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

Signature-Based Heading Determination

Richard Hua (3290620)
Bachelor of Computer Science (Honours)

Supervisors: Bernhard Hengst & Claude Sammut
Assessor: Maurice Pagnucco

September 7, 2013

Acknowledgements

Thanks to all of my friends, both in the lab and outside, for your companionship over the last year, I've had some good times. Thanks particularly to Emily for your ongoing encouragement and support. Thanks also to my family for providing a great environment for me to grow over the past couple of years.

Thank you to Claude and especially Bernhard, for all the guidance and advice you gave me even when I felt things were hopeless.

Contents

Chapter 1

Introduction

In the field of robotics, the ability for a robot to localise itself is crucial to its ability to perform tasks successfully. This is a non-trivial task, as the process of localisation typically involves a long computational pipeline subject to error at each step:

1. Storing raw data streamed from hardware sensors (such as a video camera)
2. Recognising known features within the sensor data (such as a landmark in a visual frame)
3. In some cases, using a kinematic transform to map these features to a robot-relative co-ordinate plane
4. Matching the features to a known world model in order to compute the robot's position
5. Using probabilistic filtering techniques to aggregate position estimates over time and adjust for movement

The error accumulated throughout this process can be significant, especially at the feature recognition level, as it must be done in real-time, from many possible visual perspectives, and in the face of noise, changing lighting conditions, and obstructions.

The RoboCup Standard Platform League, an international robotics competition, is one domain where research into this area can be tested in a practical environment. The RoboCup SPL is a robot soccer competition held between universities using standardised robotic hardware (the 'standard platform'), where the main differentiator is the quality of the software research implemented on the robots. This is an excellent test environment, and this thesis is largely concerned with, and measured by, success in this domain.

In RoboCup, the environment is a 9 meter by 6 meter green soccer field, with white field lines and symmetrical yellow goal posts [17]. In this domain, key landmarks are traditionally field lines, goal posts, and field edges. By correlating multiple landmark observations, the robot's position and heading on the field can be determined. It is important to note, however, that due to the symmetry of the field the robot's position can often be ambiguous, with any set of in-field observations at best reducing the number of possible robot positions to two.

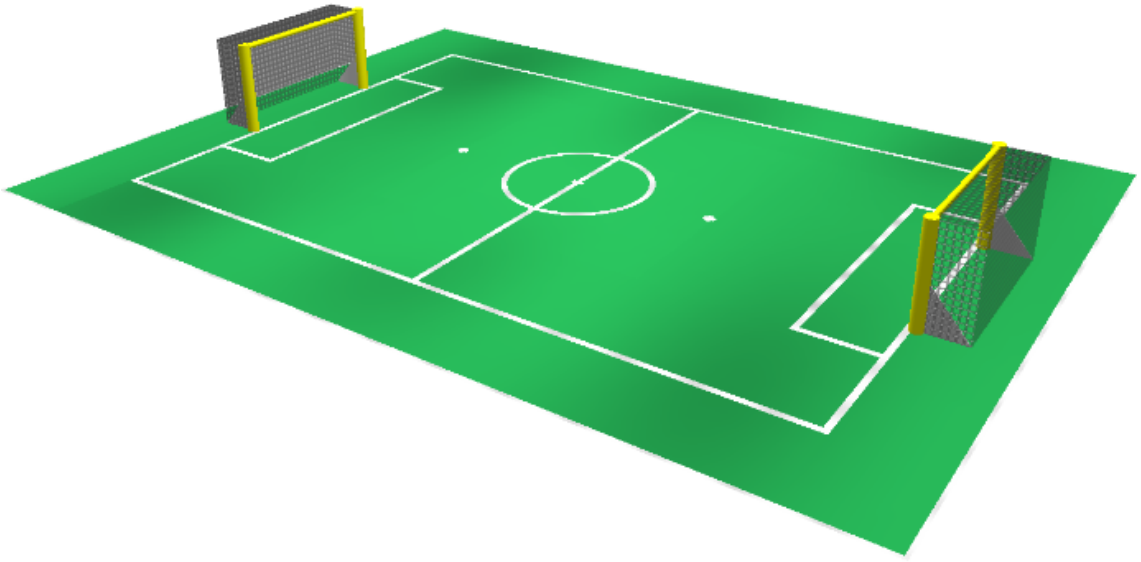

Figure 1.1: An illustration of the RoboCup field and landmarks [17].

To date, the UNSW RoboCup team (rUNSWift) has primarily resolved this symmetrical ambiguity by beginning from a known starting position and, at each time-step, combining the previous position with multiple feature observations using a modified Iterative Closest Point (ICP) algorithm to keep the robot's position up to date. This practice of correlating multiple sources of information to produce a globally consistent hypothesis has proven to be very powerful [2].

This thesis will aim to contribute another type of observation to the RoboCup system centred around heading estimates deduced using known field properties. With a huge increase in the size of the RoboCup field this year from that of the previous year, this has become important because the number of features that can be detected within the robot's field of view has decreased significantly, reducing the ability of the ICP system to correlate these features to produce a single strong observation. As well as this, being able to track the robot's heading is particularly important, especially around the centre of the field where location alone is insufficient to disambiguate between two possible hypotheses of the robot's pose.

In particular, the core detail this thesis aims to exploit is the fact that all straight field lines on the RoboCup field are either parallel or perpendicular to each other. By performing an abstract analysis of the directions that lines appear to face on the field, we are able to then deduce the robot's heading. The key difference between this and existing approaches is that it does not rely on the detection of field features such as corners or T-intersections, and does not need to match these features to a pre-existing map of the field. More specifically, this approach decouples the practice of computing the robot's *location* from its *heading*. In some cases, it will be able to determine the robot's heading where existing methods cannot, such as when there are not enough features in view to determine the robot's location.

The rest of this thesis will cover some background research (Chapter 2), followed by the general approach taken (Chapter 3). Because two distinct methods were attempted, these will be covered separately in Chapters 4 and 5, along with results. Finally, we will conclude with an evaluation (6) and conclusion (7).

Chapter 2

Background

This chapter will explore the background knowledge required to approach this problem. We will begin with a clearer explanation of the context of the problem in Section 2.1, where we will explain the wider rUNSWift feature detection system, before examining key concerns. In Section 2.2, we will explore general line-finding algorithms in computer vision. Finally, in Section 2.3 we will examine various field line detection systems in practice in the RoboCup competition, focusing on the novelties added to these pre-existing algorithms to make them work in competition. Although our emphasis on line-finding algorithms does not relate directly to the approach taken in this thesis, they help establish a general context behind feature detection and provide useful ideas, some of which were used in the line signature approach described in Chapter 5.

2.1 Problem Context

This section will aim to define the problem more clearly, by examining the context of this heading deduction system and identifying the key concerns.

2.1.1 Vision Process

Before we begin, it is important to have an understanding of the context in which this heading deduction module will run, and in particular to understand the inputs and outputs of this system. The process starts with two raw images taken from the robot's dual cameras. The Aldebaran Nao Version 4 has a top camera and a bottom camera which provide a combined vertical field of view that ranges from the feet to above eye level [1]. The raw images from the two cameras are stored in a *vision frame* (Figure 2.1), which additionally stores information such as time stamps and the kinematics snapshot, a record of the robot's stance at the time the image was taken [3].

From these raw images, *saliency foveae* are generated. These saliency foveae highlight specific features of the original images, and include colour-classified, grey-scale and edge saliency images.

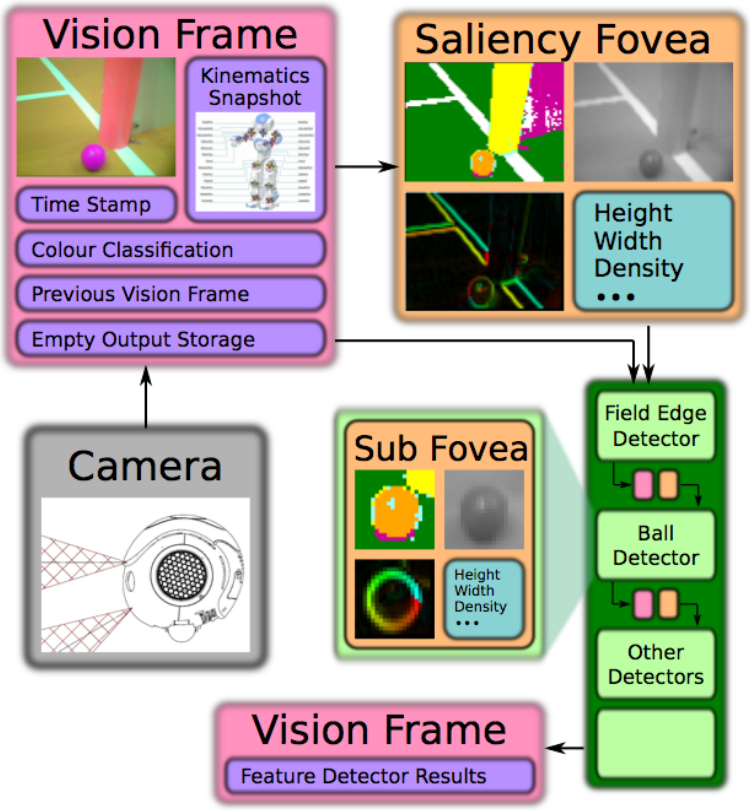

Figure 2.1: An illustration of the rUNSWift vision system. Image taken from [3, p. 7].

The term *fovea* is used generically here and in the rUNSWift code-base to describe a snapshot of all or part of an image at a specific resolution [3]. In this case, the saliency foveae are down-sampled versions of the original image for efficiency reasons (80 × 60 in 2011 and 160 × 120 in 2012). The heavy use of these foveae throughout the rest of the vision system hence subsidises the cost of generating them. For this thesis, for example, the edge saliency fovea is of particular importance - in this image, the value of each pixel is a measure of the magnitude of the colour change between it and the neighbouring pixels in the original image. Hence, the edge saliency fovea is important for determining those points which are likely to lie on line boundaries in the original image.

From here, the saliency foveae are fed to a series of feature detectors, including a field edge detector, a ball detector, and a field line detector [3]. It is important to note that the feature detectors run in a specific order and that the output of each feature detector is available to the next [10]. In particular, our heading detector will run immediately after the field line detection, and the output of the field edge detector will be known to us, an observation that will be useful for this thesis. The outputs of these detectors are stored back in the vision frame, and will be subsequently used to localise the robot.

2.1.2 Image Plane vs Ground Plane

An important concept in this thesis is the distinction between the *image plane* and the *ground plane* [10]. Points in the image plane are referenced according to the co-ordinates of that pixel in the image - in the rUNSWift code base, (0,0) is taken to refer to the top left corner of the image. Points in the ground plane, however, refer to the physical location of the observed point on the ground relative to the robot's feet. In the rUNSWift code base, co-ordinates on the ground plane are taken with the point (0,0) at the robot's feet, the first co-ordinate measuring the distance in the sagittal plane and the second co-ordinate measuring the distance in the coronal plane. Hence the point (1000,0) would represent the point on the ground 1000 millimetres in front of the robot [10]. As seen in Figure 2.2, points in the ground plane are much more useful because they more accurately reveal whether lines are parallel or perpendicular, a key piece of information in this thesis. In order to use this information, points must be mapped from the image plane to the ground plane. This is achieved by determining the robot's perspective and compensating for it. The kinematic chain developed by the rUNSWift team is able to determine this relationship by taking all of the robot's joint angles into account [21].

