Contents

1 Background Information

1.1 Introduction

Robocup is a competition that aims to produce a team of humanoid robots that can beat the world champion human team in soccer, by the year 2050. In an effort to fulfil this goal, universities, schools and independent entrants are encouraged to develop hardware and software to compete in a yearly tournament against each other. The tournament is comprised of many leagues, one of which is the four-legged league, where entrants have to program Sony AIBO entertainment robots to play soccer as a team. No hardware modifications are allowed in this league, and so the challenge is to develop software that is superior to the other teams.

The University of NSW has participated in the four legged league for a number of years, and has a remarkable track record. Last year, the UNSW team "rUNSWift" were world champions of the four legged league. With many changes to this world-beating code base, the new members of the 2004 rUNSWift team were able to reach the quarter finals in this year's competition held in Lisbon, Portugal.

For a robot to be able to play the game of soccer, one of the fundamental pieces of information it needs to know is where it is on the field. For robots, the process of using information from lower-level senses, such as the "vision" supplied by the video camera inside the AIBO's head, to form an estimation of the location and orientation of themselves on the field is called "localisation". Localisation not only covers the robot working out where it is on the field, but also its teammates, the opposition team members and the ball. This report covers changes made to the localisation portion of team rUNSWift's code.

Although unrelated to localisation, a small section at the end of this report is devoted to the behaviour of the goalkeeper, which has been completely redone from last year.

1.2 Related Information

The Robocup tournament is held annually, and many universities participate in its various leagues. Information on the tournament itself can be found at [1]. Arguably, one of the most action packed and popular leagues in the tournament is the four-legged league [2], which uses set hardware in the form of Sony's AIBO entertainment robot. The AIBO is a versatile robotics platform that can be reprogrammed to perform many tasks, including the playing of soccer games. For AIBO specifications, documentation and tools for programming the robot, consult the AIBO SDE webpage [3].

Although the league itself may seem limited by the fact that no hardware modifications are allowed, in actual fact the different strategies and sensor processing techniques implemented in software by the different teams have a massive impact on the quality of play, and the superior teams can thrash the weaker teams by scoring margins in the vicinity of 15 goals to 0. The soccer matches in the legged league are refereed, with rules, described in [4], to be adhered to.

The main portion of the legged-league tournament is the soccer matches, however, there are also extra "challenges" that teams can participate in. This year, there were three challenges to participate in: an open challenge, a localisation challenge, and a variable

lighting challenge, which are described in more detail in [5]. The localisation challenge, which the rUNSWift 2004 team came first in, is described in this report.

The rUNSWift 2004 team (See [6]) used the code from last year as a starting point. Many of the systems used from last year have been carried over, or extended. This is the case for the localisation system, where the basic self-localisation system has been greatly extended. For an overview of the system used by last year's team, the "Localisation" chapter of the 2003 team's report [7] should be consulted.

This report is limited to only discussing the localisation system, however there are times when the other modules of the entire rUNSWift system are referred to. The rUNSWift software architecture is divided into modules, as described in sections 1.4.1 and 1.4.2 of [11], these being vision, localisation, locomotion, behaviours and wireless control. Generally, there is a reference for the different modules of the system: consult [10] for low level vision, [9] for the high level vision system (that the localisation module is highly dependent on), [11] and [12] for the high level behaviours that utilise the localisation information, and [13] for information on locomotion and many of the utilities used by the team.

The localisation system from last year relied on a method of filtering sensor information called a "kalman filter". A good introduction to the workings of the kalman filter can be found in [14]. However, the kalman filter is by no means the only method of utilizing sensor information to form a consistent localisation estimate; another popular method of doing this is by "Monte-Carlo" methods, such as a particle filter. A description of implementing a particle filter for robot localisation is given in [16]. Although the rUNSWift team have never used a particle filter, it is useful to examine the technology that some of the other Robocup teams, such as the "German Team", have used to good effect in the past.

In extending the existing localisation system, the kalman filter was changed to handle more than one position hypothesis, with the implementation described in [15] acting as a guide for some of the mathematics. When implementing the field line localisation system, rUNSWift's [8] and the German Team's [17] implementation from last year had been studied, and some of the techniques used were based on these works.

1.3 Conventions and Terminology

For Robocup matches, all objects are colour coded: there is an orange ball that is kicked or pushed around by the Sony AIBOs (which are simply termed robots in this report) on a green field with white lines marking out regions such as the goal box, goal line, centre line and centre circle. At the corner of the field are posts with two separate colours on them, these are beacons that the robots can use to navigate, or localise by. The goals themselves are also colour coded, blue for the blue team's own goal, and yellow for the red team's own goal. Teams consist of four robots each, one goalkeeper, and three forwards. At the edge of the field is a white, low, thick wall with sides angled at 45° to stop the ball rolling off the field; this is termed the field border. Outside the border is a white wall about 30cm in height that blocks out some of the background image, which includes all objects that are not defined inside the RoboCup environment, such as spectators, tables, and any other objects inside the room that houses the RoboCup field.

Figure 1.1: A RoboCup match in action.

The low-level vision portion of the vision module is responsible for taking the raw data from the robot's video camera mounted in its head, and classifying every pixel in each frame according to its colour to produce a classification plane, or c-plane from the raw image data. Since each object on the Robocup field is colour coded, the colour classification of a pixel determines which object it is part of, eg. any objects that are classified "ball orange" are probably part of the ball. A particular pixel on the c-plane is specified by its c-plane coordinates, which have their origin in the top left, with x position increasing to the right, and y position increasing downwards. Regions on the c-plane that have the same classification are compiled into blobs.

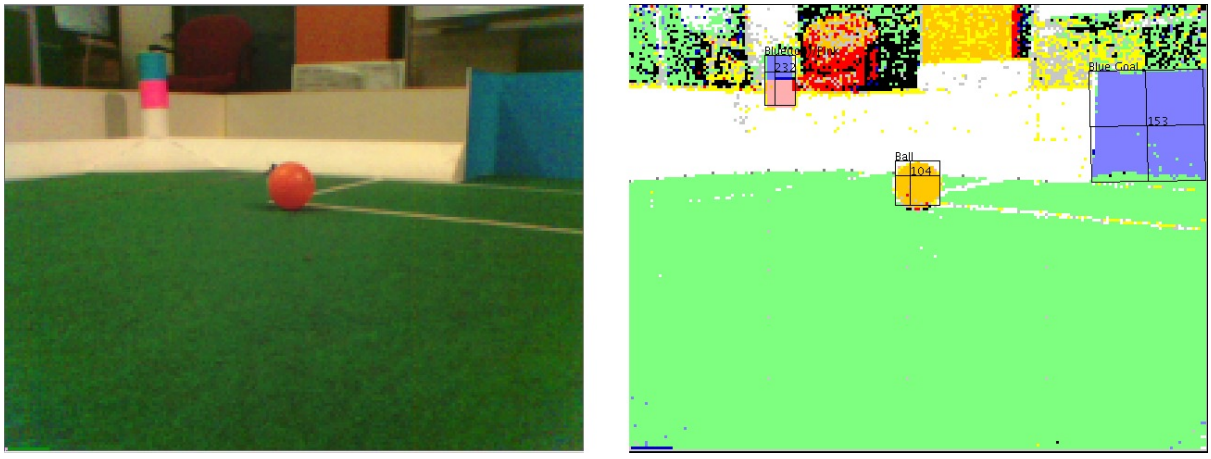

Figure 1.2: A camera frame, and its corresponding c-plane. Note the bounding boxes surrounding identified objects, such as the ball, beacon and goal.

The high-level vision in the vision module is responsible for recognizing defined RoboCup objects, such as the ball, beacons, goals and other robots, from the blobs provided by the low-level vision. Because there is noise in all c-plane data, plus an unknown background of possible colours, it cannot be concluded that every coloured blob corresponds to a specific RoboCup object, e.g. not every ball orange blob corresponds to the ball. The

high-level vision applies sanity checks to each coloured blob to ensure that any classified object is a sensible match to the physical properties of that object. Some of these sanity checks involve checking the object against the horizon, which is an estimation of where the top of the outer wall around the field would appear on the c-plane. Most RoboCup objects appear below the horizon, while above the horizon is usually just the background, which does not hold any useful information. Every identified object has a surrounding bounding box that determines the object's dimensions in pixels on the c-plane, and each object gets a distance and heading estimate in robot-relative coordinates.

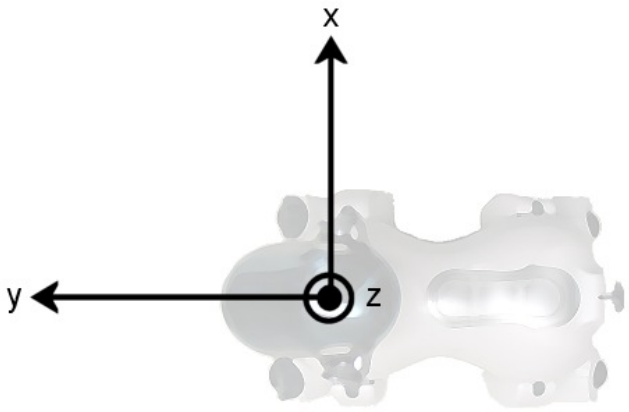

Figure 1.3: The robot-relative coordinate system

The robot-relative coordinate system, shown in figure 1.3, has its origin on the ground, under the base of the neck of the robot. The y-direction points forward in the same direction as the robot's body, the x-direction points left, and the z-direction points up from the field surface. Converting from c-plane coordinates to robot-local coordinates involves projection down to the ground plane, as explained in [18].

The localisation, or GPS module, is responsible for taking the identified objects from the vision module and using the beacons to calculate the robot's own position in global coordinates, or calculating the global coordinates of other robots or the ball if it identifies robot or ball objects. The global coordinate system, shown in figure 1.4, is dependent on which team the robot is on. The origin is defined as the goal to the left of the robot's own goal when facing up field. The x-axis points across field to the right side beacon, while the y-axis points up field. Measurements are in centimetres. For objects that have a definable orientation, such as the robot itself, a heading can also be defined. The heading is measured in degrees, and is zero when pointing in the direction of the x-axis, increasing counter-clockwise (So the y-axis is at 90°). Converting between local and global coordinate systems is a simple matter of a rotation and a translation.

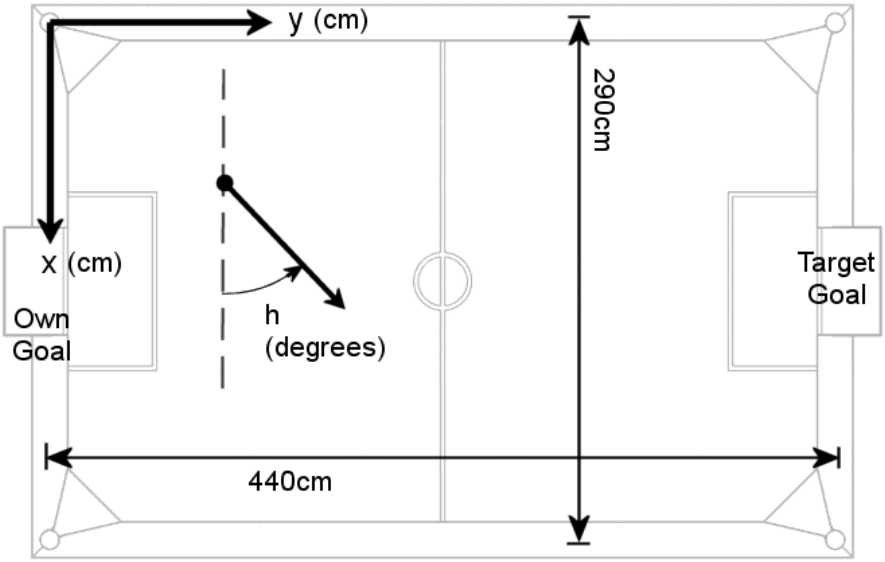

Figure 1.4: The global coordinate system.

In the diagrams in this report, a box with a line at the front will be used to indicate the physical location of a robot, while a dot with an arrow pointing from it will represent a position estimation made by the robot, optionally with an ellipse surrounding it that represents the variance of that estimation.

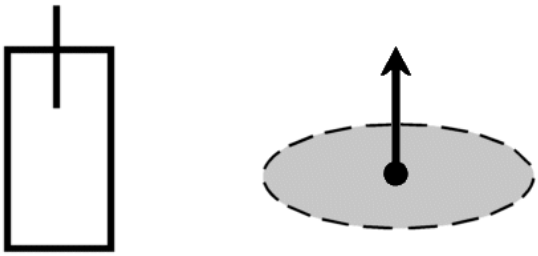

Figure 1.5: Diagrammatic representation of a robot (on the left) and position / heading estimate with variance (on the right).

2 Modifications To The Kalman Filters

The localisation or GPS module of the rUNSWift code is responsible for taking recognized objects from the vision module, and using the robot-relative distances and angles associated with each object observation to build a global model of positions of the robot itself, its teammates, opposition robots and the ball. There are several different sections to the GPS module, dealing with the different objects. Much of the rUNSWift 2003 GPS code has been carried over from last year, but a significant extension has been made to the robot's self localisation code. The sections of the module dealing with calculating the world ball model have undergone virtually no changes, while the section devoted to opposition tracking has had a small change to it. All changes made are documented in this chapter.

2.1 Multiple Gaussian Mode Probability Distribution for Self Localisation

2.1.1 Previous Localisation Method

The self localisation system used by the rUNSWift 2003 team was based on an extended kalman filter. The extended kalman filter is a computationally efficient method for calculating an estimation of a state, which in this case is the robot's field location and orientation, in the form of a Gaussian (or normal) probability distribution given observations that can have non-linear or non-Gaussian probability distributions.

