points simultaneously.

Object Selection
: Clicking either within an object or one of its control points will select the object. The currently selected object's edges are boldly highlighted.

Object Deletion
: An object can be deleted by first selecting it and then pressing the delete key.

A.2 Image Navigation

Classifying small objects can be fiddly, but the image classifier includes several features to ease the classification of fine details.

Zoom
: Zooming in and out can be achieved using the scroll wheel. Intuitively, the region under the mouse will remain centred in the image.

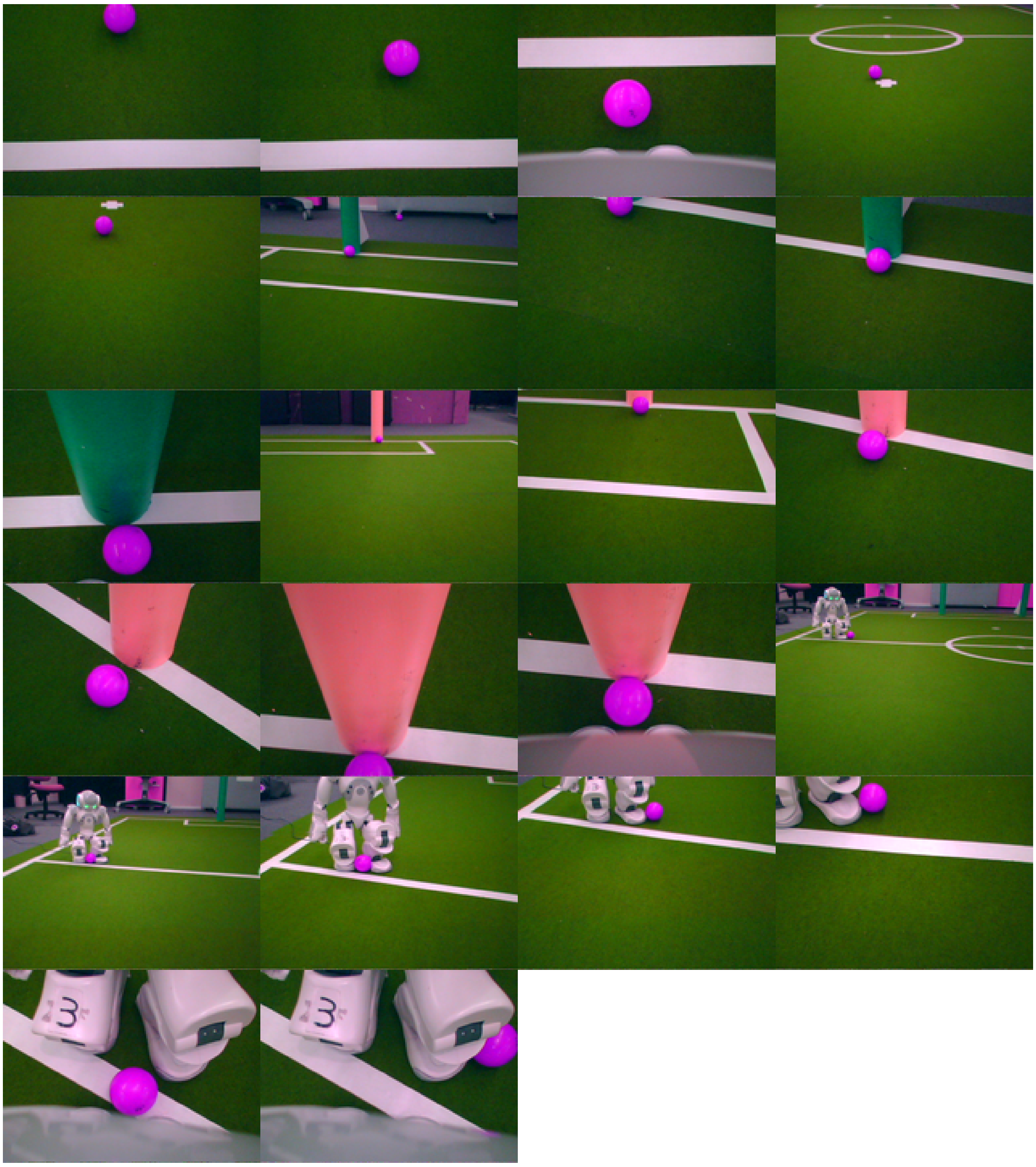
Side Scrolling
: Once zoomed in, there are often times when it would be convenient to side scroll to a new location instead of zooming out from the current location and back in on the new one. Right clicking on an edge of the image will scroll the view port in that direction.