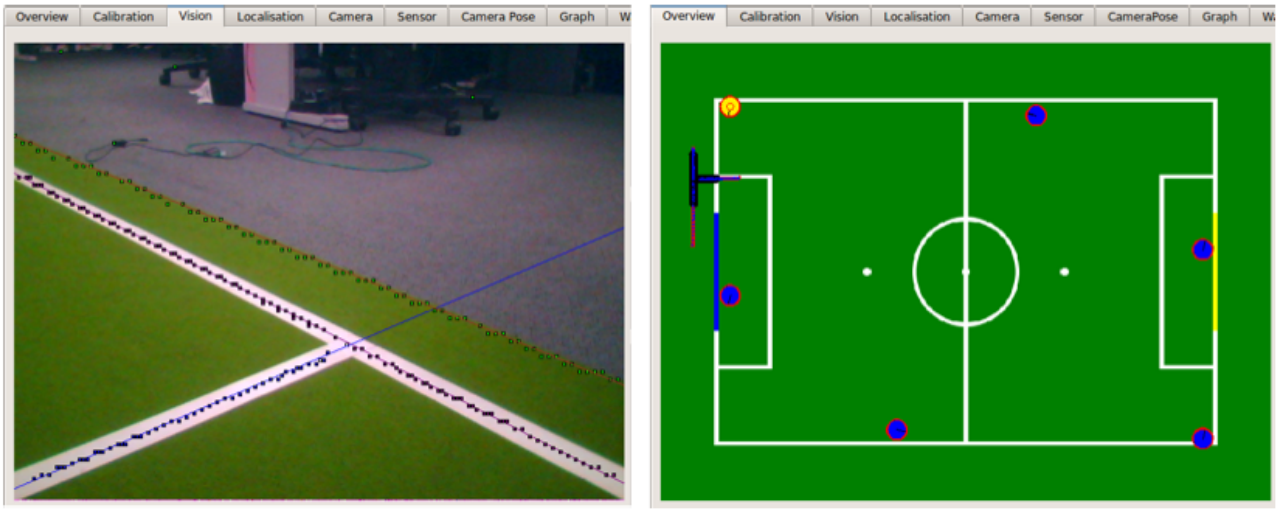

Figure 2.2: In the image (left), the field lines do not appear perpendicular, but when projected onto the ground plane (right), it becomes apparent that they are. Taken from [10, p. 5].

2.2 General Line Finding Algorithms

This section will aim to provide further background by exploring general line finding algorithms used in computer vision — merge-and-split, the Hough transform, and random sample consensus.

2.2.1 Merge-and-Split Algorithm

The *merge-and-split algorithm*, also known as the *Ramer-Douglas-Peucker* algorithm, is one of the earliest proposed line-fitting algorithms [20]. This algorithm seeks to create a simple approximation of a curve using a series of connected line segments. This makes it an interesting approach for detecting RoboCup field lines, which can essentially be considered to be connected line segments.

In the merge-and-split algorithm, both the input and output consists of an ordered list of points representing the end-points of connected line segments:

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3), ..., (x_{n-2}, y_{n-2}), (x_{n-1}, y_{n-1}), (x_n, y_n)\}.$$

The aim is for the algorithm to remove points from the input list to create an output that is a simpler representation of the curve [20].

The recursive algorithm begins by marking both the first and end points to be kept (representing the segment joining the start and end points of the curve), and then uses a divide and conquer approach to improve its approximation as follows [15]:

- The perpendicular distance from every remaining point to the line segment is calculated.
  - All points within a certain error threshold ϵ are assumed to belong to that line segment and are immediately discarded.
- Of the remaining points, the point with the greatest distance from the line segment is marked as kept, in effect splitting the existing line segment into two.
- The algorithm reruns on each line segment, until all points have been either discarded or marked as kept.

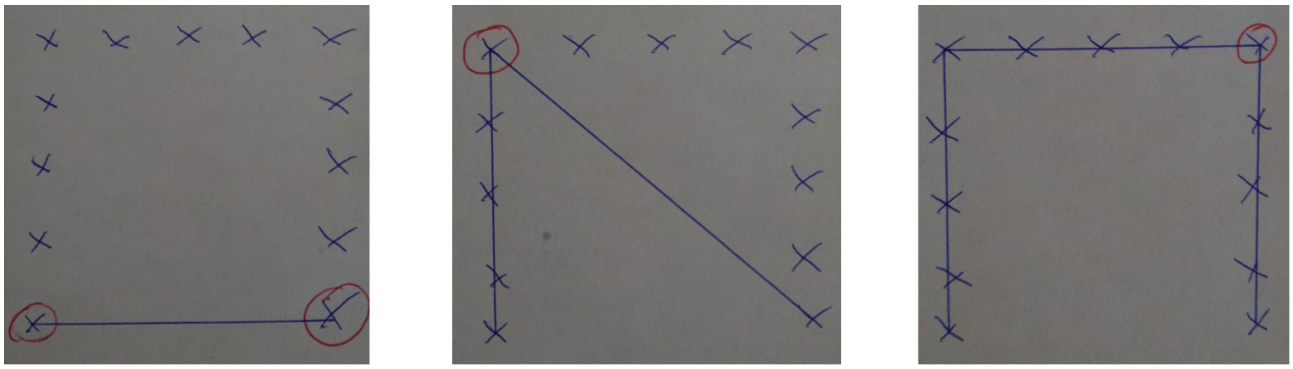

Figure 2.3: A hand-drawn illustration of the algorithm on a set of points.

This algorithm boasts a number of advantages, including simplicity and easy detection of valid corner points, which can assist with localisation enormously. However, there are a number

of obvious robustness issues which would need to be addressed in any implementation for RoboCup:

- It requires some knowledge of the ordering of points, and in particular the start and end points at the very least.
- In the algorithm's unmodified state, the input must always be a curve with a single start and end-point, however certain features of the RoboCup field will not fulfil that definition (for example, T-intersections).
- It is extremely sensitive to noise - there is no mechanism to deal with situations where a random point appears somewhere in the image.

2.2.2 Hough Transform

The *Hough Transform* is a commonly used vision processing algorithm. The problem of noise is addressed in the Hough Transform using a voting mechanism.

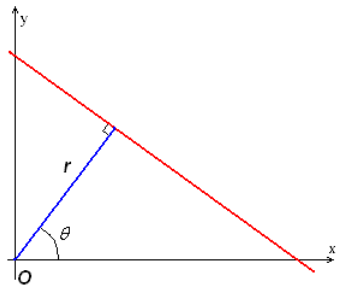

Figure 2.4: A line with (r, θ) co-ordinates.

In the Hough Transform, the aim is to deduce the equations of all lines inside an image. One of the key modifications to the Hough Transform after it was originally proposed was to represent lines using (r, θ) polar co-ordinates [5]. A line with (r, θ) co-ordinates in this representation is tangent to a circle with radius r, at the point where the radius forms an angle θ with the x-axis. This representation is used, as opposed to the (m, b) parameters of the traditional line equation, because the latter is incapable of representing vertical lines.

By trigonometry, the equation of a line (r, θ) is:

$$y = -\frac{\cos\theta}{\sin\theta}x + \frac{r}{\sin\theta}$$

Or, when rearranged [5]:

$$r = x\cos\theta + y\sin\theta$$

The basic operation of the algorithm consists of iterating through the input points and tallying votes for all possible lines that pass through it in an *accumulator array* [13], which is essentially a two dimensional array indexed on r and θ. This array will contain a discretisation of every possible value of r (given the size of the input image) and every possible value of θ, and hence encapsulate every possible line in the image. After this tallying process is complete, the local maxima of the accumulator array are selected as the lines present in the image. The pseudo-code is as follows.

Algorithm 1 The Hough Transform

```
initialise all entries of accumulator[][] to 0
for all input points (x, y) do
    for every discrete value of θ do
        r ← x cos θ + y sin θ
        accumulator[r][θ] ← accumulator[r][θ] + 1
    end for
end for
initialise lineList as empty list
for all entries in accumulator[][] do
    if current entry is greater than all surrounding entries then
        Add current (r, θ) pair to lineList
    end if
end for
return lineList
```

Note that this is the most basic version of the algorithm, as many optimisations exist to improve both speed and accuracy. For example, a Gaussian blur is commonly applied before the process of finding the local maxima [13].

This algorithm has the advantage of robustness, and the fact that all lines can be detected in a single pass through the input points. However, it suffers from the fact that lines are represented in a form without a start or end point, making it difficult to use in applications where this knowledge is required. Furthermore, the Hough Transform has a reputation for being computationally intensive due to the large size of the accumulator array, and cannot be run in the constrained environment of RoboCup without heavy optimisations [14].

2.2.3 Random Sample Consensus

The *Random Sample Consensus* algorithm, also known as *RANSAC*, uses heuristics coupled with randomisation techniques in an attempt to speed up the line finding process. These heuristics are able to effectively deal with noise by incorporating aspects of voting and techniques such as measuring the mean squared error of line estimates.

At its core, the RANSAC algorithm works according to the following method [7]:

- Fit a line based on a randomly selected subset of the input points.
- Perform a heuristic measure of how 'good' the line is in fitting the input points.
- After a fixed number of iterations, or when a sufficiently good line is found, end the process and select the best line.

This process is repeated for as many lines as required.

An important aspect of this algorithm is the use of *inliers* [16], or points which fit a certain data model, as opposed to *outliers*, which do not fit the model and can be considered to either be noise or to fit to some other model. In the context of the algorithm, an inlier is defined as a point that is within a pre-defined distance from the line. Frequently, when a line does not have enough inliers, the line is immediately discarded, and the heuristic measure is not applied. Sometimes, when the line has sufficient inliers, it will first be refitted based only on the inliers identified. This can generate more accurate line estimates before the heuristic measure is applied.

Another important aspect of this algorithm is the choice of heuristic used to evaluate lines and parameter values. Commonly, the heuristic will account both for the number of inliers and some measure of the average error of the inliers from the line [16]. There are also a number of parameters to consider, including the number of points used to generate line estimates, the distance from a line within which a point will be considered an inlier, the minimum number of inliers required for a line, and the number of iterations used to find a line.

This algorithm is well-suited to cases where the proportion of inliers is high, as this results in a high probability of correctly guessing the line and terminating early. The difficulty with RANSAC is the high level of randomness, with volatility in run-time varying significantly depending on the quality and volume of the input points. Furthermore, the RANSAC algorithm has a heavy dependence on a large number of parameters that require a large amount of tuning based on the specific application. However, if these weaknesses are adequately addressed, the RANSAC algorithm can potentially be made to run very quickly when compared to other general purpose algorithms.

2.3 Line Finding in RoboCup

In this section, we will explore these line finding systems in practice, with an emphasis on novelties introduced to existing algorithms to enable them to function in the RoboCup competition. We will explore the existing random sample consensus algorithm of rUNSWift, the Hough transforms of the Northern Bites and the UPennalizers, and finish with an overview of the clustering algorithm used by B-Human.

2.3.1 rUNSWift

Implementation

The 2011 rUNSWift team used an implementation of RANSAC running on the robot. One of the interesting aspects of this particular implementation is the emphasis on generating quality

candidate points. A number of steps were taken to ensure the quality of the input points were as high as possible, including sanity checks such as checking for field line colours [10]. To increase the accuracy of the algorithm, the points were also projected from the image plane into the ground plane. The RANSAC process was further sped up by identifying field line boundaries (green-white edges) in order to combine edge points from either side of the field line into one point in the centre of the line.