The state estimation was continually updated based on robot movement, and then corrected whenever one or more of the beacons or goals surrounding the field were seen. With uniquely identifiable landmarks, and little false readings due to the high background wall, the system was successful in providing accurate and robust information of the robot's location and orientation, provided the robot regularly corrected its location by actively looking for the landmarks at time intervals.

One of the limitations of the extended kalman filter is that it assumes the state probability distribution is Gaussian. With a Gaussian distribution, there is only a single maximum, or mode, i.e. it only works on a single hypothesis at once. This was adequate for the 2003 Robocup rules, however, the 2004 rules state the two central beacons were to be removed, which meant that there would be less information to localise from. To make up for this deficiency another source of localisation information would be needed, the prime candidate being the white lines marked on the field, and the white border surrounding the field.

If the robot were to see a single line, then it cannot uniquely identify that line and hence deduce that it is in a particular field location, rather, seeing a single line constrains the robot to be in one of several locations on the field (as in figure 2.1), but there is no reason to favour one of these locations over another.

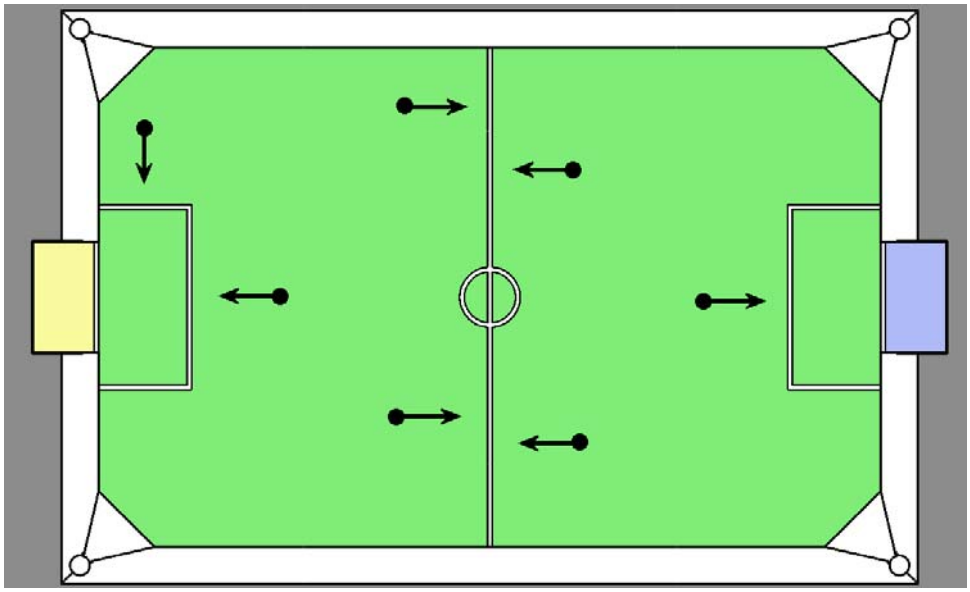

Figure 2.1: Seeing just a single straight line means the robot could be in a large number of possible positions (not all possibilities are shown).

The only solution to this is to keep track of all the possible locations simultaneously, which the existing kalman filter is unable to do.

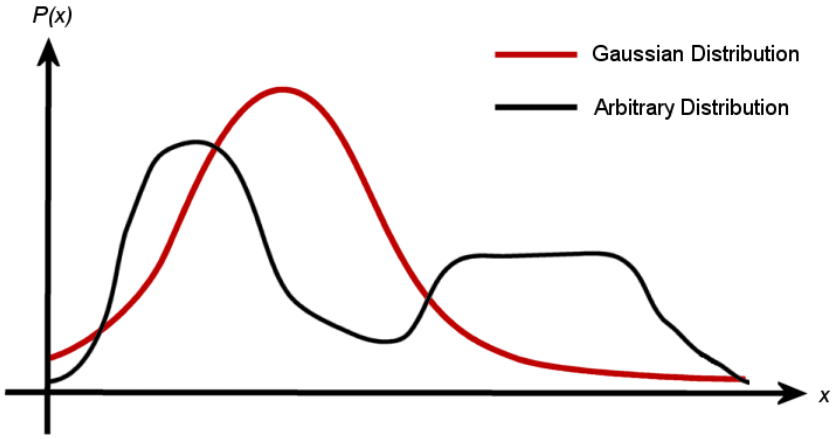

Figure 2.2: A Gaussian distribution can make a poor match to distributions with more then one local maximum.

Another limitation of the extended kalman filter is that its convergence can be affected if the measurement noise of observations has a non-Gaussian distribution. When a robot consistently views a particular beacon, and that beacon is correctly classified, then the observation noise is approximately Gaussian, and so this is not an issue. However, when a beacon is wrongly identified, or "phantom" beacons are seen (a phantom beacon is where the robot has classified a beacon where there isn't one), the filter's output is affected. The incidence of mis-classified beacons increased in 2004 due to reduction of the outer wall height, leading to increased background interference, and the decreased colour resolution of the camera in the Sony ERS-7 model compared to 2003's ERS-210 model.

2.1.2 Modified Probability Distribution for Self Location

To accommodate the non-unique observation information, a localisation system that could represent more than one hypothesis on the robot's position was necessary. A multiple hypothesis system would also help the position estimation accuracy in case of mis-classified or phantom beacons, as two separate hypotheses could be kept of the robot's location: one using the new (dramatically different) observation information, and one ignoring this information.

One option was to implement a Monte-Carlo localisation scheme, or "particle filter". In such a scheme, each particle represents a separate location & orientation hypothesis, while the relative density of the particles gives the probability of being in a certain location. Upon receiving observation information, each particle is individually evaluated to see how well it fits the given observation, and is assigned a weighting based on how well it fits. To update the distribution for robot movement, a particle is selected at random; where particle selection is weighted proportionally to the particle's probability. The selected particle is moved according to the robot's movement, and then some noise is added to the particle. This means that following the movement update, all particles are equally weighted again, until the next observation arrives.

While the particle filter has the advantage of being able to represent any form of probability distribution, it was not used for a couple of reasons. One is that the detail with which the distribution is represented increases with the number of particles used, but increasing the number of particles increases the computational load. To get an accuracy similar to that of the existing kalman filter would require significantly more processing time. The other reason was that it would require a complete re-write of the entire localisation system, which would be quite a wasteful use of the time available, especially since the existing system developed in previous years worked well in many situations. Instead, the extended kalman filter was modified to handle more than one Gaussian distribution at a time, resulting in a "Multiple-mode" kalman filter.

The existing kalman filter worked on the assumption that the state could be represented by a Gaussian distribution with a mean of $\underline{\mu}$, a 3-dimensional vector containing the x and y field coordinates and a heading angle h (this is the "global" coordinate system). The Gaussian distribution also has an associated covariance matrix $\underline{\underline{C}}$, which gives a measure of how certain the mean is in each of the 3 dimensions. Mathematically, the probability P of being in state $\underline{x}$, where the vector $\underline{x}$ is again a triple of two field coordinates and a heading, is:

$$P(\underline{x}) = \frac{1}{\sqrt{2\pi}^3 \sqrt{\det \underline{\underline{C}}}} \exp\left(-\frac{1}{2} (\underline{x} - \underline{\mu})^T \underline{\underline{C}}^{-1} (\underline{x} - \underline{\mu}) \right) \qquad \text{(Eq 2.1)}$$

where:
det specifies the determinant of the following matrix.
exp specifies the exponential of the following expression.
T superscript specifies the transpose of the previous expression.

The scalar expression in front of the exponential of equation 1.1 is the normalisation constant, which ensures that P is a valid probability distribution that integrates over all $\underline{x}$ values to 1.

The multi-modal extension to the kalman filter assumes a probability distribution to be the sum of n Gaussian distributions, each with a separate mean $\underline{\mu}_i$ and covariance matrix $\underline{\underline{C}}_i$. A weighting scalar w_i is also associated with each mode. Thus the new distribution is expressed as:

$$P(\underline{x}) = \sum_i^n \frac{w_i}{\sqrt{2\pi}^3 \sqrt{\det \underline{\underline{C}}_i}} \exp\left(-\frac{1}{2} (\underline{x} - \underline{\mu}_i)^T \underline{\underline{C}}_i^{-1} (\underline{x} - \underline{\mu}_i) \right) \qquad \text{(Eq 2.2)}$$

To ensure this expression is a valid probability distribution, the integral over all possible x values must be 1. Equation 2.1, which is contained in equation 2.2, already integrates to 1, so this constraint simplifies down to:

$$\sum_i^n w_i = 1 \qquad \text{(Eq 2.3)}$$

The new probability distribution is implemented by storing an array of Gaussians, each with a weight, mean and covariance. Both stages of the kalman filter cycle are performed on each mode individually, so the new localization system can be thought of as n kalman filters working in parallel. In the prediction stage, i.e. updating for robot movement, the weighting of each Gaussian mode remains unchanged. However, in the correction stage, i.e. updating for the robot's visual observations, the weights are scaled according to how well the observations seen match the current mode. Finally, the weights must be renormalized so that a valid probability distribution can be maintained.

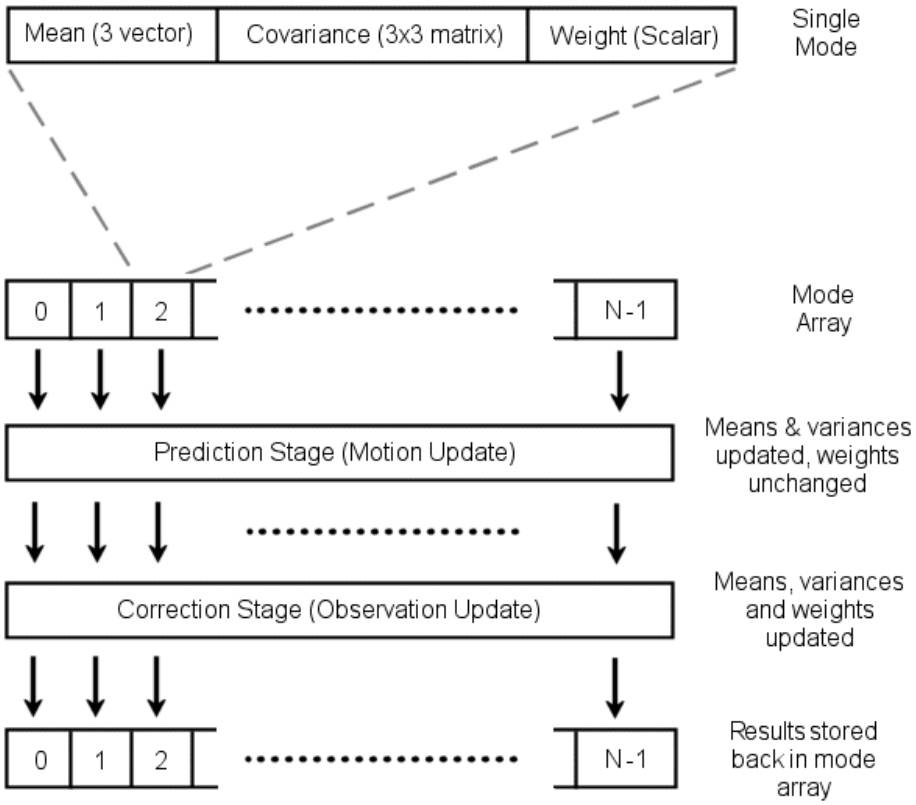

Figure 2.3: The multi-mode probability distribution is represented by an array of single modes. Kalman updates are performed on each mode independently.

The observation update and renormalization stages are now explained in more detail.

2.1.3 Weight Scaling for Observation Updates

When observation information is applied to each Gaussian mode, the weight of that mode needs to be altered to reflect how well the current state matches the given information. The modes that better match the observation data should receive greater weight than the non-matching modes since they are the more consistent modes, and so (presumably) hold more accurate information. A weight scaling function is needed that takes into account the difference between each mode's mean and what the observation implies the mean should be, as well as mode and observation variances.

A suitable scaling function was obtained by first considering a simpler one-dimensional case, and extending it to the three needed. In the one-dimensional case, we consider the current state as having a Gaussian probability distribution with scalar mean μ and variance C:

$$P(x) = \frac{1}{\sqrt{2\pi}\sqrt{C}} \exp\left(-\frac{1}{2} \frac{(x-\mu)^2}{C} \right) \qquad \text{(Eq 2.4)}$$

An observation arrives which claims the state probability has mean v and variance D. This is represented by the distribution O:

$$O(x) = \frac{1}{\sqrt{2\pi}\sqrt{D}} \exp\left(-\frac{1}{2} \frac{(x-\nu)^2}{D} \right) \qquad \text{(Eq 2.5)}$$

The probability M of both these conditions occurring simultaneously is the product of equations 2.4 and 2.5:

$$M(x) = P(\underline{x})O(\underline{x})$$

$$M(x) = \frac{S}{\sqrt{2\pi}\sqrt{B}} \exp\left(-\frac{1}{2} \frac{(x-\lambda)^2}{B} \right) \qquad \text{(Eq 2.6)}$$

where:

$$S = \frac{1}{\sqrt{2\pi}\sqrt{C+D}} \exp\left(-\frac{1}{2} \frac{(\mu-\nu)^2}{C+D} \right) \qquad \text{(Eq 2.7)}$$

$$\lambda = \frac{D\mu + C\nu}{C+D}$$

$$B = \frac{CD}{C+D}$$

The result in equation 2.6 is the product of two Gaussian distributions; one dependent on x, while the other (represented by S in equation 2.7) is independent and hence is a constant scalar for any given state and observation. Integrating equation 2.6 over all x gives a result of S, indicating that the expression given in equation 2.7 could be a suitable scaling factor for a 1 dimensional update. An examination of equation 2.7 shows that S decreases as the distance

between the means of P and O increase, while increasing the variances will generally increase S, as long as the variances don't grow too large.