Side Scroll and Drag
: Dragging an object whilst side scrolling will cause the object to be moved along with the view port.

Appendix B

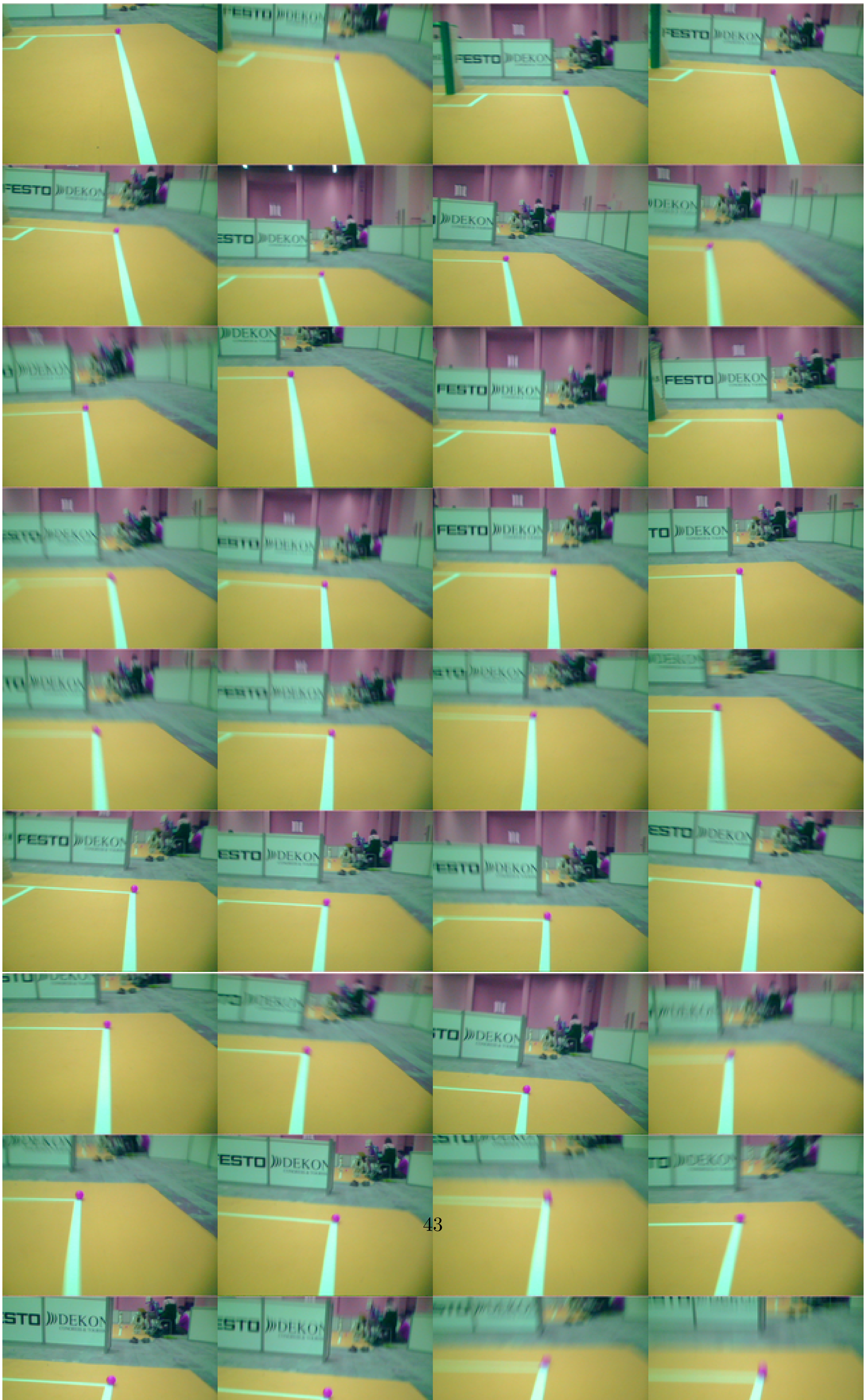
Ball Test Sets

B.1 Close Ball Test



B.2 Far Ball Test

B.3 Chase Test



Appendix C

Process to Reproduce Ball Statistics

1. Acquire the classified logs from 0xfaded.com/robocup/ball_stats.tar.gz. This archive also contains all snippets of code listed below.