The 2011 algorithm was also able to detect circles. As the RANSAC line algorithm would often detect parts of circles, and the RANSAC circle algorithm would often detect parts of lines, Harris [10] compromised by combining the two algorithms and running them in parallel, before selecting the better result out of the two. The difficulty lay in devising a heuristic that could accurately determine which shape was better, however he found that a relatively simple variance calculation was sufficient.

Results

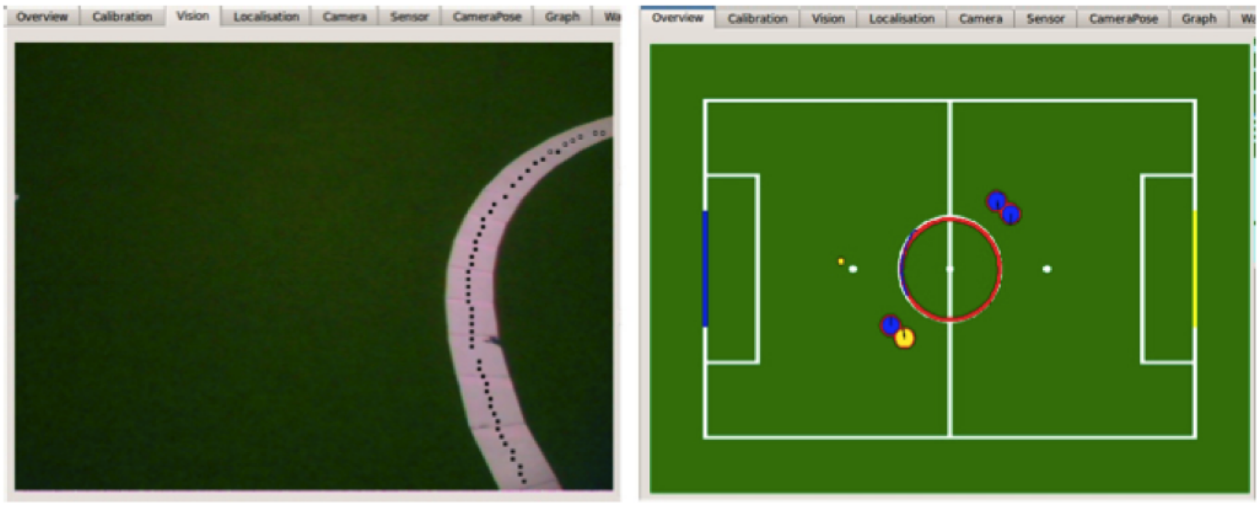

Figure 2.5: The RANSAC algorithm detecting the centre circle even when only a part of it was visible [10].

The vision algorithm developed in 2011 was a huge step forward for the team as no equivalent system had existed previously. It was able to accurately detect lines and match them to field features correctly in a large variety of situations, as well as distinguish lines from circles, providing a vast improvement in competition performance.

However, it also suffered from a number of flaws. It was unable to detect lines further than 1.5 metres away [10], largely because it operated on an 80 x 60 edge saliency fovea. This in turn had its root in the performance of the algorithm, which struggled with the busier frames even at this resolution. Figure 2.6 shows the high volatility in the run-time of the algorithm,

particularly in terms of the candidate point selection. However, this improved somewhat in the following years as processing power, and hence fovea resolution, increased on the robots.

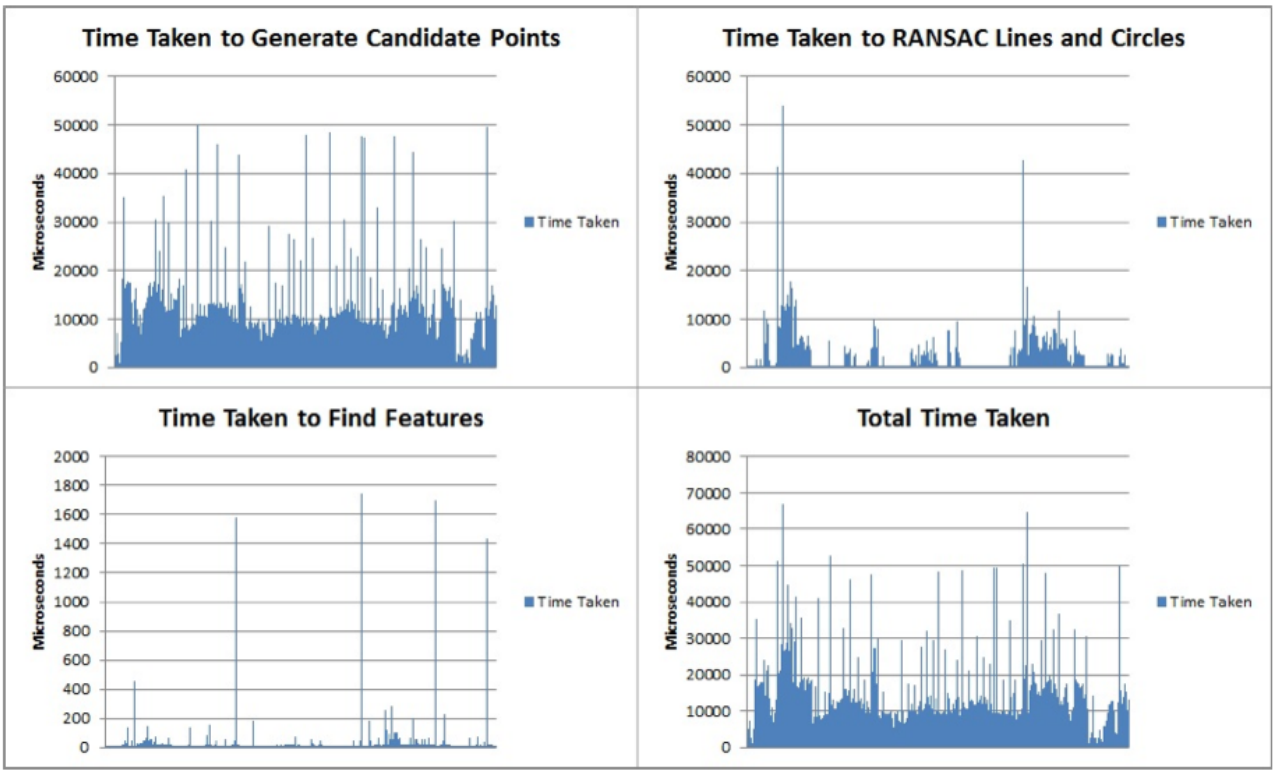

Figure 2.6: The performance of the RANSAC algorithm [10].

From previous analysis using GProf [12], we concluded that the process of generating candidate points takes up an overwhelming majority of the computation time. This is consistent with the description of the large amount of effort taken to sanity check these points, transform them into the ground plane, and to merge points from either side of the field line into one. Unfortunately, speeding up this process is non-trivial; for example, if points from either side of the field line were not merged, the line detection algorithm could be expected to take double the computation time.

Binning Algorithm

An interesting aspect of the 2011 paper [10] was its description of another approach that was originally taken. The novelty was to pre-group the input points by using the gradient direction outputted by the edge detection algorithm. Once these points had been pre-grouped, the RANSAC line algorithm could be run in each group independently and detect the lines in minimal time, due to the high proportion of inliers. Furthermore, a RANSAC circle algorithm could be run on the remaining points to detect circles.

Unfortunately, while this approach was successful in detecting lines, it was unable to adequately

differentiate between lines and circles. More specifically, the centre circle generated a large number of short line segments.

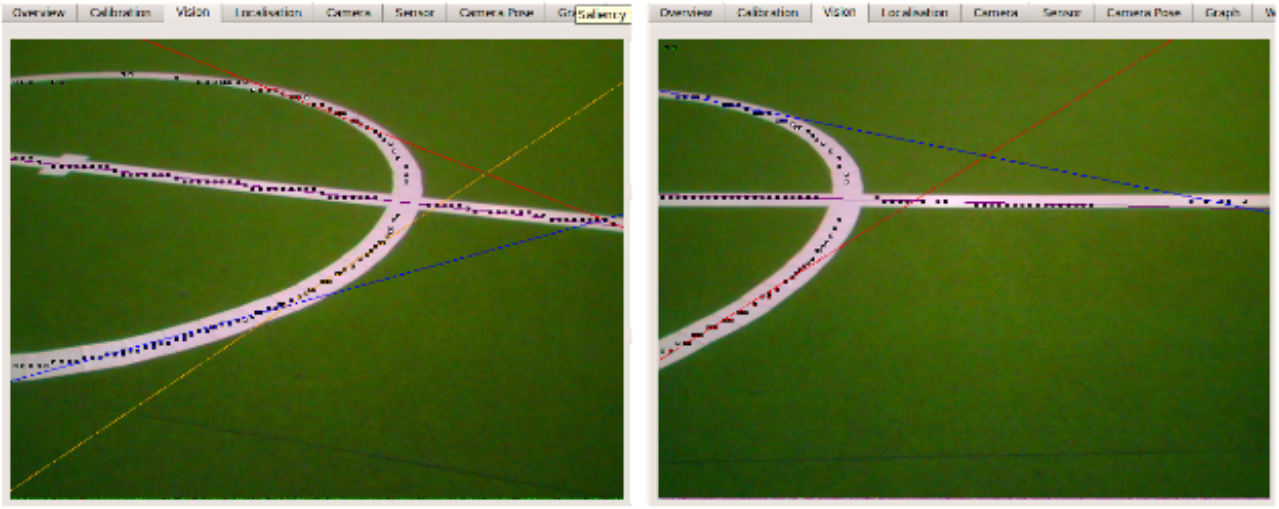

Figure 2.7: The RANSAC algorithm detecting line segments in the centre circle [10].

Although this approach was not successful, it remains promising and could potentially be adapted in some way. The pre-grouping of the points based on gradient angle, for example, could be used as a pre-processing step to the Hough Transform. The use of the points' edge gradient information is interesting and another way of using this information will be explored in Northern Bites' Hough Transform.

2.3.2 Northern Bites

In 2012, the Northern Bites implemented a Hough Transform which was notable for its heavy use of optimisations to speed up the process. Some of these optimisations, detailed in their Symposium paper on the Hough Transform, included [14]:

- Removing the use of floating point angles, and instead representing angles with single 8-bit chars. This meant that instead of a scale from 0 to 2π, a scale from 0 to 255 was used to represent a full revolution. This served to shrink the size of the accumulator array (and hence speed up the process of finding maxima), as well as speed up the calculations in the algorithm. It also meant that more values could be loaded into registers simultaneously, which will be detailed later on.
- Pre-computing the costly arctangent function, as well as the division operation, for all possible values in the edge detection process. This was enabled by the restrictions on the angle values, which reduced the size of the lookup table.
- Using the original edge gradients of the points. Instead of incrementing the accumulator array at every possible line passing through a point, only lines with an angle of ± 5 were incremented. This would likely have also increased the accuracy of the lines detected.

However, the most significant novelty in their approach was to strategically write parts of the algorithm in assembly language. This enabled them to use certain native features of the Geode processors of the Naos which were not fully utilised by the C++ compiler, in particular [14]:

- Concurrent hardware pipelines. The Geode LX processor had both Integer Unit and Floating Point Unit pipelines that could be operated simultaneously.
- Vector instructions in the Floating Point Unit. The Geode LX processor could operate the x86 MMX instruction set, which was capable of performing complex operations on multiple values in a single 32-bit or 64-bit register. In essence, this meant that, for example, four angles could be loaded into one register, and an operation could be applied on all of them simultaneously.
- Cache management. By pre-fetching values into the fastest cache using the 3DNow! instruction set, the I/O bound of the algorithm could be reduced to nearly zero.