A logical 3 dimensional extension to the scale function given by equation 2.7 would be:

$$S = \frac{1}{\sqrt{\det(\underline{\underline{C}} + \underline{\underline{D}})}} \exp\left(-\frac{1}{2} (\underline{\mu} - \underline{\nu})^T (\underline{\underline{C}} + \underline{\underline{D}})^{-1} (\underline{\mu} - \underline{\nu}) \right) \qquad \text{(Eq 2.8)}$$

For 3D vector means $\underline{\mu}$ and $\underline{\nu}$, and 3x3 covariance matrices $\underline{\underline{C}}$ and $\underline{\underline{D}}$. Note that in this expression the constant scalar involving 2π has been removed, since it would be applied to all mode weights, and so would effectively be removed anyway when the distribution renormalisation is performed. Also note equation 2.8's similarity to the "likelihood equation" (equation 15) in [15].

For use in the kalman filter, the state mean has to be converted into observation space by applying the non-linear function h to each Gaussian's mean $\underline{\mu}_i$, while an approximation to the state covariance in observation space can be made by multiplying the mode's covariance $\underline{\underline{C}}_i$ by the Jacobian matrix $\underline{\underline{H}}$ of function h, evaluated at the mean $\underline{\mu}_i$. The final scaling function also did not use the inverse determinant term shown at the front of equation 2.8, as testing revealed that this term tended to give too small weightings to large variance modes. Finally, the weight scaler S for an observation $\underline{z}$ with covariance $\underline{\underline{R}}$, applied to a single Gaussian mode with mean $\underline{\mu}_i$ and covariance $\underline{\underline{C}}_i$, is defined by:

$$S = \exp\left(\frac{1}{2} (h(\underline{\mu}_i) - \underline{z})^T E^{-1} (h(\underline{\mu}_i) - \underline{z}) \right) \qquad \text{(Eq 2.9)}$$

where:

$$\underline{\underline{E}} = \underline{\underline{R}} + \underline{\underline{H}}^T \underline{\underline{C}}_i \underline{\underline{H}} \qquad \text{(Eq 2.10)}$$

Using the new scaling function given by equation 2.9, the amended kalman observation update system is shown in figure 2.4.

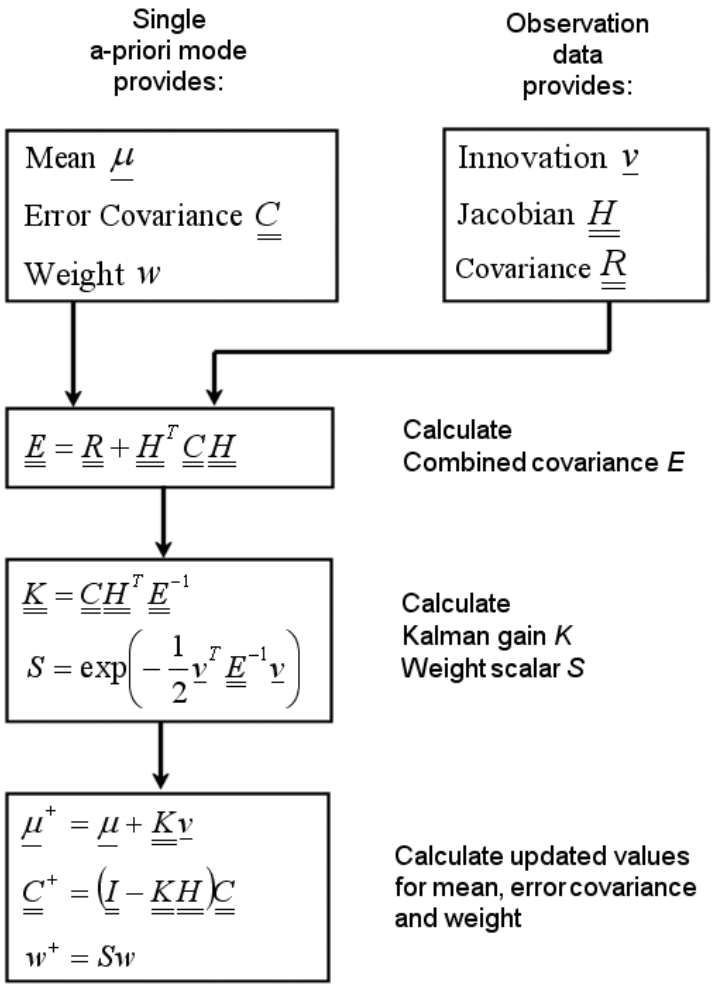

Figure 2.4: Kalman observation update for a single mode.

2.1.4 Distribution Renormalisation

Because of the weighting constraint given by equation 2.3, it is necessary to renormalise the weights after every observation update, where individual mode weights are scaled differently. This is a relatively simple process where the total weight of all Gaussian modes is calculated, and then each mode's weight is divided by this total. Following this procedure, the total weight will be 1, and the sum of Gaussians can be regarded as a valid probability distribution.

After the normalisation is complete, similar Gaussian modes are merged together. The merging is performed to ensure that lower probability modes with a significantly different hypothesis from the highest probability mode don't get "starved out" of the fixed size Gaussian mode array. Consider the following example shown in figure 2.5.

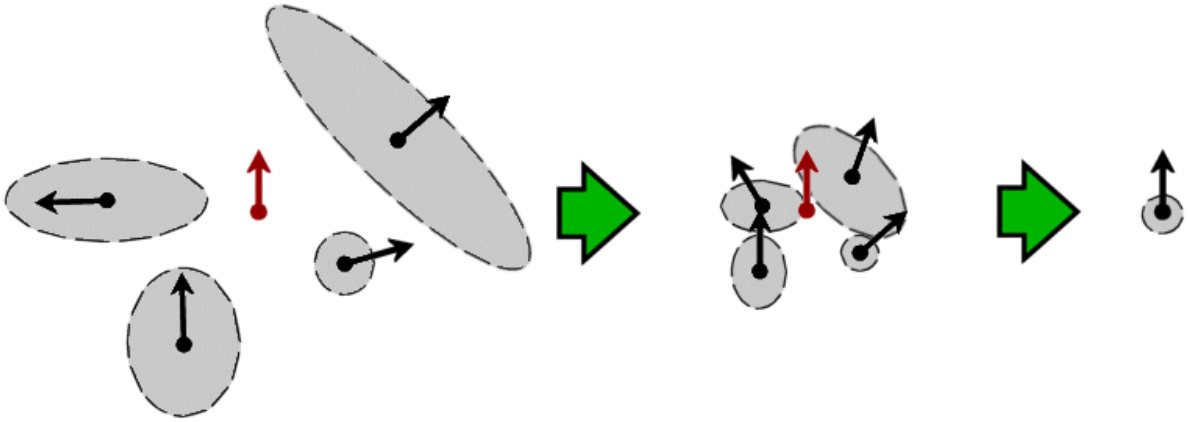

Figure 2.5: The central directed point represents the applied update. The other four directed points represent Gaussian modes with covariances represented by their surrounding ellipses. If the same update is applied consistently, the four different modes will converge into one. Note the covariances will get progressively more circular, due to applying a correction update, but then growing the variance equally in all directions during the motion update.

If the mode array is limited to storing 4 modes, and is initialised to have 4 random modes of equal weight, and then an observation that has sufficient dimension to specify a single field point & heading is repeatedly applied, all modes will eventually converge to that point and heading. Some modes may take longer than others to converge, and they may result in significantly different weights, however, after some time all 4 modes will have means in (almost) the same spot. At this time, the descriptive power of the mode array is being wasted, as there is no more room for other modes in the array, whereas if all 4 modes were merged into 1, it would be possible to store another 3 modes with different means and variances.

The merging process works by comparing every pair of modes and applying two distance metrics to them. If both distances between the two modes are less than certain thresholds, then the two modes are merged by creating a new mode with the weighted average of their means and covariance matrices, and their total weight. The two merged modes are then removed. The two distance metrics used were:

1. The sum of squares of elements in the difference between the two means, and then
2. The sum of squares of elements in the difference between the two covariance matrices.

If the first metric resulted in a distance greater than the set threshold, then the second metric is not applied.

After all similar modes have been merged together, the mode array is sorted according to weight. In most cases the weight order will not have changed much, so an insertion sort is used since it is simple and efficient in this case.

2.1.5 Global Maximum of the New Probability Distribution

In theory, the best estimate of the robot's position and heading would come from the global maximum of the probability distribution in equation 2.2. To find the global maximum, the probability of each local maximum in the distribution would have to be evaluated, and the highest probability maximum recorded. Finding the local maximums of equation 2.2 is not

easy, as can be seen upon examination of the gradient of P, shown in equation 2.11. One condition for a local maximum is equation 2.12, i.e. the gradient equals the zero vector.

$$\underline{\nabla} P(\underline{x}) = \sum_i^n \frac{w_i}{\sqrt{2\pi}^3 \sqrt{\det \underline{\underline{C}}_i}} \exp\left(-\frac{1}{2} (\underline{x} - \underline{\mu}_i)^T \underline{\underline{C}}_i^{-1} (\underline{x} - \underline{\mu}_i) \right) \underline{\underline{C}}_i^{-1} (\underline{\mu}_i - \underline{x}) \qquad \text{(Eq 2.11)}$$

$$\underline{\nabla} P(\underline{x}) = \underline{0} \qquad \text{(Eq 2.12)}$$

Solving equation 2.12 analytically would normally be an impossible task, unless an approximation is made. We can apply the approximation that close to each Gaussian's mean, the only significant contributor to the gradient is the Gaussian that we are close to. This simplifies equation 2.11 down to:

$$\underline{\nabla} P(\underline{x}) = \frac{w_i}{\sqrt{2\pi}^3 \sqrt{\det \underline{\underline{C}}_i}} \underline{\underline{C}}_i^{-1} (\underline{\mu}_i - \underline{x}) \qquad \text{(Eq 2.13)}$$

when: $\underline{x} \to \underline{\mu}_i$

With the new approximation, solving equation 2.12 is a simple task, the solution being that the local maximums are located at the means of each of the Gaussian modes, which is an intuitive result. After calculating the probability values at each local maximum, the mean with the highest probability would be selected. In most cases, the mean selected is from the Gaussian mode with the highest weight. We can therefore simplify the task of finding the global maximum further by merely using the mean of the highest weight Gaussian as our position estimate. This is the method implemented in the 2004 rUNSWift code, as it was decided that the other more complex methods would not result in a useful increase in accuracy for the position / heading estimate of the robot. A comparison of the different results from each method can be seen in the 1 dimensional example below, in figure 2.6. This is an atypical case where the Gaussian mean with the highest probability is not the mean of the highest weight mode.

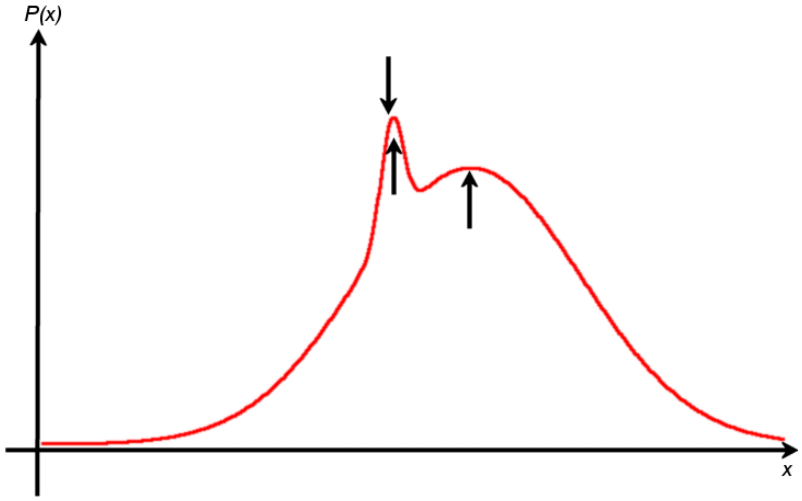

Figure 2.6: The sum of a high weight, large variance mode and a low weight, small variance mode. Also shown is the distribution maximums calculated by the three different methods.

2.2 Filtering Updates of the Kalman Filter Used for Self Localisation

2.2.1 Reasons for and Measurement of Noisy Updates

A number of circumstances have changed in the interval from the 2003 competition to the 2004 competition, which have worked to increase the amount of noisy or wrong observation readings being input into the kalman filter. These include:

- Lowering of the white outer walls to 30cm in the 2004 Robocup rules. In 2003, the four beacon landmarks at the corners of the field had a clean white background from medium to far distances. At close distances, the beacons may not have had the white background, but at that range they where large enough to be easily recognizable. In 2004, the beacons are taller than the outer walls by almost 10cm, meaning that wherever the robot is on the field, some part of any beacon it looks at will have an unknown background. The unknown background could include colours that are used in the other beacons, which can result in incorrectly identifying a beacon, or it could be the same colour as the beacon, affecting the distance / heading measurement. Work done on the sanity checks in the high level vision of rUNSWift 2004's robots [9] has helped to substantially reduce the incidence of this occurring, however it cannot stop all bad readings.

- New model Sony AIBO ERS-7 having a reflective, white finish. In 2003, the older model Sony AIBO ERS-210 robot was used, which in the majority of cases had a grey, slightly matt finish that, together with the more angular surfaces, removed specular reflections and reduced diffuse reflections as well. Due to the new model's reflectivity and its more curved shape, the surface of a robot seen by another robot will not be classified as completely white near the interface with the uniform or a joint. This is especially problematic when seeing a robot with a red uniform on, since at the edges of the uniform, the colours are classified as many small patches of yellow and pink, which the robot can mistake for a yellow and pink beacon.