2. Modify the vision tester print out error measurements. Currently, the vision tester only supports averaging of results and cannot produce histograms or fit bell curves. The compromise is to print out each score individually and processes the scores using gnuplot. The lines below need to be uncommented in VisionTestFrameResult.cpp.

   ```
       /*
       std::cout << "r: " << radius_score << std::endl;
       std::cout << "o: " << offset_score << std::endl;
       */
   ```

3. Create a configuration file for the log. The only required options are those that specify the classification file. An example has been provided.

   ```
   [vision]
   top_calibration = /path/to/top.nnmc.bz2
   bot_calibration = /path/to/bot.nnmc.bz2
   ```

4. Invoke the vision tester and save the output to a file.

   ```
   ./visiontester -c /path/to/config.cfg -d /path/to/dump.ofn > output
   ```

5. Bucket the results using bucket.pl. Both the radii and offsets need to be bucketed separately.

   ```
   grep 'r: ' | cut -c4- | ./bucket.pl > radii_bucket
   grep 'o: ' | cut -c4- | ./bucket.pl > offset_bucket
   ```

6. Plot and fit the bucketed data using gnuplot.

   ```
   gnuplot radii.plt
   gnuplot offset.plt
   ```

C.1 bucket.pl

```
#!/usr/bin/perl -w

$bucket_size = 0.05;
sub bin {
        my $x = shift @_;
        my $s = shift @_;

        return int(($x + $s/2) / $s) * $s;
}

$buckets{bin($_, $bucket_size)} ++ while(<>);
print "$_\t$buckets{$_}\n" foreach(sort keys %buckets);
```

C.2 offset.plt

```
reset
unset title

set terminal postscript landscape enhanced color solid
set output 'offset.ps'

set boxwidth bw*0.6
set style fill solid 0.4

set title "2011 Distribution of Offset Error"
set xrange [0.0:1]
set yrange [0:*]
set xlabel "Result (offset / radius)"
set ylabel "Frequency"

a1    = 0.88
mu    = 0.16
sigma = 0.05
fitted(x) = a1*2.7182818**-((x-mu)**2/(2*sigma**2))/(sqrt(2*pi)*sigma)

fit fitted(x) 'offset_new_bucket' using 1:2 via a1,mu,sigma

plot 'offset_new_bucket' using 1:2 smooth frequency ti 'freq' w boxes, \
     fitted(x) ti sprintf("%2.2f*norm(%2.2f, %2.2f)", a1, mu, sigma)
```

Bibliography

[1] *Nao Green Documentation.*

[2] *Nao Red Documentation.*

[3] Design criteria standard mil-std-1472f. Technical report, August 1999.

[4] Yannick Allusse, Patrick Horain, Ankit Agarwal, and Cindula Saipriyadarshan. Gpucv: an opensource gpu-accelerated framework forimage processing and computer vision. In *Proceeding of the 16th ACM international conference on Multimedia*, MM '08, pages 1089–1092, New York, NY, USA, 2008. ACM.

[5] Jack Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[6] RoboCup Techincal Committee. Robocup standard platform league (nao) rule book. Technical report, May 2011.

[7] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Image segmentation using local variation. In *in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 98–104, 1998.

[8] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, June 1981.

[9] Sean Harris. Efficient Feature Detection Using RANSAC. Honours thesis, The University of New South Wales, 2011.

[10] Feng-Hsiung Hsu. IBM's Deep Blue Chess grandmaster chips.

[11] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, AGENTS '97, pages 340–347, New York, NY, USA, 1997. ACM.

[12] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawai, and Hitoshi Matsubara. Robocup: A challenge problem for ai and robotics. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, volume 1395 of *Lecture Notes in Computer Science*, pages 1–19. Springer Berlin / Heidelberg, 1998. 10.1007/3-540-64473-3-46.

[13] Jimmy Kurniawan. Multi-Modal Machine-Learned Robot Detection for RoboCup SPL. Honours thesis, The University of New South Wales, 2011.

[14] Adrian Ratter, Bernhard Hengst, Brad Hall, Brock White, Benjamin Vance, David Claridge, Hung Nguyen, Jayen Ashar, Stuart Robinson, and Yanjin Zhu. Runswift team report. Technical report, 2010.

[15] Belinda Teh. Ball Modelling and its Applications in Robot Goalie Behaviours. Honours thesis, The University of New South Wales, 2011.