These processor optimisations were used all throughout the algorithm, but particularly in the Hough voting process. In every clock tick, vector instructions on the Floating Point Unit were used to compute locations to be incremented, while the Integer Unit was simultaneously used to increment array locations identified in previous ticks [14].

The result of these optimisations was that the implemented Hough Transform could run at full speed on images with resolutions of 320×240 pixels, with a factor of six improvement over the same implementation in C++ [14].

No comment was made on the accuracy, however the paper makes the observation that there is very little dependency on colour calibration. This helped to reduce the complexity of the algorithm, which has only three parameters.

This is a promising approach that could be used in the future to speed up the RANSAC algorithm or the candidate point generation.

2.3.3 UPennalizers

The UPennalizers implemented a Hough Transform for field line detection. Like rUNSWift, colour information was used to identify candidate points. This involved a simple process of identifying white pixels which neighboured green pixels. The 2011 team report [9] is not detailed in the description of their implementation or results, however it did describe the key innovation of speeding up the process by only considering lines that were either parallel to or perpendicular to the global maximum in the Hough accumulator array. This is an effective use of the knowledge that all straight field lines are either parallel to or perpendicular to each other, and was the inspiration for the core idea behind this thesis.

2.3.4 B-Human

B-Human used a tailored clustering algorithm based on the approach used by Heyer et al. [11] to detect lines. They paired input points into line segments and represented them using the Hesse Normal Form, which expresses a line in terms of its direction and distance from the origin. By doing so, they were able to cluster segments with similar properties based on their Hesse Normal Forms [18]. Given enough segments in a cluster and that sanity checks were passed, these clusters were classified as a line using the given properties of the cluster.

To detect circles, B-Human used the remaining segments and calculated the intersections of their perpendicular bisectors. If they generally intersected at a distance roughly equal to the radius of the centre circle, the segments would be classified as a circle [18].

This approach has proven to be very effective in competition, operating at a reasonable accuracy at a resolution of 320×240 on the Nao Version 3. However, the $O(n^2)$ run-time of the algorithm came at the expense of optimality and simplicity. Smaller line segments were often discarded [18], and many sanity checks (and accompanying parameters and thresholds) were required to ensure that false positives were detected. This makes the algorithm very specific to the team for which it was implemented.

Chapter 3

General Approach

The key aim of this thesis is to generate a 'signature' from which the robot's heading can be deduced. This is primarily achieved by using the global knowledge that all straight lines on the RoboCup field will be either parallel or perpendicular to each other. If there was a way to generate a histogram of detected angles relative to the robot, and shift this to match the expected distribution of four angles 90 degrees apart, we could then use the amount of shift to determine possible values of the robot's heading.

In this thesis, two methods of generating a signature were attempted - using the edge gradient values of salient points in the image, and using the robot-relative angles of straight lines detected from these salient points. In both cases, a 'binning' approach inspired by Harris [10] was used to pre-cluster the points based on gradient. In the first case, a gradient histogram could be used to form an 'impression' of the orientation of the field, and in the second case, the pre-clustered groups could be used as a starting point for a RANSAC line matching algorithm.

3.1 Salient Points

Before anything could be done, salient points from the image were selected. These points were chosen based on how strong of an edge they represented (via their magnitude) and whether they were likely to be a local maxima (to avoid duplicate edge points).

Algorithm 2 Salient Point Selection

```
for each input point with edge values dx and dy do
    magnitude ← dx^2 + dy^2
    if magnitude is greater than points above and below or magnitude is greater than points
to the left and right then
        if magnitude > THRESHOLD then
            Save input point
        end if
    end if
end for
```

Some simplifications were made for efficiency here. For example, the magnitude calculation here

does not reflect the real magnitude, but this is sufficient to generate a monotonically increasing relationship for our purposes. As well as this, points were only required to be a local maxima in one direction so as to avoid ruling out legitimate points when the line they belonged to was close to vertical or horizontal. Note also that colour information has not been used here at all, to reduce dependency on colour calibration. However, some colour information could be used in future as a sanity check to rule out commonly misleading points, such as those on the boundary of goal posts.

3.2 Point Buckets

Once the points were selected, they needed to be bucketed according to angle. Because these buckets needed to be discrete, 360 angles were used for the sake of simplicity. This was calculated using the following simple algorithm. Note the negative dy sign - this is simply to compensate for the fact that in our image plane, the y axis points downwards instead of upwards.

Algorithm 3 Discrete Angle Mapping

for each input point with edge values dx and dy **do**
  $theta \leftarrow atan2(-dy, dx)$
  $discreteTheta \leftarrow \frac{theta \times 360}{2\pi}$
  if $discreteTheta < 0$ **then**
    $discreteTheta \leftarrow discreteTheta + 360$
  end if
end for

This angle was then used to index into an array of point lists, in which the point was stored.

3.3 Signature Generation

The output of the previous step is subject to significant noise. The aim of this step is to successfully neutralise the noise and select the dominant angles at which the greatest amount of points lie. This was achieved by tallying the number of points at each angle and applying a strong Gaussian blur, with a kernel of length 31 and σ^2 set to 64.

Once this had been achieved, local maxima were selected to represent dominant angles. The best results were achieved when each angle was checked to be the largest within a big radius (of size 20). Furthermore, a pre-Gaussian blur threshold and a post-Gaussian blur threshold were used to sanity check for peaks that were too small to be significant.

Through this process, two ways could be used to generate a signature. In Chapter 4, we describe a method by which each salient point's edge gradient was mapped to the ground plane

before clustering, creating a signature that is immediately usable after this step is complete. In Chapter 5, the salient points were left in the image plane, and the clustered groups identified by this process were used to find straight lines which could be then mapped into the ground plane and used to generate a signature. This will be described more in those chapters.

3.4 Heading Deduction

Regardless of the method of signature generation, the output of the above process is a list of robot-relative line angles. The process of deducing a heading hypothesis once the signature has been generated is remarkably simple, and based on the observation that, regardless of the number of lines observed, there will always be four possible hypotheses of the robot's heading, each 90 degrees apart.

The justification for this observation is as follows - given an observation of a robot's relative orientation to one line, there are two possible locations of the robot relative to that line, 180 degrees apart. However, it is important to note that the line itself can be in one of two different orientations relative to the field, with each orientation being 90 degrees apart. With this in mind, there are four possible orientations of the robot relative to the field (as opposed to the line). When an observation of two or more perpendicular lines is made, the rationale is similar - the same process of logic can be applied to both line angles, and the result is that there are four possible hypotheses for the robot's position, each 90 degrees apart.

Based on this observation, we note that there will always be one robot hypothesis that lies in the range $[0, 90)$ degrees, and hence it would be more compact to represent all four possible headings using this 'primary' angle. At this stage, deducing the angle becomes simple, and can be done as follows.

Algorithm 4 Simple Heading Deduction

```
for each robot relative angle θ do
    heading ← θ mod 90
end for
```

This method of heading deduction can simply be envisioned as follows - we continuously rotate the line by 90 degrees (as we do not know how the line is oriented relative to the field) until our field-relative heading falls into the range $[0, 90)$. This also has the nice side effect of side-stepping the issues caused by the fact that the robot-relative co-ordinate plane and the field-relative co-ordinate plane are 90 degrees apart.

As well as this, this method of calculation is strongly reinforced when the field has a distinct signature, and weakened when the angles given are not consistent. In other words, provided that the input angles are strictly parallel or perpendicular, this method of calculation will

collapse all of the input angles to the same robot heading. However, if the input angles are not consistent, the output headings will not be consistent either.

By analysing the distribution of output angles, we are therefore given a measure of uncertainty (albeit a crude one that does not account for systematic errors). We can exploit this information to rule out outliers, provide an uncertainty estimate as an output to a localisation filter, and even to discard frames where the information is inconsistent, such as when too many center circle points have been included. Here, we can see the benefits of using both sides of each field line to reinforce observations, and we are able to exploit our global knowledge of how field line angles must be distributed, something that is not directly utilised in feature-based approaches.

Although the localisation filter would then need to disambiguate between the four possible angles generated, we argue that this is a very worthwhile contribution when combined with other information to provide a coherent world view. As mentioned previously, this practice of correlating multiple sources of information to produce a globally consistent hypothesis has proven to be very powerful [2]. Furthermore, it is likely that a localisation filter would be able to apply this information over several time-steps and maintain a dominant hypothesis. At the very least, this approach will provide fine-tuning to complement the heading generated by existing methods and provide an estimate in some cases where existing (feature-based) methods fail.

One difficulty with this approach lies in the fact that the output is circular (an angle of 90 degrees is equivalent to an angle of 0 degrees). This makes it difficult to apply concepts such as expected value, mean and median, which are typically used for distributions that are linear. The median in particular becomes much less useful, as selecting a start and end data point by which to order the data set can often be debatable. As well as this, the mean suffers from wrapping problems - for example, the average of 89 degrees and 1 degree should be 0 degrees, not 45 degrees.

Fortunately, there exists an established body of literature dedicated to the analysis of circular data [8] [4]. One of the simpler approaches to calculating the mean is to map each data point from an angle to a cartesian co-ordinate on the unit circle. We can then average these co-ordinates (on x and y separately) to produce the mean, which will be at an angle that best represents a middle ground between all the input angles. If we think about this a bit more carefully, we are essentially representing each angle with a unit vector and averaging all of these vectors to produce a mean vector. Note that the mean vector will have length 1 when all data points are identical, and length 0 when the data points are in complete disagreement (on opposite sides of the unit circle). The length of this vector can therefore give us a final heuristic for the consistency of the data [8].

To detect outliers, we followed a very simple algorithm. As long as there existed a data point that was greater than 15 degrees from the mean, we discarded that data point and recalculated

Algorithm 5 Mean Heading Computation

$sumY \leftarrow 0$
$sumX \leftarrow 0$
for each data point θ in the range $[0, 90)$ **do**
    $sumY \leftarrow sumY + \sin 4\theta$
    $sumX \leftarrow sumX + \cos 4\theta$
end for
$meanY \leftarrow \dfrac{sumY}{numPts}$
$meanX \leftarrow \dfrac{sumX}{numPts}$
$mean \leftarrow \dfrac{atan2(meanY, meanX)}{4}$
$consistency \leftarrow \sqrt{meanY^2 + meanX^2}$

the mean. If there is ever a stage where we have less than 3 data points, there is not enough certainty in the frame and no heading observation is reported for that frame. This has some desirable effects:

- Because our goal is to only provide estimates that are reliable, our 15 degree threshold ensures that the data set we use to calculate our estimate will always be reasonably consistent.
- If the results are simply inconsistent in general, then the algorithm will keep discarding points until there are less than 3 data points, at which point it will give up on the dataset. Note that it is almost impossible for more data points to be discarded than left over - in general, there are at most 6 data point candidates in any one frame.
- The threshold of 3 data points is just enough to have a good estimate (provided the consistency between these points is good), and not so prohibitively high that too many frames will be discarded.

Once this process is complete, we also use the consistency value we calculated from the length of the mean vector as a sanity check. It is important to note that the consistency value doesn't decrease linearly with angular spread. In particular, the length of the mean vector will decrease slowly initially as angular spread increases, before decreasing at an accelerating rate. For this reason, we choose to accept a heading estimate only if the consistency value is greater than 0.85.