- Increased noise / poor colour resolution and chromatic distortion from the ERS-7's camera. Extra noise around the pixels that make up the beacons on the robot's c-plane can lead to more noise in the distance and heading estimates for those beacons, especially at long distances where one pixel's difference can mean a change in distance of half a metre. As a result, the kalman filter will converge to its best estimate slower. This is partly offset by the higher resolution and frame rate of the new camera. While the chromatic distortion around the edges of the camera image had been removed by the rUNSWift 2004 team [10], there is still less colour resolution available near the edges of the image. While this doesn't increase the noise significantly, it can mean that the robot will fail to see a beacon where there is one, which increases the robot's susceptibility to noise by decreasing the amount of useful information input into the kalman filter.

In order to filter out noisy updates, a measurement of how unlikely an update is, given the current state, is needed. Fortunately, we already have a measurement function for this task in the form of equation 1.9. Given a mode's mean and covariance, and an observation's measurement and covariance, this equation will return a value in the range (0, 1], where lower measurements indicate less likely matches. If either the mode's or observation's covariance is large, then the equation will tend to give higher results, while if both covariances are small, the difference between the two means will have a large affect on the result, with large differences receiving very low likelihood.

With the noise measurement equation, the robot can classify the observations that receive a low likelihood factor as noise, and take appropriate precautions. It should be noted that if the robot has no idea where it is, i.e. the highly weighted modes have large variance, then effective noise filtering is not possible, since all observation will receive a larger likelihood value. This is appropriate, since if the robot doesn't know where it is, then it wouldn't be able to tell if an observation is unlikely or not.

2.2.2 Basic Filtering for Beacons and Goals

Before applying the dynamic noise filtering, it is appropriate to use some static filters on all observations beforehand, so that using measurements from situations that are known to produce inaccurate readings can be avoided. There are six fixed landmarks located around the field: four beacons at each corner of the field, and two goals opposite each other in the centres of the two shorter sides of the field. For each landmark the robot recognizes in any one camera frame, two pieces of information are provided: a distance estimate and its associated variance, and a heading estimate and its associated variance. Each piece of information provides an extra dimension to the observation update, so for example, seeing two beacons in one camera frame can provide up to four dimensions in the observation.

Whenever a landmark is recognized, in most cases at least the angle to it will be used. The distance estimate, however, can get quite inaccurate at longer ranges, due to the way the estimate is determined. In most cases, the distance is calculated by measuring the number of pixels along one of the sides or between the centroids of the two different coloured blobs that make up the object. The distance is then proportional to the inverse of that dimension measurement. The trouble is that the dimension measurement can only have an integer result, so that when the dimension measurement is small, a change of one pixel in the dimension will result in a large change in the distance. Also, because of the inverse relationship, a decrease in dimension of one pixel will generally result in a change of distance that is larger than the change of distance if the dimension was increased by one pixel (See figure 2.7). This effect is also more noticeable at larger distances.

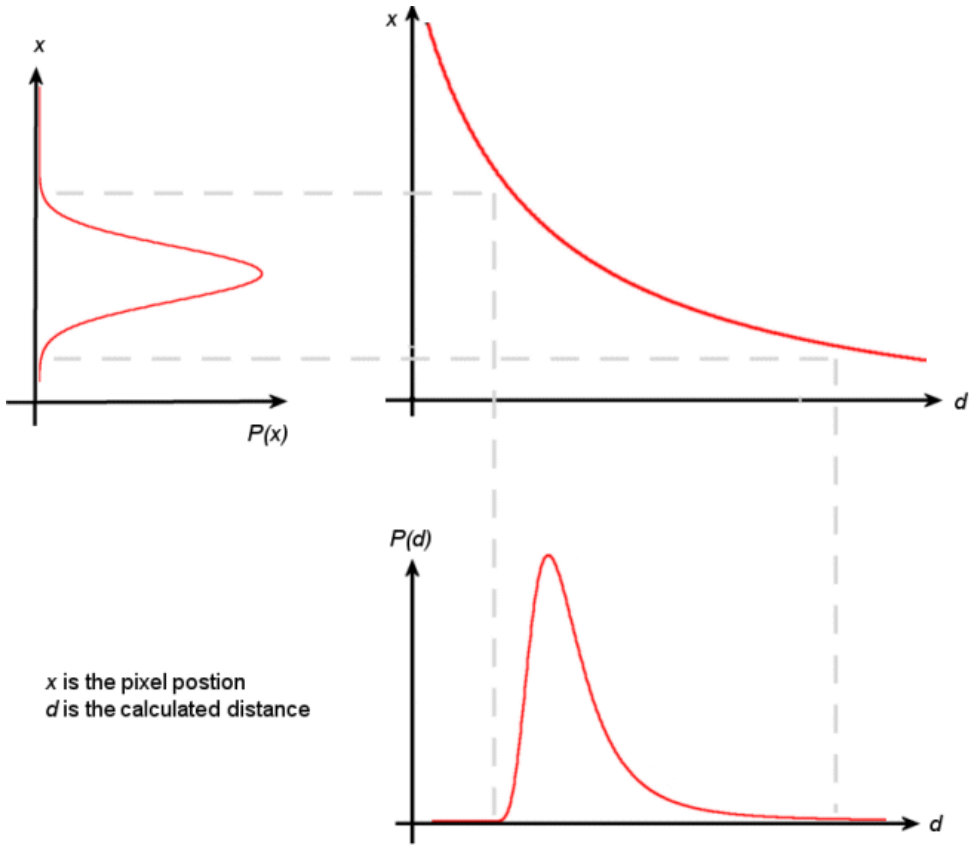

Figure 2.7: The inverse size-distance relationship, and the non-Gaussian measurements it produces. Although the dimension in pixels of an object seen may exhibit Gaussian variance, the resultant distance does not.

The result of this is that the distribution of distance measurements for long range objects is large in variance, but also non-Gaussian, tending to overestimate results. For this reason, the distance estimate is used only when the object is reported to be within a certain range, and within a certain variance limit. The allowed range for the goals is smaller (about 2.5m) than the allowed range for the beacons (about 4.5m), even though the goal is larger and should theoretically produce more accurate distance measurements. This is partly due to the relatively undeveloped method for estimating goal distance, but also because of the goal's irregular shape, where distance estimates can change depending on the angle it is viewed at and which side is being viewed.

There are also situations where both the landmark's distance and variance are not used, i.e. the landmark is disregarded completely. This only occurs for the goals at close range (< 70cm), where the goal will occupy a very large portion the robot's camera frame or the edges of the goal will be off the edge of the frame, and so accurate distance information become nearly impossible to estimate. The distance estimate for goals given to the kalman filter by the lower level vision system starts getting less accurate within a range of about 70cm, and within 50cm range the estimate is clamped to 50cm. The check is implemented in a simple manner; if the estimated distance is less than 70cm, the goal is not used for localizing at all. This check is very important for when the robot is inside the goal, e.g. for the goalkeeper, looking out towards the field, but still has some of the goal on the edge of its camera frame. In this case, the reported goal distance is 50cm, so the goal is skipped. If the goal was not skipped, then the estimated heading to it would be approximately the camera's

horizontal field of view divided by two (which is about 28 degrees). This would make the robot think it is facing in towards the centre of the goal, rather than out towards the field, which is definitely not helpful (See figure 2.8).

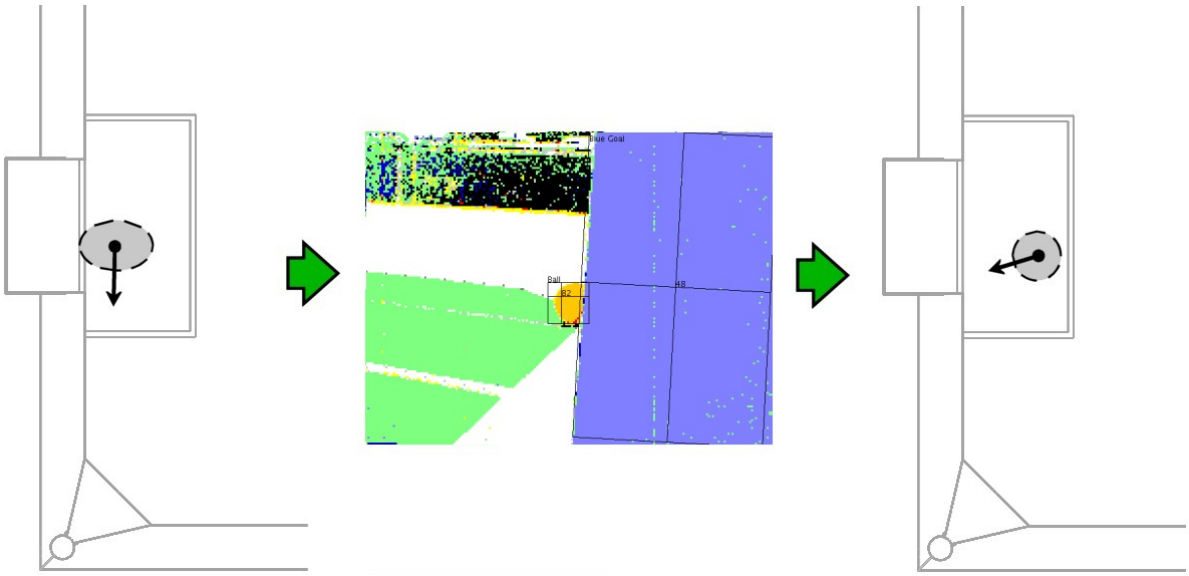

Figure 2.8: The robot begins with a correct estimation of its position, but then sees only the goal to the side of the c-plane, which is approximately in front of the robot. This throws the position estimation off by making it turn towards the inside of the goal.

As extra protection against this circumstance, and also to prevent localisation off goals when the direction to the goal is almost parallel to the goal entrance, a final check is included: the current robot position estimate must be greater than 15% of the field length away from the side of the field that the goal is in (See figure 2.9).

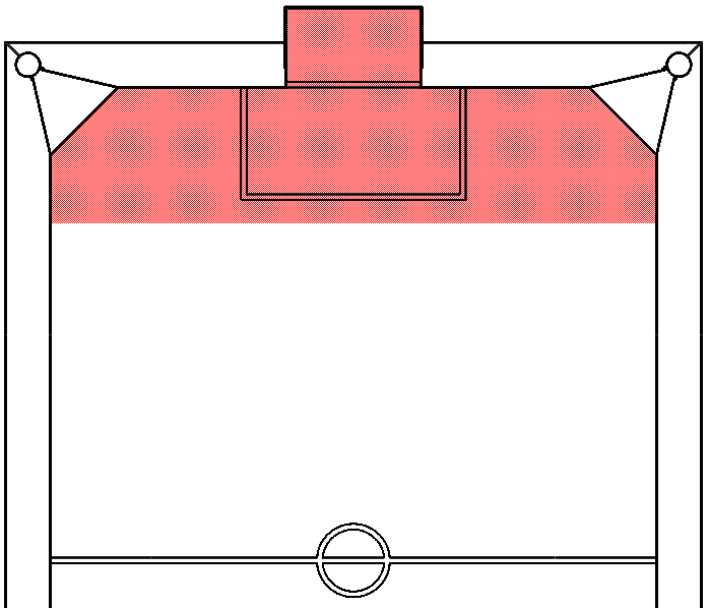

Figure 2.9: If the robot is within 15% of the field length from a goal, it will not localise off that goal. This region is shown above.

2.2.3 Filtering Unlikely Observation Updates

Once an observation has passed the static filters, its likelihood with respect to each Gaussian mode can be assessed using equation 1.9. If the likelihood value is below a certain threshold value, then it is classified as noise. A number of different responses to noise observations were implemented and tested, and these will now be explained.

The first and most simplistic approach was to throw the observation data out if the likelihood of it matching with the current mode was below a threshold, while also increasing the mode's variance. This was found to be effective when the robot was only localising off the beacons and goals. However, when there was another mechanism for decreasing each mode's variance, e.g. when localisation off field lines was implemented, this method tended to fail; leaving the robot mislocalised even after it had made several observations of the beacons around the field. This was caused by the fact that the two different localisation methods, i.e. beacon / goal localisation and field line localisation, were applied as two separate updates; so when the mislocalised robot saw beacons that weren't likely considering its current location estimate, it would throw that observation data out and increase the mode variance, while the field line localisation system would find a nearby match for the lines seen and update the mode, decreasing its variance. Thus the two localisation systems would conflict with each other, leaving the robot's position estimate static.

Other disadvantages of this method were that it did not make effective use of the new multi-modal kalman filter, and it would waste observation data that may be useful in a future time.

The second method involved splitting each mode into two whenever an observation is applied. One of the new modes would have the observation data applied to it in a kalman update, as normal. The other mode would not be updated with the new data, but its weight would be multiplied by a scaling factor given by an approximation of the probability of receiving noise information (the value used for this scalar was 10^{-2}), called the "noise scalar". This method is equivalent to considering each observation as the sum of two separate terms: the first being the actual observation measurement, a Gaussian distribution, and the second being a uniform distribution with a probability equal to that of receiving a noisy estimate.

Although this method makes better use of the new multi-modal implementation of the kalman filter and it doesn't explicitly discard any observation information, it still was slow to re-orientate after the robot became mislocalised, often staying mislocalised for tens of seconds before it found a better estimate.

The final method implemented was an extension of the second method above. As well as splitting each mode before applying any observation update, an extra mode is added to the state distribution before the updates are performed. The mode added is a very low weight, very large variance mode that is centred on the field, i.e. it represents the robot having no confidence in its current position information.

Testing during games has shown that including this large variance mode in the distribution significantly reduces the amount of time taken for the robot to become re-orientated after getting lost, or being transported to a completely different field position.