We are now able to supply our heading to a localisation filter. It is also desirable to provide some kind of uncertainty estimate - it is unclear whether the consistency value we calculated will be a satisfactory measure. As an alternative, we also calculated the standard deviation from the mean using the traditional approach. We will look at how well these reflect error in the results sections of the following chapters, but for now we leave both measures available to

be tested in a localisation filter. Of course, it is worth noting that there are multiple sources of error, and both our consistency estimate and standard deviation calculation only accounts for these indirectly:

- Kinematics
- Raw gradient values
- False lines.

Chapter 4

Point Signature

4.1 Methodology

The point-based approach was the first attempt at generating a signature, relying on analysing the edge gradients of salient points to create an 'impression' of the orientation of the field. In order to fully exploit the property that all field lines on the field must be either parallel or perpendicular, it was necessary to map the edge gradients of these points from the image plane to the ground plane so that they could be analysed.

However, some properties of this mapping must be taken into consideration. In particular, parallel lines in the image plane do not necessarily map into parallel lines in the ground plane, reflecting the fact that this mapping depends on the position of the point as well as the angle of its gradient. With this in mind, this mapping needs to be done on a per-point basis, before any bucketing occurs, so that the end result of the process described in Chapter 3 is a histogram of edge gradients in the ground plane.

The method by which an angle can be mapped from the image plane to the ground plane is non-trivial. Because a mechanism of mapping an image plane location to a ground plane location already existed in the rUNSWift codebase, we sought to adapt the existing mechanism to map angles, rather than attempting to write our own method from scratch. Our first approach was as follows.

Algorithm 6 Discrete Angle Mapping

function GROUNDPLANEGRADIENT($p1, dx, dy$)
  $p2 \leftarrow (p1.x - dy, p1.y + dx)$
  $rrp1 \leftarrow convertToRobotRelative(p1)$
  $rrp2 \leftarrow convertToRobotRelative(p2)$
  $\theta \leftarrow atan2(rrp2.y - rrp1.y, rrp2.x - rrp1.x)$
  $discreteTheta \leftarrow \frac{theta \times 360}{2\pi}$
  if $discreteTheta < 0$ **then**
    $discreteTheta \leftarrow discreteTheta + 360$
  end if
end function

In essence, this approach involved projecting a second point $p2$ perpendicular to the direction of the edge gradient for $p1$ (simulating another point on the line which $p1$ would theoretically lie on). Once $p2$ had been calculated, both points were mapped into the ground plane, mimicking the action of mapping the line into the ground plane. Finally, once this has been achieved, the angle of the line in the ground plane is calculated.

There are some interesting points to observe here.

Firstly, this approach relies on the fact that a line in the image plane will always map to a line in the ground plane, and hence the second point $p2$, if it lay on the same line, should have the same gradient in the ground plane as $p1$. Note that we cannot map the edge gradient of $p1$ directly, because a right angle in the image plane is not necessarily a right angle in the ground plane.

Secondly, the use of the $atan2$ function to calculate the gradient of the ground plane line is what allows for us to map opposite sides of a field line to point in opposite directions, 180 degrees apart, in the ground plane. By keeping the order of $p1$ and $p2$ constant in our $atan2$ function, we are in effect calculating the angle of a vector pointing in opposite directions on the line. Because this vector has $p1$ as its base, a beneficial side effect of this approach is that we obtain the ground plane co-ordinates of $p1$ in addition to its gradient.

Thirdly, we note that the final angle is not in fact the equivalent gradient angle that we started with, but gives the angle of a vector pointing along the direction of the line rather than perpendicular to it. It is possible to get the perpendicular of the line once again by shifting our final result by 90 degrees, but in practice this is not important because we already have our signature regardless. Another observation is that the robot-relative co-ordinate system is different to the image co-ordinate system, with the y axis pointing forwards from the robot and the x axis pointing to the left. In our implementation, for the sake of more intuitive angles, we mapped this angle to a co-ordinate system where y points forwards and x points to the right. This has been omitted here for simplicity.

Although this was a promising approach in theory, we quickly ran into an error. In practice, the computed edge values dx and dy had large values in our code, and projecting points using these values frequently created points that were below the bottom of the image or above the horizon line, causing undefined mappings to occur. Furthermore, it is easy to imagine our margin of error becoming significant as we approach the horizon. As a fix to these problems, we scaled dx and dy down to a maximum value of 20, and ignored points for which the computed vector extended beyond the image boundaries or field edge. In future, it may be better to derive a gradient mapping directly, perhaps by doing something akin to differentiating the ground plane co-ordinate mapping function.

4.2 Results

As mentioned in the previous section, the end result of this process is a histogram of angles in the ground plane, with particularly dominant angles identified. In theory, this would result in a signature consisting of angles separated by 90 degrees. Unfortunately, we found that this was rarely the case.

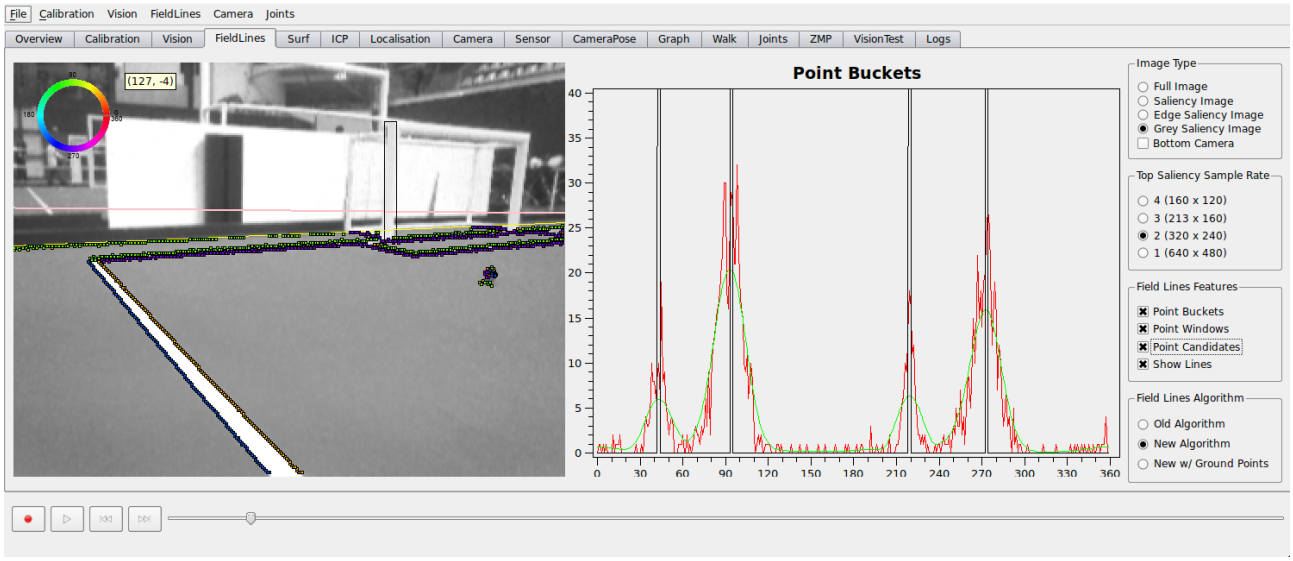

Figure 4.1: An image with points mapped in the image plane, displayed in a custom tab in the offnao debugging tool.

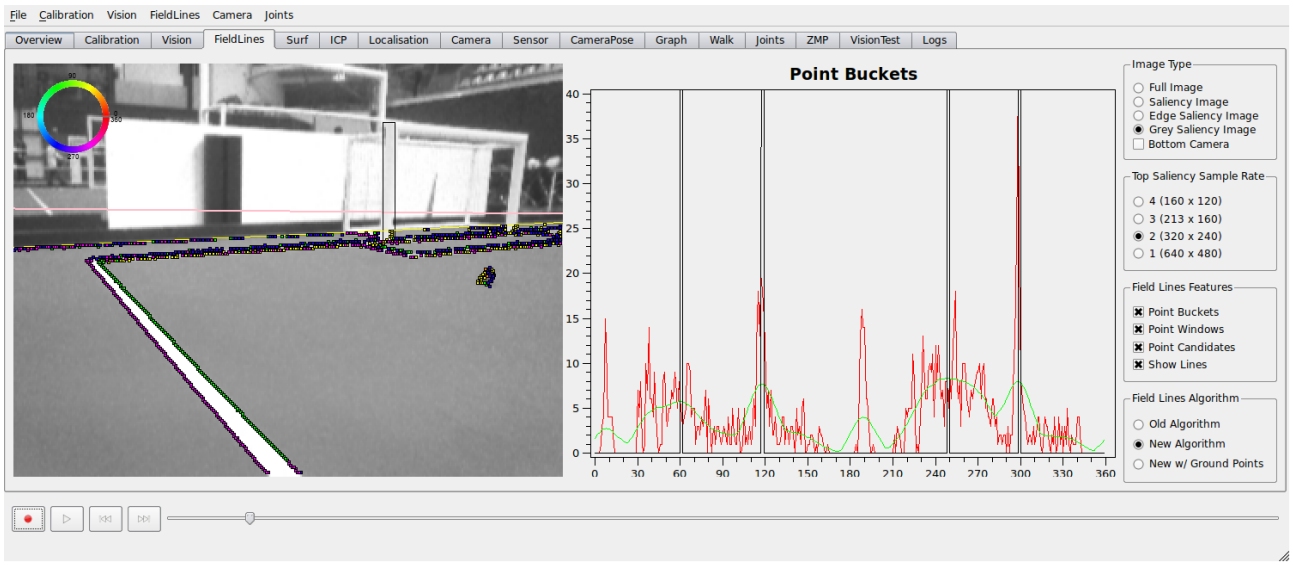

Figure 4.2: The corresponding image with points mapped in the ground plane.

In the graph on the right of the above figures, the x axis represents the angle in degrees, and the y axis represents the number of points. The red curve displays raw point counts, with the green curve displaying the smoothed result and the black peaks showing the maxima selected. As we can see, the peaks are very clearly defined in the image plane, but completely degrade in the ground plane, failing to produce any discernable signature.

Despite a lot of investigation and code review, we were unable to determine the root cause of this problem. Some analysis of the potential causes will be provided in Chapter 6. What became clear, however, is that we needed to try a different approach. In the following chapter, we attempt to create a 'line signature' by first using RANSAC to identify lines within each (image plane) bucket, and then mapping each resulting line into the ground plane to get our signature. This has the advantage that, by effectively aggregating points into a line first and mapping the end points of the line itself, the error in ground plane mapping will be significantly reduced.

Nonetheless, it is a shame that this approach was not successful, as it used a feature of the field that was very distinct from anything used by any other current vision module, and did so in a way that could be considered elegant. In future, the problem could potentially be addressed by calculating a method of converting an image plane angle into the ground plane directly, rather than using the makeshift method of mapping two points. This would also provide a significant speed up to this algorithm, reducing the number of ground plane mappings that need to be done.

Chapter 5

Line Signature

5.1 Methodology

The second approach attempted was to leave the points in the image plane, use our pre-clustered points to run very efficient RANSAC line matching algorithms, and generate our signature from the angles of the resultant lines. This approach was attempted for a number of reasons:

- By effectively aggregating points into a line first and mapping the end points of the line itself, the error in ground plane mapping will be significantly reduced.
- By significantly decreasing the number of points that need to be mapped into the ground plane, the performance of the algorithm can improve dramatically.
- The 'line signature' algorithm is not bound by the same constraints of a normal field line detection algorithm, in particular:
  - It detects both sides of the line, providing more information.
  - It has no interest in the center circle.
  - It can use the principle that all lines must be parallel or perpendicular to ensure information is consistent, and throw away information that is not. For example, it can sanity check circle segments using this principle. If the circle segment just happens to be parallel or perpendicular to the rest of the detected lines, it simply reinforces the existing heading estimate.
  - It does not need to identify field features such as corners or t-intersections. It can therefore even afford to miss several lines and still be able to determine the heading of the robot where a field line detection algorithm would not be able to (for example, when facing just the side-line).
  - It could be used as the starting point of a complete field line detection system, similar to the one attempted in [10].