2.2.4 Additions to the Prediction Update

The prediction stage of the kalman filter, where the robot's position estimate is updated according to the robots motion, has been slightly modified from last year's code. There are two additions: one is an attempt to limit the inaccuracy caused by the robot getting stuck on a wall or in a scrum with another robot, the other is to improve the accuracy of the odometry information passed to the prediction update stage when different gain settings are used on the robot's actuators.

As shown in section 3.2.6 of [7], the prediction update for the kalman filter has an input of three scalar values, representing the amounts the robot has been told to move forward, left, and turn. These values are obtained almost straight from the high level controlling code of the robot, i.e. the "behaviour" code, the only modification being clipping the values to within the robot's movement capabilities, and scaling from odometry calibrations. Hence if the behaviour code tells the robot to move forward, the robot's position estimate will move forward, regardless of whether the robot is blocked by an obstacle or not.

A simple mechanism to detect if the robot is stuck on an obstacle, based on measuring the duty cycle of the leg effectors in the robot, has been implemented by the rUNSWift 2004 team (Section 5.6.2 of [11]). This stuck detection code returns a scalar quantity, where increasing values indicate the robot's motors are working harder, and therefore the robot's limbs are being restricted by some form of impediment, usually another robot, but possibly also the barrier wall on the outside of the field. The new motion update for the kalman filter takes this extra scalar as an input, and if it is above a threshold value, then the update quantities for forward, left and turn are scaled to smaller values, and each mode's variance is increased faster than normal. The amount which the update quantities are scaled and the rate at which the variance increases (above normal) is determined by applying two different empirical equations to the stuck quantity.

Testing of the motion update in situations where the robot was stuck revealed that position estimate is improved some of the time, however, the stuck quantity input into the motion update is still quite erratic and unreliable, and so the new motion update is not always effective.

	Distance Travelled (cm)	Standard Deviation (cm)
Stuck detection	75.1274375	45.95935976
No stuck detection	97.0143375	20.51965826

Figure 2.10: The results of a motion update with and without the stuck detection, when the robot is being held in place while it tries to walk forward. The motion update with stuck detection gives a better result, since it has a smaller distance travelled, and a larger standard deviation. When the robot was free to walk, the results with and without stuck detection are similar.

The second change to the motion update of the robot's position estimate is not a change to the prediction stage of the kalman update function, but rather to the inputs into that function. The ERS-7 model robot has a sensor that checks the current being drawn from the battery. The operating system running the robot's code, called OPEN-R, integrates the battery current sensor's recent readings and checks to see if these exceed a certain value, which would indicate that a motor has been working too hard (due to being jammed or obstructed).

If the threshold is exceeded, the robot is shut down to prevent any further damage to the hardware. This damage prevention mechanism tended to cause shutdowns of the robots during play of games, which obviously had disastrous consequences for the team.

To prevent the shutdowns from occurring, the rUNSWift 2004 team implemented a "dynamic gain" system (Section 5.7 of [11]), which would continually monitor the battery current integration value, and if it approached the shutdown threshold the motors would be switched from their normal high gain mode to low gain mode. The current used by the motors in any of the joints in the robot is determined by the difference between the joint sensor's reading of its current position and the desired position for that joint, multiplied by the joint gain. Switching to low gain values for all leg joints decreased the current used by the motors, which in turn made the battery current integration gradually decrease, reducing the risk of shutdown.

While dynamic gain helped solve the problem of shutting down robots, it did cause another problem in the accuracy of the self localisation: giving the robot the same movement command would result in different movement speeds on the two different gain settings. This was especially evident when the robot was ordered to turn at a certain speed; the resulting physical turn rate of the robot at low gain was about half that of the turning speed at high gain. So when the robot was on a low gain setting, its position estimate would turn faster than the robot was actually turning. To counter this, two separate odometry calibrations for the robot were made, one for each gain setting. The robot would then select the correct odometry values to use based on which gain setting it was on. This was an improvement over the previous system, which always used the odometry calibration for the high gain settings.

2.3 Update to Opponent Tracking System

2.3.1 Previous System

The method of opponent tracking used by the rUNSWift team in 2003 utilised the information form of the kalman filter (See section 3.5 of [7]), combined with an observation matching function that was a hybrid of two simple methods: updating only the most probable opponent, and updating all opponents with a weighting according to how probable that opponent is.

While the information form of the kalman filter was effective for distributing information among all robots on the team, the observation matching method did have some problems. Observation matching refers to deciding which opponent a visual observation of a robot on the other team should apply to, taking into consideration that all robots on the other team are indistinguishable from one another. The 2003 observation matching method applied the observation update to all robots, but with a varying weight determined by how closely the observation matched the current opponent state (Equation 3.130 of [7]). One of the problems with updating every opponent position given any opposition sighting is that continual viewing of just a single robot will make all four opponent position estimations converge to the same position.

Also problematic is the use of the opposition distance estimation when deciding which opposition filter to update. Currently the visually determined distance estimate for robots seen can fluctuate massively, while the angular measurement is reasonably stable and accurate. Use of the distance estimate can lead to updates on the wrong opposition team member.

2.3.2 Limitations of the Current Visual Robot Detection

For a detailed description of the visual robot detection system used by the rUNSWift 2004 team, consult [9]. The system used, although modified from last year's code, has some limitations that have to be taken into account when matching visual robot observation information to a particular opposition position estimate. The cause of these limitations is due to the appearance of the new model Sony AIBO ERS-7 model.

In 2003, the latest AIBO model was the ERS-210 which in most cases had a dark grey plastic finish for most parts of its body, making it distinct from the other colours used on the Robocup playing field. This made it easier to decide if the separate blobs of colour that make up each robot's uniform were all part of the same robot or not, as the coloured blobs of red and blue could (in many cases) be joined by the grey of the plastic on the robot. With the new model ERS-7, the majority of plastic on the robot is white, making it the same colour as the lines marked on the field, the outer border of the field, and the barrier walls surrounding the field.

As a consequence, seeing the multiple patches that make up a single robot's uniform may result in two or more distinct robots being reported, since the different patches may not be recognized as part of the same robot (See figure 2.11). Since each of these 'sub-robots' will be smaller and possibly higher in a frame than the actual robot, the distance estimates to the each of the observations can be significantly larger than their actual distance, because the distance estimation techniques are based on bounding box size and point projection.

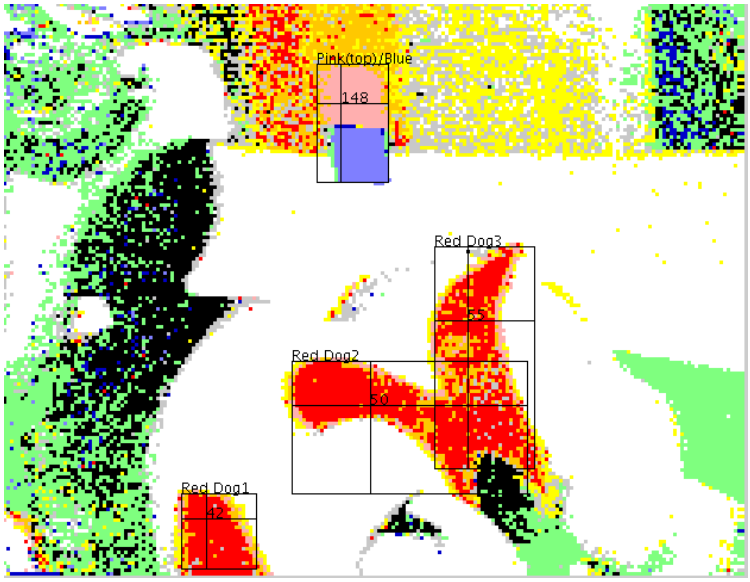

Figure 2.11: A single robot is mistaken for several different robots

It can also work the other way, with uniform patches from two separate robots being mistaken for a single robot. This is a common occurrence since it can happen anytime one robot is partially occluding another; an ongoing problem which is particularly difficult to solve and was present in all previous years' teams. With the new model robots, it is also possible that two separate (non – occluding) robots could blend into the white background and be mistaken for one. The issue hasn't been addressed by the 2004 team, mainly because it is too hard to fix, and the payoff for fixing it would be small, as neither the distance nor angle estimate is affected much when the robots are combined.

In all cases, the angular estimate is affected, however usually not by a large amount ($< 10^{\circ}$). This makes the angle estimate a much more reliable measurement than the distance.

2.3.3 Revised Observation Matching

A different algorithm for matching visual robot observations to opposition position estimates was devised, with the intention of avoiding the problems of the previous system, while taking into account the limitations of the current visual robot detection system. The new algorithm, shown in figure 2.12, is no more complex than the one used in previous years.

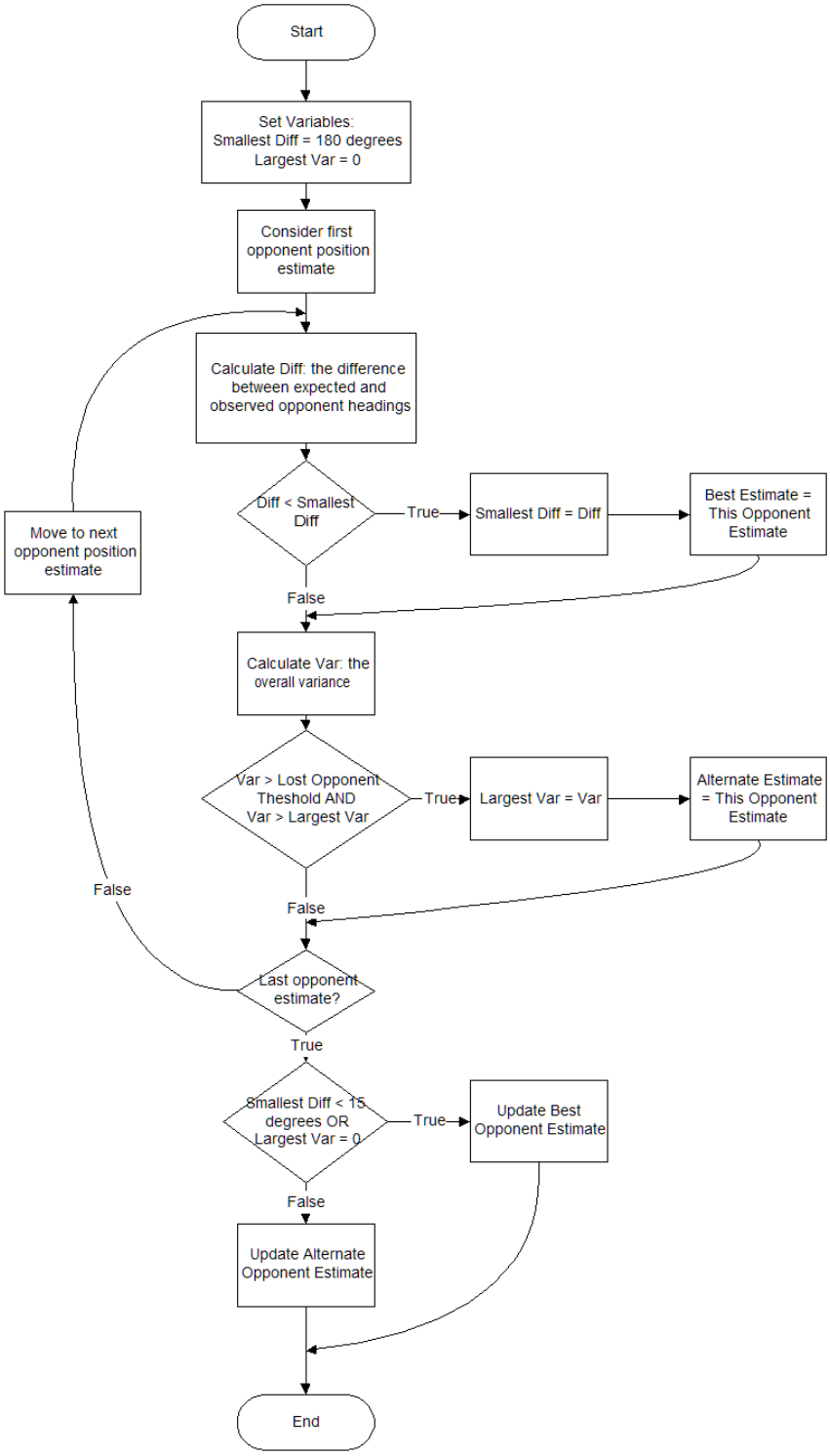

Figure 2.12: Robot matching algorithm flowchart.

The new system no longer applies an update to all opposition position estimates for every observation received, i.e. only one opposition position is updated for each observation, and the method for deciding which opposition member is updated is now only based on the observation angle estimate, even though the distance estimate is still used in the actual update. Calculating the difference between the expected and observation angles $\Delta\theta$ (shown in figure 2.13) is done using the formula:

$$\Delta\theta = \left| \text{norm}\left(\tan^{-1}\left(\frac{o_y - s_y}{o_x - s_x} \right) - s_\theta - \theta_{obs} \right) \right| \qquad \text{(Eq 2.14)}$$

where: o_x, o_y is the current position estimate of an opposition player
s_x, s_y, s_θ is the current estimate of robot's own position, heading
θ_{obs} is the heading estimate from the visual observation
norm is a function that normalizes an angle to the range $(-\pi, \pi]$

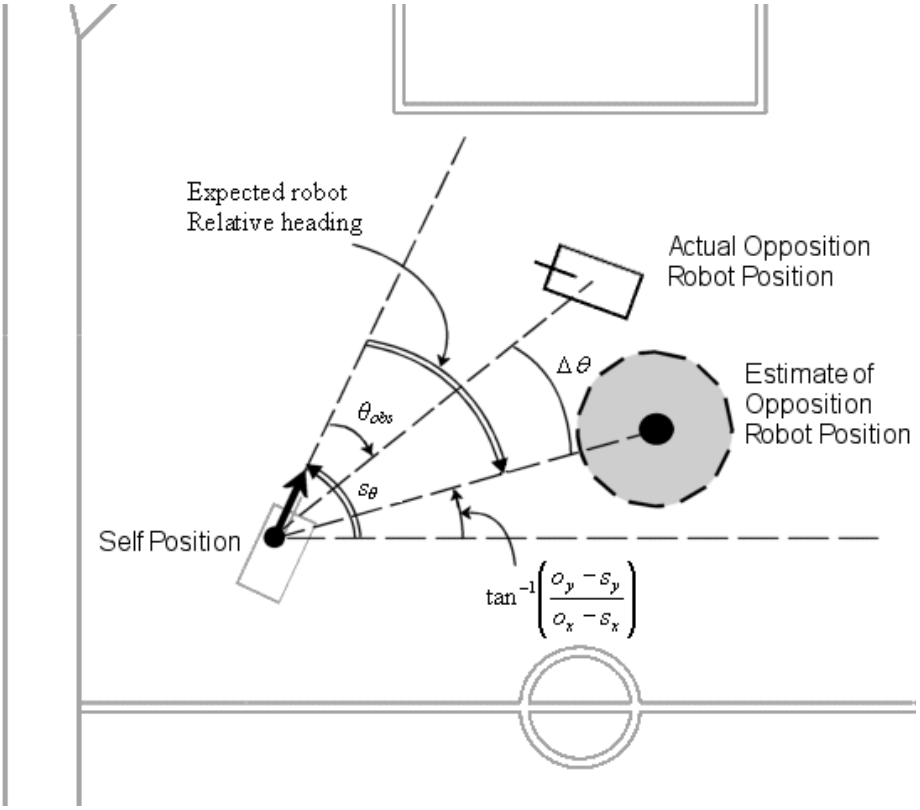

Figure 2.13: Angles involved in calculating Δθ.

The overall variance $C_{overall}$ of the opposition position estimate is found using the formula:

$$C_{overall} = \sqrt{C_{0,0}^2 + C_{1,1}^2} \qquad \text{(Eq 2.15)}$$

where: $C_{0,0}$ and $C_{1,1}$ are the diagonal elements of the opposition member's covariance matrix, i.e. the inverse of the information matrix.

The idea behind the algorithm is quite simple; apply the update to the opponent that best matches the observation angle, or if none match well, apply it to an opponent that hasn't been updated in a while. While the method isn't very precise, e.g. It doesn't take into account opponent variance in considering the angular match, and the estimator for the overall variance is not strictly correct, it does work. It is difficult to determine whether performance is better than the 2003 method or not, since the major precision bottleneck is not the observation matching method but the visual robot detection. Testing shows that opponent position tracking accuracy is similar to the old algorithm.

3 Field Line Localisation

The Robocup 2004 competition rules differ from the previous year's rules in a number of areas, one of which is the removal of the two central beacons. The removal of these beacons mean that every robot has a lower probability of seeing a fixed landmark it can localise off, reducing the total localisation information available to the robot, and consequently reducing its localisation accuracy. To make up for the information deficiency, another source of information must be used, and the field line markings and edges are a good candidate. The following sections describe in detail the field line localisation system used by the rUNSWift 2004 team.

The idea of using field lines for localisation is by no means new to the team. In the past, some attempts to utilise this information have been made (See [8]), with varying levels of success, however these have been unreliable, computationally expensive, or only applicable to a limited number of situations. The method explained here is still computationally expensive, however the accuracy obtained for that expense is usually good, and it can be used in almost all game situations. It is loosely based on the "NightOwl" system described in [8], but the modifications are substantial.

Like NightOwl, the system detects points of interest, such as field lines and borders, on the c-plane from each frame of video. It then tries to find a field position and heading for the robot that would maximise the probability of seeing the points of interest in the configuration detected. It does this by projecting the video frame (c-plane) points onto the horizontal field plane, into a robot-relative coordinate space. It then transforms the local coordinates to certain global positions, and checks how well the interest points match the actual field lines in these global positions. The kalman filter is then updated with the global position where the best match occurs.

3.1 Visual Field Line and Border Detection

3.1.1 Locating Line and Border Points

The first stage of field line localisation is to detect the points of interest on the c-plane. There are three different types of field surface that can supply localisation information:

1) White lines marked on the field. Each line is required by the rules to be 25mm in width, and are completely flat on the surface of the field.
2) The beginning of the white borders that surround the field. The borders are not flat, they have a height of 10cm, and are significantly wider than the field lines also. Consequently they appear much wider than the lines in the c-plane.
3) Large areas of green. These are the "deep field points" (or just field points), corresponding to sections of field that are not near any white markings.

Examples of these areas are shown in figure 3.1. The goal is to locate these areas on the c-plane, and remember their location by occasionally storing their screen coordinates and classification (line, border or field).

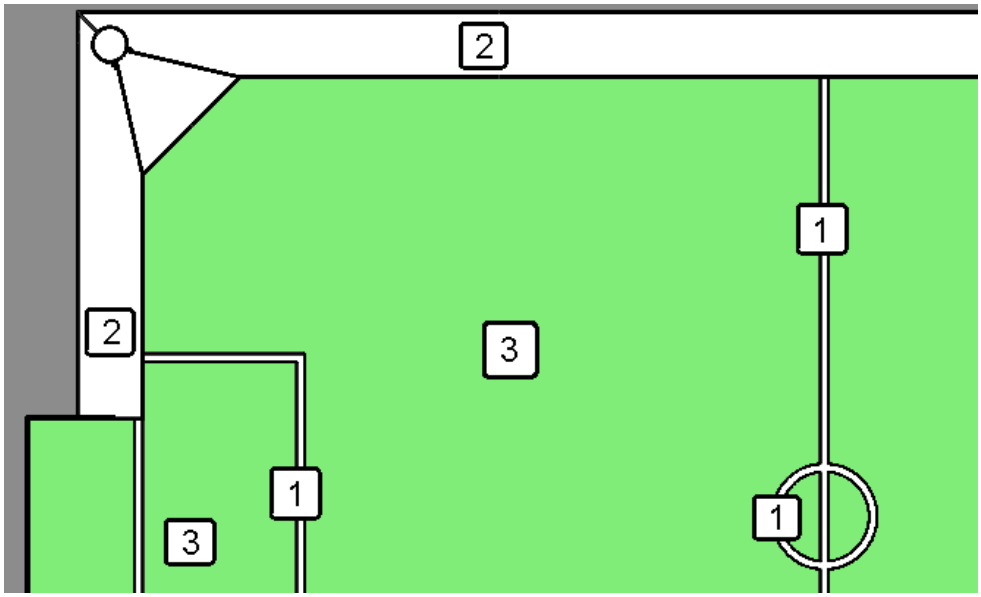

Figure 3.1: Examples of different point types

Detection of the field points is described in the next section. Because of their similarity, the line and border point types are detected in the same procedure. This involves scanning across the c-plane in scan lines that a form a regular horizontal and vertical grid. The spacing between grid scan lines is 10 pixels; the lower resolution is used because scanning the whole c-plane would be very time consuming and would result in hundreds (possibly thousands) of points which would further reduce the speed of the field line localisation code. The lines only cover the area of the c-plane that is below the estimated horizon, so that the time taken scanning and processing the discovered points is not wasted by having to cull points that are part of the background.

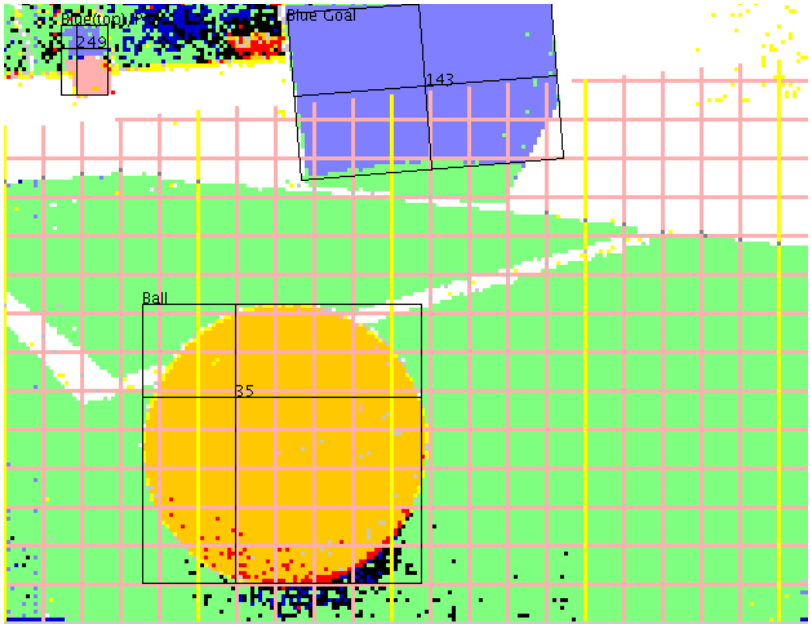

Figure 3.2: The scan grid for line and border points, shown as pink and yellow lines. Notice how they terminate at the horizon. The yellow lines indicate where field points are also checked for.

When the scan encounters a transition from green to white, the position of the transition is remembered. If there is a corresponding white to green transition within a certain distance further along the scan, then the total green, white, green transition is classified as a field line, and the average of the two transitions is recorded as the location of a "line point". If the transition from white back to green doesn't happen for a large distance, or doesn't occur before reaching the edge of the screen, then the white region is classified as a border, which needs to be marked at its base with a point. Working out which side of the white region is the base is done using the following rules:

- If the scan line is vertical, the lower of the two transition points must be the base, and so the point is recorded here.
- If the scan line is horizontal, the side of the white area that is closest to the centre of the screen is the base, so this is marked with a point.

The white section for a border has a minimum length that needs to be satisfied. This minimum is a value greater than that of the maximum length allowed for a line to be classified. Thus there is number of white region lengths that fall between the border minimum and line maximum, and these cases are regarded as ambiguous; since it cannot clearly identify the feature as either a line or a border, no point is recorded.

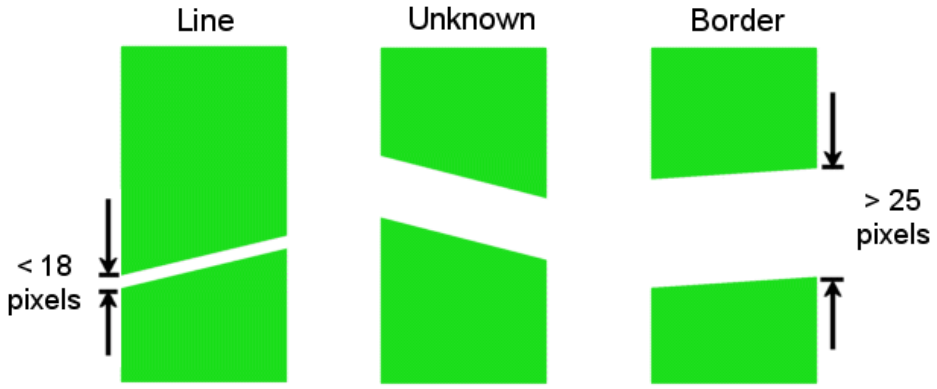

Figure 3.3: Different feature lengths produce different point types

The green / white transition detection is somewhat noise resistant. It allows for one pixel of noise (i.e. anything other than green or white) to occur between the green and white areas. This was added because examination of the c-plane shows that the white field lines can often be classified yellow, grey or even black near the sides. Another countermeasure against noise is the requirement that there be two consecutive green pixels as part of the green to white transition. This was introduced so that single green pixels followed by single white pixels, which can occur in a robot or in the background, don't get classified as a line or border point.

3.1.2 Deep Field Points

Detection of deep field points is slightly different to that of lines and borders. With field points, only vertical scan lines are used, and the spacing between consecutive scan lines is five times larger. Because the c-plane usually contains vast stretches of green, using less scan lines avoids an excess of field points. The field point detection doesn't look for colour transitions, instead it detects green points that are surrounded (along the scan line) by green.

The detector uses a "field interval" value that defines the number of green pixels that any green pixel needs to be surrounded by in order to be classified as a field point. Also, consecutive field points cannot be within this field interval of each other (again, this is to avoid an excess number of points). The length of the field interval diminishes as the scan moves up towards the estimated horizon, or the top of the c-plane if the horizon is off screen. In most cases, areas near the top of the screen correspond to points that are further away on the field. Because the distance follows an inverse relationship, having constantly spaced field points would results in many field points being projected close to the robot while the density of points far away from the robot would be low. Decreasing the field interval near the horizon allows the field points to be closer together, and so the projected point density is evened out somewhat.

The projection compensation is not infallible due its assumption that higher points on the c-plane correspond to further distance from the robot. This is not always the case, for example if the robot's head is tilted down and panned fully to the left. In this case the points furthest from the robot are on the left side of the c-plane, and points the distance difference between points at the top and bottom of the c-plane is minimal.

3.1.3 Sanity Checks for Points

There exists a pre-requisite for the point detection to be run at all. The ERS-7's method of retrieving camera frame information has some side-effects that do not become apparent until the head is moving. The frame data is retrieved from the camera sequentially from left to right, meaning the pixel data from the left of the frame may represent an earlier time to that of the right side of the frame. This means that when the head is turning quickly, the camera image can become warped, making normally straight line appear curved, and perpendicular angles acute or obtuse. To avoid the inaccurate point positions that would be detected from such fast moving frames, the c-plane scanning for points is not run at all if the joint angle sensors in the head are changing quickly.