In order to do this, we grouped the point buckets from Chapter 3 into overlapping 'windows' of 30 degrees, before selecting the strongest point windows in which to run our RANSAC algorithm

by using a simple local maxima. As well as this, we reused the lines detected by the field edge module, which serve as a very effective addition to the data set produced by this approach.

However, this approach generated problems of its own. In particular, leaving the points in the image plane could cause perpendicular lines to appear to have a similar angle in the image, and parallel field lines to appear to be different angles in the image. Because our method of identifying candidate points relies on detecting peaks, a smaller peak could be 'hidden' in a bigger peak and not be selected, causing that line to not be detected.

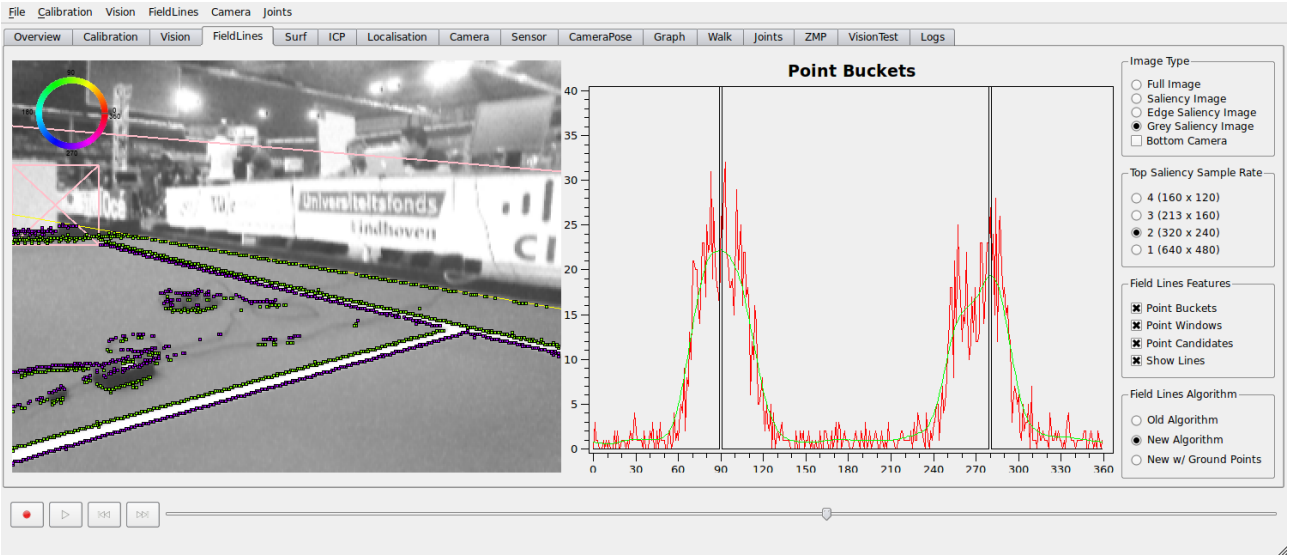

Figure 5.1: In this image, only two peaks are identified in the image graph, instead of the four expected.

In Figure 5.1, we note that only two peaks are identified in the image graph, instead of the four expected. Upon closer inspection of the graph, we can observe that there are actually four peaks, however they are close enough and at the right height to give the impression of only two peaks, as identified by the green trend line. Fortunately, in this case points from both of the lines were included as part of each 'unified' peak, rather than one of the lines being excluded.

We chose to address this problem as follows. Firstly, we set our windows to be very large, with a leeway of 30 degrees on either side of an angle, and ran our RANSAC algorithm 6 times within each window. This meant that although the smaller peaks could sometimes be 'swallowed' in the larger peaks, the large size of our point windows meant they could be included anyway and matched by RANSAC. Secondly, because our windows could now overlap significantly, we selected peaks that had the most points out of all other angles within 20 degrees. This served to reduce the overlap considerably, while allowing overlap to occur in some cases where it is uncertain which window the points would be better placed in.

For the RANSAC algorithm, we reused much of our existing RANSAC module, with several parameters modified. Firstly, because we used information from both sides of each field line rather than attempting to find the centre point, along with the fact that we used image plane points rather than the mapped ground plane equivalent, we could be much more strict about

what constituted an inlier, setting the threshold to be 2 pixels wide. Secondly, we sanity-checked lines based on whether they were 'floating' in the image. In other words, if a line had start and end points that were both not at image boundaries, this line was not 'cut off' and we could afford to be much stricter about the length it must be in the ground plane. Finally, we tuned thresholds to reflect points that lay in the image plane rather than the ground plane. This proved to remove most of the noise created by robots and, in particular, the center circle, as we will see in the next section.

5.2 Results

The goal of these experiments was to test the accuracy of the algorithm as well as the performance across different field features and distances. These experiments were conducted by recording dump files at a number of fixed locations, and then running the algorithm on the dumps offline, ensuring consistency. For each angle tested in each experiment, a minimum of 6 frames were analysed (depending on the number of frames in the dump, to prevent selection bias). Following this, the statistics for each angle in each experiment were averaged across all of the frames. The raw experiment data is provided for reference in Appendix A.

Because these experiments were run offline, the run-times recorded will not directly reflect run-times on the Nao's. The experiments were run on the following hardware:

```
Architecture:          i686
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                2
On-line CPU(s) list:   0,1
Thread(s) per core:    1
Core(s) per socket:    2
Socket(s):             1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 15
Stepping:              10
CPU MHz:               2000.000
BogoMIPS:              3989.97
Virtualisation:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              4096K
```

Because of this discrepancy, a percentage ratio was generated on our hardware by comparing the run-time of our headings module with the run-time of the existing field line detection module. This will be by no means completely accurate, and is intended to give a rough idea of order of magnitude. Furthermore, these run-times were generated simply by taking the difference between the system time at the beginning and end of each module. This would create further inaccuracies, specifically when the running thread is interrupted. However, given that we are measuring in the microseconds, measuring run-times is difficult regardless, given the overheads generated by traditional benchmarking methods. Despite these flaws, however, the run-time ratio was remarkably consistent across experiments, and is likely to be suitable for its intended purposes of producing a rough estimate.

It is important to note that in each angle of these experiments, the field was marked with two points - one for the position of the robot, and one for the position the robot was to face. The robot was placed on the first point, and was turned until it faced the second point, using a process of judgement with the naked eye. With this in mind, there is potential for human error in the ground truth heading in these experiments, estimated to be $\pm 10°$.

5.2.1 Experiment 1

Figure 5.2: Experiment 1, focusing on a field corner. Simplified headings are 0, 60, 45, and 30 degrees respectively.

The first experiment was deliberately constructed to be simple, to verify that the algorithm works as expected. For the most part, this produced good results, with the margin of error within the 10° bound.

True Heading (°)	Calculated Heading (°)	True Error (°)	Lines	Consistency	Standard Deviation (°)	Headings Time (us)	Field Lines Time (us)	Time Ratio
0	89.2	0.8	3.8	1.00	0.6	1165	3200	36%
60	65.0	5.0	5.0	0.95	3.2	1163	3058	38%
45	45.8	0.8	6.0	0.99	2.4	1169	2964	39%
30	26.8	3.2	5.3	0.98	2.8	1499	2964	51%

5.2.2 Experiment 2

Figure 5.3: Experiment 2, focusing on the penalty box. Simplified headings are 0, 60 and 30 degrees respectively.

The second experiment was constructed to be somewhat more complicated. In Figure 5.4 we see that, because of strict ground plane length threshold settings, neither the smaller lines in the penalty box nor the partial line segment on the right of the image are detected. Furthermore, we see that due to a colour calibration quirk, the algorithm has been given a false edge (which is consistent across multiple frames).

Figure 5.4: In the third phase of the experiment, there was a false edge in many frames.

However, the results remain good. In the algorithm, the false edge is detected as an outlier and removed due to its disagreement with the mean, and the mean is then recalculated. As well as this, despite the smaller line segments not being picked up, there remain enough large segments to compute a reasonably confident estimate.

True Heading	Calculated Heading	True Error	Lines	Consistency	Standard Deviation	Headings Time (us)	Field Lines Time (us)	Time Ratio
0	0.3	0.3	5.1	1.00	0.8	2204	5002	44%
60	65.7	5.7	5.1	1.00	1.6	1737	3759	46%
30	23.8	6.2	6.7	0.99	1.7	1677	4091	41%

Note that, although the error can be as much as 6.2 degrees, the results still show a high level of confidence, with the consistency value as high as 0.99 and the standard deviation only 1.7 for the third angle. It seems likely that this discrepancy could be caused by errors in placement.

5.2.3 Experiment 3

Figure 5.5: Experiment 3, focusing on the center circle and a wide view across the rest of the field. Simplified headings are 0, 45 and 45 degrees respectively.

The third experiment was constructed to test the ability of the algorithm to deal with the center circle, as well as to test the distance from which line signatures could be produced. In Figure 5.6, we see that the algorithm is unable to detect any of the goal line at this distance, although most of the sideline was detected (allowing us to see where the algorithm cuts off).

As well as this, we see that in spite of very strict threshold settings, there is a false positive detected in the center circle (a yellow line), and one of the field edges is slightly flawed. Again, the false positive in the center circle is discarded. The flawed field edge, however, was integrated into the algorithm, but the mean calculation was just barely able to produce a satisfactory estimate. This frame was somewhat exceptional out of the frames in this phase of the experiment - the center circle false positive showed up in roughly 50% of frames, and the flawed edge rarely showed up at all.