After detection, the points must pass several tests, or "sanity checks", before they are used in the localisation system. Some of the tests apply to all point types, while other tests are specific to the different point types (line, border or field).

There are two tests that all point types must pass. The first test makes sure the point isn't inside the bounding box of the ball or a robot. The ball is classified mainly orange in the c-plane, however, there can be white areas near the edges due to specular reflections from the field lighting. Other robots seen in the c-plane feature a lot of white, but also many other colours due to the large number of reflections and shadows present from the irregular shape of the robot. The face, in particular, has a highly reflective surface with a dark grey backing, and it can be classified all manner of colours, including field green and white. Due to the noise colours inside these objects, sometimes line points are mistakenly identified by the detector. This check aims to remove those points. It does a good job, as long as the object detection works well. In general, the rUNSWift 2004 ball detection is very good, and no mistaken line points turn up inside the ball. The robot detection, on the other hand, is not so good and can be very bad when close to another robot, i.e. when it fills most the robot's view. Unfortunately, close-by robots also tend to have larger amounts of green on their surface, which means that a close encounter with another robot can throw off the field line localisation, due to a large number of false line points. It is hoped that improved robot detection in the future can limit this problem.

The second of the general checks is to make sure that the point is not in the corner of the c-plane. The camera in the ERS-7 tends to have very poor colour resolution at the frame's corners, which can lead to misclassified pixels. In general, points detected in these areas cannot be trusted to be correct, and so all are thrown away. The test is simple in implementation: for each corner, the Manhattan distance from the point to the corners is calculated, and if it is less than a threshold, the point is discarded. That is, for any of the four corners, if

$$|p_x - c_x| + |p_y - c_y| < B \qquad \text{(Eq 3.1)}$$

where: (p_x, p_y) is point pixel position
(c_x, c_y) is corner pixel position
B is the corner buffer (set to 10 pixels)

is satisfied, then the point is discarded.

After passing the general tests, each point must then pass a final test which is tailored to individual point types. The test for line points involves checking the two pixels either side of the point that are perpendicular to the scan line direction, if either of the points are green, the point is discarded. The reasoning behind this test is that any green, white, green transition along the scan line will be classified as a line point, even a single noise pixel of white in a field of green. To avoid these noise pixels, it is sensible to ensure it is part of a line. Valid line points occur at the intersection of the scan lines and the field lines, and the majority of intersections occur when the scan line and field line are nearly perpendicular, so checking the two pixels perpendicular to the scan direction is an approximation of checking along the length of the field line. This test is effective at removing any single pixel noise. It introduces the possibility of discarding valid line points when the field line appears diagonally on the c-plane of the robot. However, this is rare because the robot is really only able to see diagonal field lines at close ranges, and in these cases the line will appear a few pixels thick, enough for the point to pass the test.

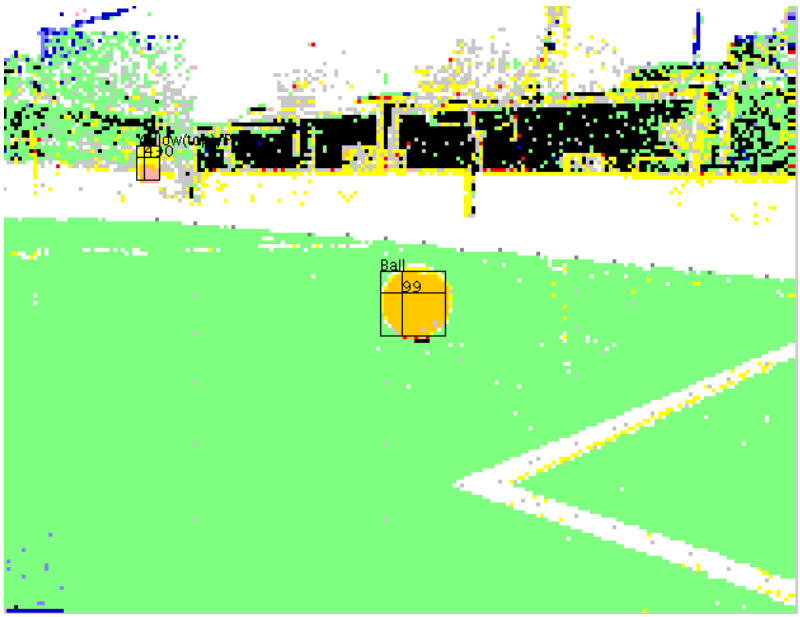

Figure 3.4: This c-plane features many single white noise pixels

amongst the green field areas. Some of these would be wrongly classed as line points if the line sanity check was not present. Line and border points are marked by single grey pixels.

Border points must pass a more stringent test. If the borders are examined from above, as in figure 3.5, it can be seen that most of the border subtends an angle of at least 180˚, e.g. the straight edges of the border occupy 180˚, while in the corners, the border can occupy an angle of 225˚. When viewed from the robot's perspective, this means that the border will generally occupy an angle of at least 180˚, when considering the base of the border (which the detected border points are located at). This allows for a strict sanity check: since the border occupies at least 180˚ at the border points, both the horizontal and vertical directions should contain large amounts of white in one direction (See figure 3.6).

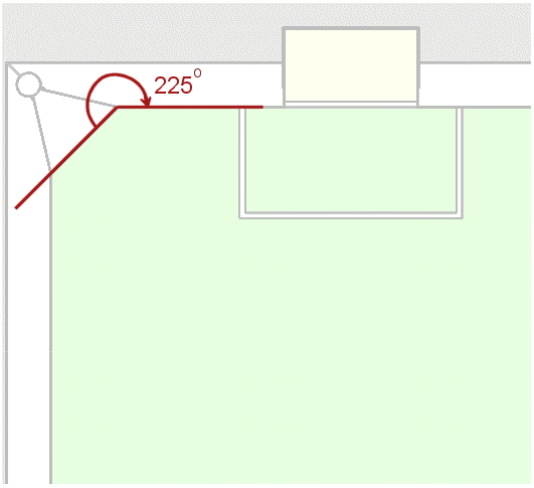

Figure 3.5: Overhead border angle.

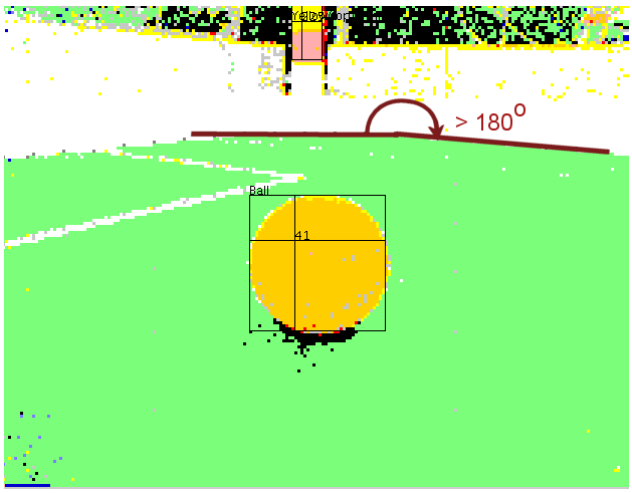

Figure 3.6: Border angle from the robot's view.

The test for each border point involves checking two lines of 15 pixels perpendicular to the scan direction, and either side of the proposed border point. One of these lines must contain no green for the point to be allowed. A check along the lines parallel to the scan direction is also done, however this is largely redundant, since to be detected there must a minimum amount of white along the scan direction anyway (See section 3.1.1). The main purpose of this sanity check is to avoid misclassification error. If a corner of a field line is seen at the edge of the c-plane, then the detector will wrongly classify it as a field border. This check will reject any points that are found in this way, as in figure 3.7.

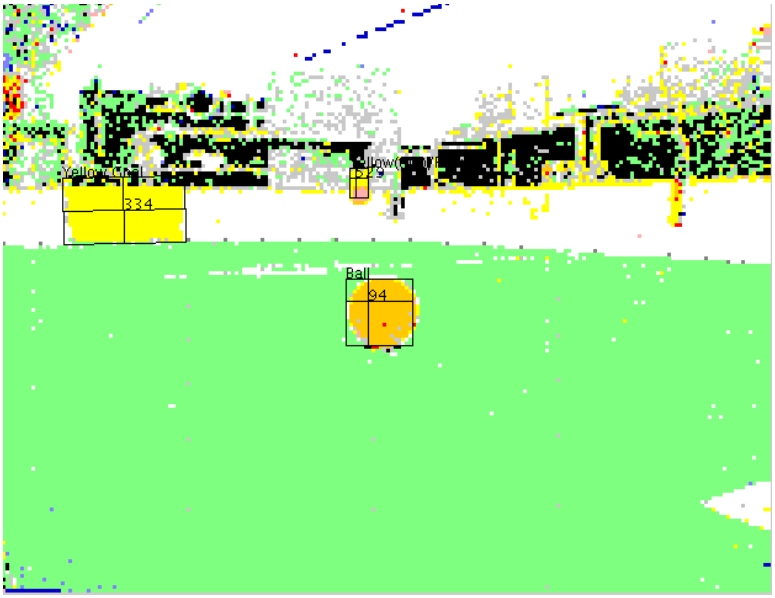

Figure 3.7: The corner of the goal box, visible in the bottom right of this c-plane, would have had incorrect border points marked on it if not for the border sanity check.

The check for field points is relatively simple, it checks in a line perpendicular to the scan direction (i.e. horizontally, since scanning is only done vertically for the field points) to make sure the field point really is surrounded by a lot of green. If there are more than two non-green pixels in the 15 pixels to the left and right of the field point, it is rejected.

3.2 Evaluating Field Line Information

3.2.1 Projection and Point Weighting

After the points of interest have been detected on the c-plane, the next stage to using the field information is to project those points from screen coordinates down onto the horizontal plane representing the field, so that they are in "robot local" coordinates. The robot has sensors in all of its joints, so by taking into account the physical dimensions of the robot, the angular position at which every joint is at, as well as the mounting location of the camera, the camera position (relative to the base of the robot's neck) and the direction the camera is facing can be calculated. Knowing the vertical and horizontal field of view and resolution of the camera allows us to project each point onto the ground plane (see page 10 of [18] for details).

The projection process is not entirely accurate, it can suffer noise from two different sources. One source is noise in the c-plane. This can cause otherwise straight edges to have pixel sized bumps in them occasionally. This can cause a line or border point to be moved from its "real" position one or two pixels. If the point is moved across, it is not so bad, but if the point is closer to the horizon and is moved up or down, it can make a large difference to where the point gets projected in robot local coordinates. The further away the projected point is from the robot, the more difference a movement of one pixel will make. In practice, this is not such a big problem, as it only affects one of the points, while the line localisation considers the contribution from all points. Also, pixel deviations aren't that common, and so most are avoided by the coarse scan grid used by the point detector.

Another source of error in the point projection is wrong joint sensor readings. This can lead to a wrong approximation of the camera's position or direction. These errors are more serious than the c-plane pixel errors since they affect the projection of all the points of interest, and so can make a bigger difference to the accuracy of the line localisation. The camera angle estimation especially will tend to make the largest difference in the projection of the points. Again, it is the points that get projected furthest away (the points on the c-plane that appear closest to the horizon) that will be affected the most by errors here. A wrong camera angle estimation will tend to affect the point projection distance the most, and this in turn will affect the angles between points that make up the field lines, skewing parallel lines, and make perpendicular intersections acute or obtuse.

To reduce the influence these wrong joint readings can have, each point is weighted after it has been projected. The weight reflects how much influence a point will have on the overall localisation. Since points that are projected to large distances are more affected by the angular changes, they are weighted less than the closer by points, which are more accurate. The weighting given to a point is a piecewise linear function of distance only, and takes the form shown in figure 3.8. The weighting is constant up to a distance of 70cm. From there it reduces down to zero at a distance of 350cm. Note that only the line and border points are weighted in this fashion. The field points don't have to be as accurate as the other two, since they indicate a large area of green. They are weighted constantly up to 350cm, and from then on receive no weight.

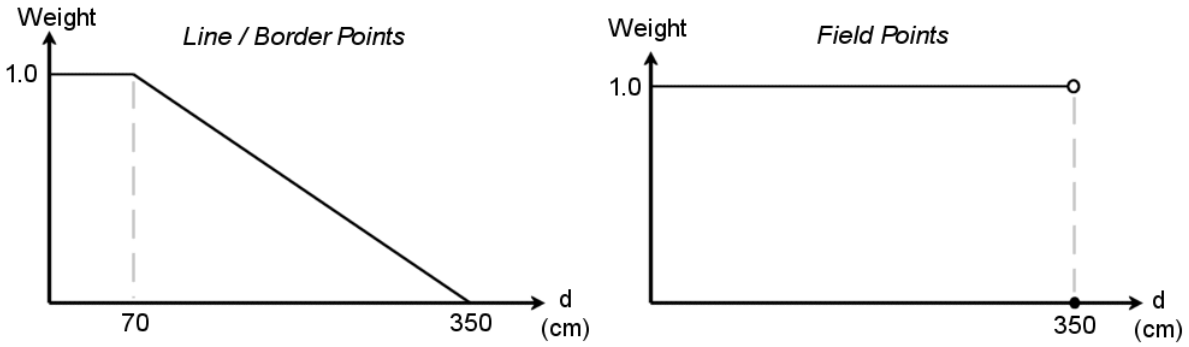

Figure 3.8: Graph of point weightings for line, border and field.

Points that are projected to very large distances receive no weight at all, and are culled from the point set so no further processing is done on them. The culling for far distance points is done not just because these points are more susceptible to angular errors, but because many of the very far points shouldn't actually have been found by the detector in the first place, e.g. they are mistaken line classifications from the background or the surface of a robot.