True Heading	Calculated Heading	True Error	Lines	Consistency	Standard Deviation	Headings Time (us)	Field Lines Time (us)	Time Ratio
0	88.6	1.4	3.1	1.00	1.4	2559	6451	40%
45	46.2	1.2	5.0	0.98	2.8	1582	3632	44%
45	43.9	1.1	5.8	0.98	3.2	1631	3794	43%

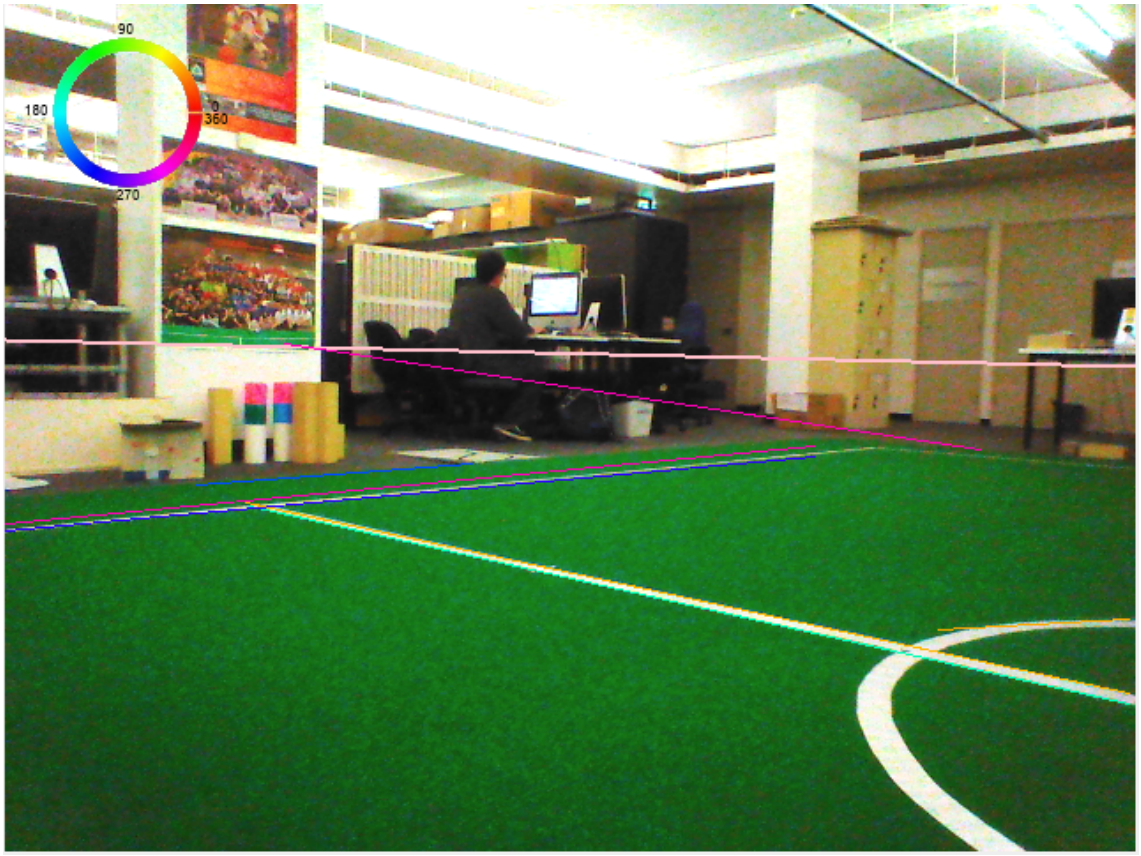

Figure 5.6: A view of the third phase of the experiment.

Chapter 6

Evaluation

From the last two chapters, we observed a big discrepancy between the point signature and line signature approach.

In Chapter 4, we found that the point signature approach suffered from a fundamental flaw - the edge gradients of points in the image plane could not be mapped accurately into the ground plane to produce a signature.

There are many possible sources of error here, some more likely than others:

- Errors in the kinematic chain, particularly when walking
- Noise in the original edge gradients
- Errors compounding from mapping two points to the ground plane and using this to calculate the angle
- Camera lens distortion

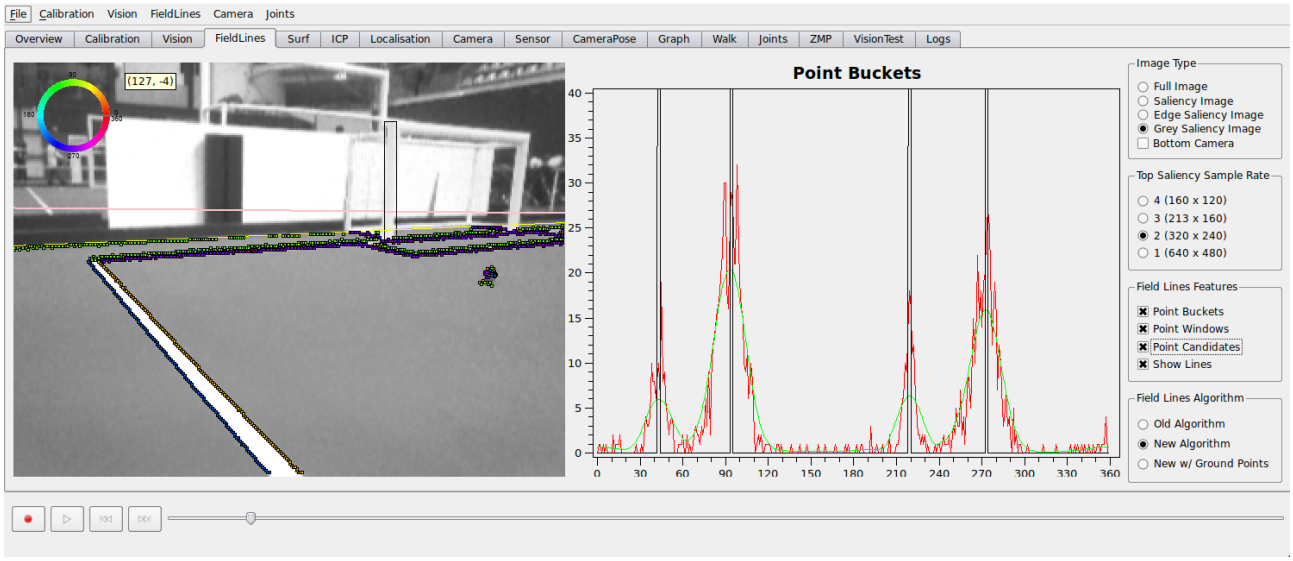

Figure 6.1: An image with points mapped in the image plane, displayed in a custom tab in the offnao debugging tool.

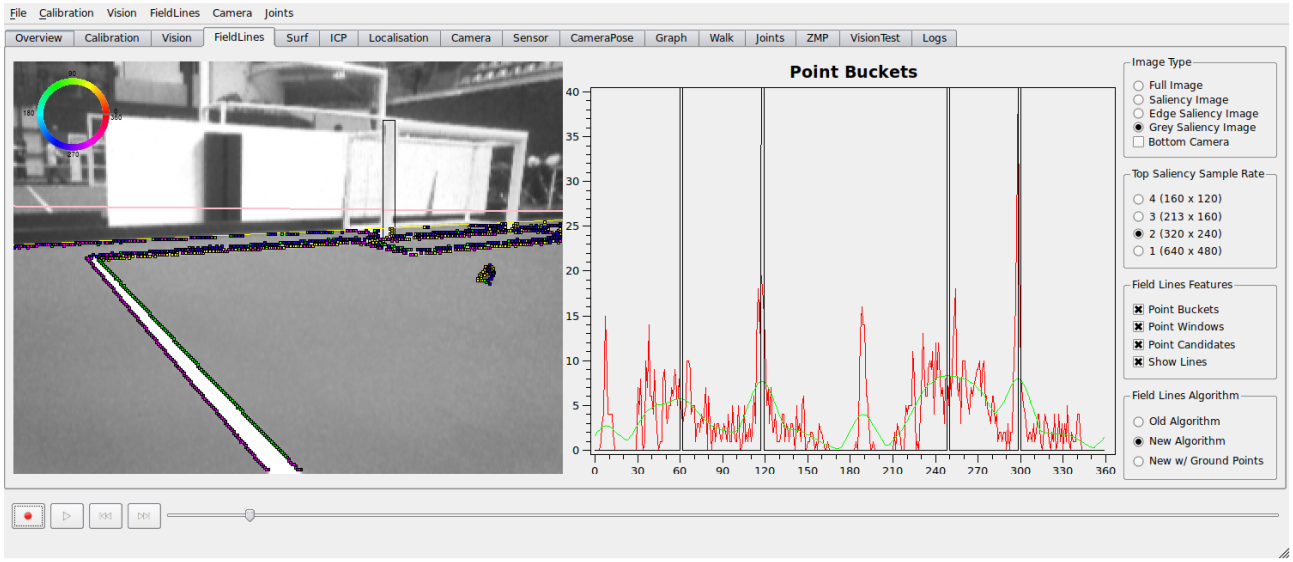

Figure 6.2: The corresponding image with points mapped in the ground plane.

It is particularly hard to guess what the impact is when these errors are compounded. One possible explanation for why the image plane graph is so clearly defined where the ground plane graph is not, is as follows. If we inspect the image plane graph (Figure 6.1) carefully, we will note that there is a peak very close to 90 degrees. This is the direction intended to be normal to the green points identified on the image. However, one would expect, by looking at the image, for this angles to be slightly greater, perhaps at 110 degrees or a little more. Furthermore, there are points on the edge of the penalty box that are green, despite clearly not being at the identified angle. Now, because the image plane mapping depends also on the location of the point being mapped, this would cause the resulting error in the ground plane gradient mapping to be of a different magnitude depending on where the point was located in the image. Because there are green points across the whole image, this would cause a range of different error values, creating the general noise and diffusion of peaks we observe in the ground plane image. As well as this, we can also observe that the horizon line (identified in pink above the field) is lower than we expect and not quite parallel to the field edge. This could also compound the error, contributing to the ground plane mapping that we observe.

Of course, this is purely conjecture. Nonetheless, this was a promising approach, and if a method of calculating the gradient directly from kinematics were devised, it could serve to simultaneously improve the accuracy of the algorithm, speed up the ground plane transformation, and maintain its simplicity.

In Chapter 5, we produced good results within our line signature experiments. The heading estimates generally showed a very high level of consistency and low standard deviation, while mostly staying within 5 degrees of the true value (which is small enough to be indistinguishable from human error). As well as this, the mechanism for removing outliers was seen to operate effectively. The algorithm generally ran in approximately 30-40% of the time that it took to run the field lines module, which is an acceptable result, although there is still room for further

improvement.

However, it is worth noting that the experiments were performed under very controlled conditions. It would be a good test to run the algorithm on a walking robot, which would effectively serve to test the accuracy of the kinematic chain under noisy conditions. As well as this, the experiments lacked the kind of visual noise that would be present in game situations, specifically when a large number of robots are present in the image (although there are a number of mechanisms that have been built and tested to address this, outlined in Section 5.1).

One of the weaknesses of both approaches is the dependency on kinematics. Although this is worse for the point-based approach, the line-based approach also needs very accurate input headings in order to work effectively. Although it could be argued that kinematics is crucial to localisation in general, discrepancies in heading are much less of an issue in a feature-based approach such as the one currently used in vision, where features such as corners and intersections can be recognised even when some of the line headings involved have an error greater than 10 degrees.

Of course, the line signature approach is currently the more reliable one, as it effectively removes noise by aggregating points into lines before analysing them. However, it is worth considering the potential of integrating this into the field line detection module, due to the overlap that is present. For example, the current field line detection module already translates all points into the ground plane, effectively saving some of the computational cost required to calculate each point's ground plane gradient. If an accurate mechanism for calculating the ground plane gradient directly could be devised, the point signature could be used as a precursor to the full field line detection algorithm and the generated heading could be used as a sanity check in addition to an output for a localisation filter. Likewise, the simple lines detected by the current field lines module could be used to generate the line signature rather than detecting the lines again for the line signature separately. However, because the current module only detects a single line at the center of each field line, some of the accuracy gained from correlating consistent information would be lost here due to the reduced number of lines available in any image. Finally, the heading module itself could be evolved into a field line detection module to replace the current one, which currently needs to run much faster.

Chapter 7

Conclusion

Localisation is an important problem in robotics, where it is crucial to every task a robot must perform. In the RoboCup competition, with the context of an increasing field size and symmetrical field features, tracking a robot's heading accurately has become particularly important, especially in positions where the robot's location cannot be disambiguated. This report has explored an idea for deducing a robot's heading that can be used to augment existing localisation systems.

One of the key differences of this approach is that it does not rely on a landmark-based approach to localisation, effectively decoupling the process of deducing a robot's heading from the process of deducing its location. The angular distribution of lines and points on the field, and the matching of this distribution to a known, 90 degree separated distribution of lines, effectively serves as a 'meta-feature' of the field.

This new approach provides some unique advantages. Firstly, it is able to use global information not explicitly used before to ensure consistency in its results. This is particularly the case when it can use both sides of field lines to reinforce observations compared to, for example, a single false positive detected in a robot body. Secondly, it does not require the robot's location to be known, allowing it to deduce a heading even when other modules have failed to identify landmarks on the field.

This approach also has some disadvantages. Firstly, it has a strong reliance on accurate kinematics. Secondly, it provides four equally probable hypotheses at any point in time, in a manner that is analogous to how localisation of position currently provides two symmetrically flipped hypotheses at any point in time. As with the latter, it is hoped that this algorithm will make a meaningful contribution when filtered with a combined set of observations to form a cohesive world view.

Bibliography

[1] Aldebaran Robotics, *Nao Specifications*, accessed 29 August 2012: <http://developer.aldebaran-robotics.com/nao/>.

[2] Peter Anderson, Youssef Hunter & Bernhard Hengst, *An ICP Inspired Inverse Sensor Model with Unknown Data Association*. The University of New South Wales, 2013. Available online: <http://cgi.cse.unsw.edu.au/~robocup/2012site/reports/ICRA-2013-ICP.pdf>.

[3] Carl Chatfield, *rUNSWift 2011 Vision System: A Foveated Vision System for Robotic Soccer*. The University of New South Wales, 2011. Available online: <http://cgi.cse.unsw.edu.au/~robocup/2011site/reports/Chatfield-Vision.pdf>.

[4] David Collett, *Outliers in Circular Data*. Journal of the Royal Statistical Society, 1980. Available online: <http://www.jstor.org/stable/2346410>.

[5] Richard O. Duda & Peter E. Hart, *Use of the Hough Transform to Detect Lines and Curves in Pictures*. Communications of the ACM, 1972. Available online: <http://www.ai.sri.com/pubs/files/tn036-duda71.pdf>.

[6] Jay Fenlason, Richard Stallman & Jeffrey Osier, *GNU gprof*. Free Software Foundation, 1993. Available online: <http://www.cs.utah.edu/dept/old/texinfo/as/gprof.html#SEC12>.

[7] Martin A. Fischler & Robert C. Bolles, *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*. Communications of the ACM, 1981. Available online: <http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA460585>.

[8] Nicholas I. Fisher, *Statistical Analysis of Circular Data*. Cambridge University Press, 1995.

[9] General Robotics Automation, Sensing and Perception Laboratory, *The University of Pennsylvania Robocup 2011 SPL Nao Soccer Team*. The University of Pennsylvania, 2011. Available online: <http://www.tzi.de/spl/bin/view/Website/Teams2012>.

[10] Sean Harris, *Efficient Feature Detection Using RANSAC*. The University of New South Wales, 2011. Available online:
<http://cgi.cse.unsw.edu.au/~robocup/2011site/reports/Harris-Vision.pdf>.

[11] Laurie J. Heyer, Semyon Kruglyak & Shibu Yooseph, *Exploring Expression Data: Identification and Analysis of Coexpressed Genes*. Genome Research, 1999.

[12] Richard Hua, *Preprocessing for Fast Feature Detection*. University of New South Wales, 2012.

[13] John Illingworth & Josef Kittler, *A Survey of the Hough Transform*, University of Surrey, 1988. Available online:
<http://www.vision.ime.usp.br/~thsant/pool/sdarticle.pdf>.

[14] John Morrison, Eric Chown & Bill Silver, *Implementing a Real-Time Hough Transform on a Mobile Robot*. Bowdoin College, 2012.

[15] Viet Nguyen, Agostino Martinelli, Nicola Tomatis & Roland Siegwart, *A Comparison of Line Extraction Algorithms using 2D Laser Range Finder for Indoor Mobile Robotics*. International Conference on Intelligent Robots and Systems, 2005. Available online:
<http://pdf.aminer.org/000/355/979/line_segment_based_map_building_and_localization_using_d_laser.pdf>.

[16] Rahul Raguram, Jan-Michael Frahm, & Marc Pollefeys, *A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus*. European Conference on Computer Vision, 2008. Available online:
<http://en.scientificcommons.org/52072633>.

[17] RoboCup Technical Committee, *RoboCup Standard Platform League (Nao) Rule Book*. 2013. Available online:
<http://www.tzi.de/spl/pub/Website/Downloads/Rules2013.pdf>.

[18] Thomas Röfer, et al., *B-Human Team Report and Code Release 2011*. 2011. Available online:
<http://www.b-human.de/publications/>.

[19] rUNSWift Team UNSW, *rUNSWift Code Base*. The University of New South Wales, 2012.

[20] Richard Szeliski, *Computer Vision: Algorithms and Applications*. 2010. Available online:
<http://www.szeliski.org/book>.

[21] Brock White, *Automatic Calibration of the Forward Kinematics for the Nao Humanoid Robot*. The University of New South Wales, 2010. Available online:
<http://runswift.cse.unsw.edu.au/confluence/download/attachments/4981038/report.pdf?version=1&modificationDate=1290385935792>.

Chapter A

Raw Experiment Data

A.1 Experiment 1

Lines	Outliers	True Heading (°)	Mean (°)	Consistency	Standard Deviation (°)	New Time (us)	Old Time (us)
4	0	0	89	1	0	1101	3305
4	0	0	89	0.999543	0.57735	1178	3209
3	0	0	89	0.997835	1.224745	1179	3159
4	0	0	0	0.999391	0.816497	1189	3178
4	0	0	0	1	0	1201	3320
4	0	0	89	0.999543	0.57735	1178	3193
4	0	0	89	0.999543	0.57735	1159	3219
4	0	0	89	0.998782	0.816497	1166	3153
4	0	0	89	1	0	1129	3161
3	0	0	89	0.999459	0.707107	1173	3150
4	0	0	89	0.998325	1	1160	3148
5	0	60	66	0.997079	1.224745	1086	3108
4	0	60	65	0.995891	1.414214	1238	2981

3	0	60	66	0.993149	1.870829	1239	3151
5	0	60	65	0.998051	1	1122	3028
6	0	60	65	0.995337	1.612452	1084	3036
6	0	60	64	0.970087	3.872983	1077	3002
5	0	60	66	0.994552	1.802776	1090	3120
4	0	60	65	0.990635	2.104417	1260	3428
7	0	60	63	0.637911	13.693064	1275	2666
6	0	45	45	0.9682	3.974921	1179	2762
6	0	45	47	0.9913	2.144761	1181	2755
6	0	45	45	0.986722	2.569047	1168	2759
6	0	45	46	0.986518	2.607681	1159	2816
6	0	45	45	0.995605	1.48324	1170	2664
6	0	45	47	0.993714	1.843909	1158	4025
5	0	30	26	0.97615	3.349959	1358	2958
5	0	30	26	0.986656	2.472066	1354	2963
5	0	30	27	0.986783	2.494438	1362	2893
5	0	30	27	0.990858	2.054805	1301	2902
6	0	30	27	0.975186	3.361547	1811	3066
6	0	30	28	0.976248	3.301515	1807	2999

A.2 Experiment 2

Lines	Outliers	True Heading (°)	Mean (°)	Consistency	Standard Deviation (°)	New Time (us)	Old Time (us)
5	0	0	0	0.999415	0.707107	2158	7178
5	0	0	1	0.997663	1.118034	2204	4610
6	0	0	1	0.99885	0.774597	2183	4873

5	0	0	2	0.999415	0.707107	2208	4869
5	0	0	0	0.99961	0.5	2245	4840
5	0	0	1	0.998441	1.118034	2182	4751
5	0	0	0	0.997663	1.118034	2149	4602
5	0	0	0	0.999415	0.866025	2173	4692
5	0	0	89	0.999026	0.707107	2280	4597
5	0	0	0	0.999415	0.707107	2393	5505
5	0	0	0	0.99961	0.5	2124	4802
5	0	0	0	0.99961	0.5	2145	4706
5	0	60	66	0.995526	1.581139	1737	3745
5	0	60	66	0.994168	1.732051	1762	3647
5	0	60	65	0.997468	1.224745	1768	3660
5	0	60	66	0.998441	1.118034	1739	3781
5	0	60	65	0.989909	2.291288	1690	3762
5	0	60	66	0.991646	2.179449	1721	3800
5	0	60	66	0.99262	2.061553	1734	3746
5	0	60	66	0.999415	0.707107	1775	3825
5	0	60	65	0.998636	0.866025	1727	3638
5	0	60	65	0.993196	1.870829	1718	3837
5	0	60	65	0.99281	1.936492	1731	3761
6	0	60	67	0.996754	1.264911	1737	3907
7	1	30	23	0.997566	1.095445	1763	3480
7	1	30	24	0.996418	1.341641	1635	6723
6	1	30	25	0.996494	1.5	1650	3760
7	1	30	23	0.994322	1.67332	1693	3433
7	1	30	23	0.993174	1.843909	1703	3456

6	1	30	25	0.987963	2.5	1615	3693

A.3 Experiment 3

Lines	Outliers	True Heading (°)	Mean (°)	Consistency	Standard Deviation (°)	New Time (us)	Old Time (us)
3	0	0	88	0.996213	1.581139	2541	6127
3	0	0	89	0.999459	0.707107	2581	6425
3	0	0	88	0.991354	2.345208	2611	6457
3	0	0	1	0.98866	2.645751	2533	6415
3	0	0	89	0.999459	0.707107	2596	6605
3	0	0	88	0.996213	1.581139	2534	6492
3	0	0	88	0.995132	1.732051	2523	6424
3	0	0	0	0.991354	2.345208	2524	6388
4	1	0	87	0.992972	2.12132	2576	6321
3	0	0	89	0.999459	0.707107	2685	6556
3	0	0	87	0.992972	2.12132	2696	6521
4	0	0	89	0.998782	0.816497	2494	6357
3	0	0	89	1	0	2523	6448
3	0	0	0	0.999459	0.707107	2628	6339
3	0	0	88	1	0	2444	6354
3	0	0	87	0.995132	1.732051	2457	6981
5	0	45	47	0.987749	2.54951	1551	3600
5	0	45	45	0.989332	2.345208	1562	3519
5	0	45	47	0.978284	3.354102	1596	3613
5	0	45	46	0.982937	3	1547	3723
5	0	45	46	0.989896	2.291288	1535	3635

5	0	45	46	0.982537	3	1606	3620
5	0	45	46	0.985055	2.783882	1567	3684
5	0	45	45	0.990859	2.179449	1589	3846
5	0	45	47	0.97904	3.354102	1583	3493
5	0	45	46	0.987366	2.54951	1520	3649
5	0	45	47	0.975365	3.570714	1633	3551
5	0	45	47	0.981944	3.122499	1617	3579
5	0	45	45	0.98873	2.44949	1659	3702
6	1	45	42	0.94607	5.315073	1594	3643
5	0	45	43	0.968073	4.062019	1666	3643
5	0	45	43	0.977701	3.391165	1637	3821
5	0	45	45	0.99552	1.581139	1632	3688
6	0	45	45	0.982497	2.966479	1572	3729
5	0	45	45	0.980037	3.278719	1637	3695
7	2	45	43	0.978654	3.316625	1664	4811
6	1	45	44	0.985038	2.783882	1664	3682
7	2	45	44	0.97904	3.316625	1620	3773
6	1	45	45	0.991635	2.179449	1613	3704
6	1	45	45	0.990857	2.179449	1662	3669
6	1	45	43	0.966535	4.1833	1613	3666