As a final check for this stage, the number of points in each classification group is counted. If there are under a certain number (in this case 4) of points of the same type, then none of the points of that type are used. This is done to prevent use of misclassified or false points. For example, if the robot is observing only the border of the field, but some noise in c-plane makes it mistakenly misclassify one of the border points as a line point, then that point will not be used.

3.2.2 Calculating Point Match Values

To make use of the projected point information, first we need a measure of how likely it is for each point type to lie at a particular location on the field. The field dimensions and line configuration are set by the Robocup rules and do not change during the course of the match; this means a look-up table of how likely it is to find the three different point types at any global field position can be pre-calculated when the robot first boots up. There are three look up tables, one for each point type, and each table is a two dimensional array (corresponding to the two global coordinates x, y) of floating point numbers, with values close to zero indicating low probability of finding a point there, and values close to one indicating high probability. The size of the look-up table determines the spatial accuracy of the given results. It was determined that an accuracy of 2 cm per array element gave a good accuracy / memory use compromise. The look-up tables are also called "match" tables, since they indicate how well a point matches its position.

The tables hold the probability of seeing a particular point type at a certain field position, with the two indices into each table corresponding directly to global field x, y coordinates divided by two, since the global coordinates are measured in centimetres, and an offset added, since we want to be able to evaluate the probability of points that lie off the field as well. An offset of 200cm (or 100 array elements) is used to create a buffer all around the field, hence an extra 200 array indices were added to each of the array dimensions, on top of the indices required to represent the entire field area.

To calculate the values contained in the match tables, the Euclidean distance to the nearest type feature is used. A "type feature" refers to the nearest field line for the line look-up table, the nearest border for the border look-up, and the entire green field area for the field table. To get the distance, the smallest distance to every type feature is calculated, then the smallest distance value is taken. The distance d_{min}, which is measured in centimetres, is then substituted into equation 2.2:

$$P(d_{\min}) = \frac{1}{1 + Sd_{\min}^2} \qquad \text{(Eq 3.2)}$$

where: S is a spread constant (Set to 0.004)

This function was used because it has the correct range of (0, 1], is continuous, and doesn't decay as quickly as the other function that was originally used, based on the normal distribution:

$$P(d_{\min}) = \exp\left(-Sd_{\min}^2\right) \qquad \text{(Eq 3.3)}$$

where: S is again a spread constant (Set to 0.001)

The P value from equation 3.2 is recalculated and assigned for every look-up table element. Obviously this is a computationally expensive process, having to calculate a distance to every type feature for every position in the look-up tables, but this is acceptable since the table only has to be setup once.

The resulting match tables are shown in figure 3.9. These tables were sufficient for use on UNSW's home field, where it is not possible for the robot to see any of the floor between the field border and the outer walls. In the Robocup tournament in Lisbon, the fields were constructed differently, with a large amount of ground between the field borders and outer walls visible by the robots. This ground was covered with same green material as the inner field. This meant that whenever a robot was near the normal field border, they would see and classify an extra set of field and border points. To stop this causing localisation problems, and to make use of the extra available information, the match look-up tables were modified, with an extra outer border, and more field between the outer and inner borders. These new look-up tables are shown in figure 3.10.

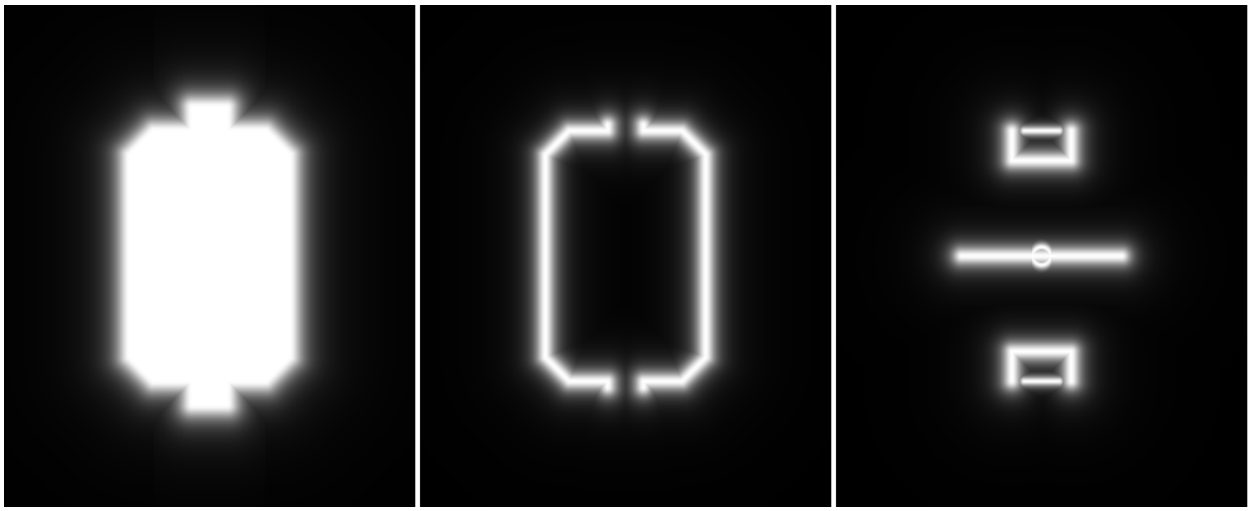

Figure 3.9: Original match lookup tables for field, border and line points.

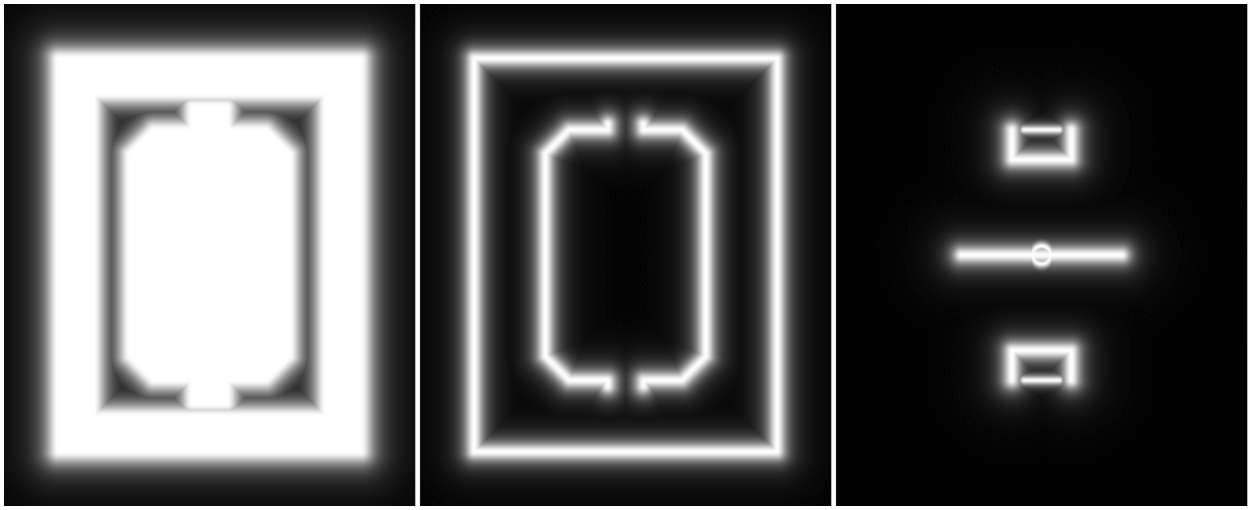

Figure 3.10: Altered field, border and line match lookup tables for Robocup tournament in Lisbon.

To retrieve the match value for a particular field position, we first convert the global field coordinates to the index space of the look-up tables:

$$(x_i, y_i) = \frac{1}{2}(x, y) + (x_{offset}, y_{offset}) \qquad \text{(Eq 3.4)}$$

where: (x_i, y_i) are the index space coordinates
(x, y) are the global field coordinates (in centimetres)
(x_{offset}, y_{offset}) are the look-up table offsets (= (100, 100))

The index space coordinates need to be rounded off to the nearest integer, so that we may access a particular array element:

$$(x_{ind}, y_{ind}) = (\text{round}(x_i), \text{round}(y_i)) \qquad \text{(Eq 3.5)}$$

where: (x_{ind}, y_{ind}) are the rounded off indices
round is a function equivalent to: $round(v) = \lfloor v + 0.5 \rfloor$

Two "bias" values are now defined, which indicate how close the index coordinates are to table elements neighbouring the element indicated by (x_{ind}, y_{ind}):

$$(x_b, y_b) = (x_i, y_i) - (x_{ind}, y_{ind}) \qquad \text{(Eq 3.6)}$$

where: (x_b, y_b) are the bias values, both in the range [-½, ½)

It is assumed that the match values in the vicinity of the element indicated by (x_{ind}, y_{ind}) are approximated by a quadratic function of the bias values:

$$M(x_b, y_b) = ax_b^2 + by_b^2 + cx_b + dy_b + e \qquad \text{(Eq 3.7)}$$

where: M is the match function.

a, b, c, d and e are unknown coefficients.

Notice that in equation 3.7, there is no term in xy. Not including the xy term means there are only have 5 unknown coefficients, which can be found by taking the match values from the look-up table element at (x_{ind}, y_{ind}) and its four vertical and horizontal neighbours, as shown in figure 3.11. It would be possible to include an xy term, where the extra coefficient could be found by taking the diagonal neighbour closest to the index space coordinates (x_i, y_i), however it was not included to keep the look-up process simple.

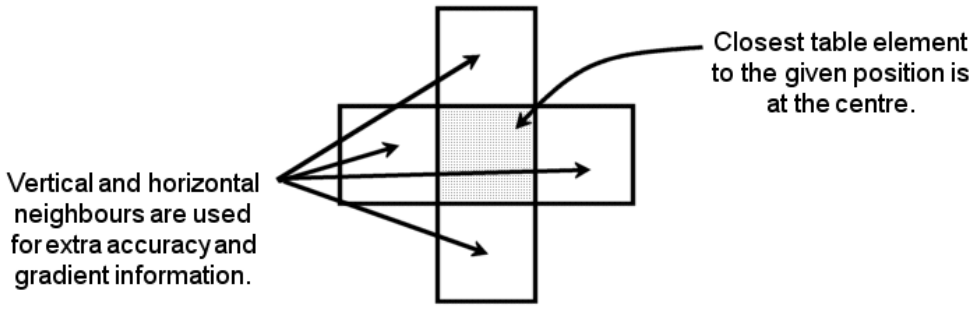

Figure 3.11: The table element and its four neighbours.

The table element at (x_{ind}, y_{ind}) corresponds to bias values of (0, 0), while the bias values of its neighbours are (0, -1), (-1, 0), (1, 0), (0, 1). Since the look-up table gives us the match values at these bias points, we can solve for a, b, c, d and e, giving:

$$a = \frac{1}{2}(m_{1,0} + m_{-1,0}) - m_{0,0} \qquad b = \frac{1}{2}(m_{0,1} + m_{0,-1}) - m_{0,0}$$
$$c = \frac{1}{2}(m_{1,0} - m_{-1,0}) \qquad d = \frac{1}{2}(m_{0,1} - m_{0,-1}) \qquad \text{(Eq 3.8)}$$
$$e = m_{0,0}$$

where: $m_{i,j}$ corresponds to the match table value at bias (i, j)

Using the above coefficients, it is then possible to substitute the bias values for the point's global coordinates into equation 3.7 to get a resulting match value, however, this method is not used because it results in more discontinuities in the match value than a linear function, as in figure 3.12.

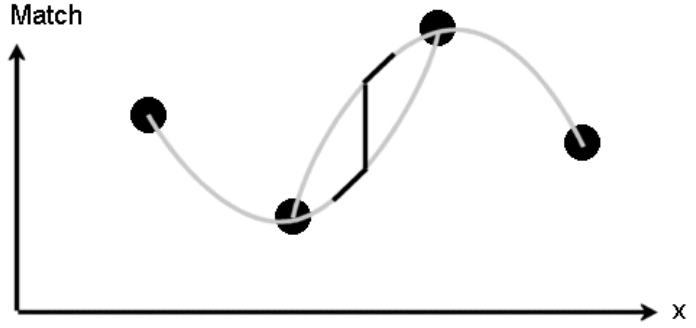

Figure 3.12: If equation 3.7 were used, the match value in one direction would be a quadratic function of the nearest three match table values. If match values were evaluated along one axis, there would be discontinuities half way between points, since one of the old neighbours would be dropped, and a new neighbour would come into use.

Instead, a simple linear interpolation method gives us the match value:

$$M(x_b, y_b) = m_{0,0} + |x_b|(m_x - m_{0,0}) + |y_b|(m_y - m_{0,0}) \quad \text{(Eq 3.9)}$$

where:

$$m_x = \begin{cases} m_{1,0} & \text{if } x_b > 0 \\ m_{-1,0} & \text{otherwise} \end{cases}$$

$$m_y = \begin{cases} m_{0,1} & \text{if } y_b > 0 \\ m_{0,-1} & \text{otherwise} \end{cases}$$

While equation 3.7 is used to supply information on the match derivative with respect to x and y:

$$\frac{\partial M}{\partial x} = \frac{1}{2}\frac{\partial M}{\partial x_b} = \frac{1}{2}(2ax_b + c) \qquad \frac{\partial M}{\partial y} = \frac{1}{2}\frac{\partial M}{\partial y_b} = \frac{1}{2}(2by_b + d)$$

$$\begin{aligned} \frac{\partial M}{\partial x} &= \frac{1}{2}\left[\left(x_b + \frac{1}{2}\right)(m_{1,0} - m_{0,0}) + \left(x_b - \frac{1}{2}\right)(m_{-1,0} - m_{0,0})\right] \\ \frac{\partial M}{\partial y} &= \frac{1}{2}\left[\left(y_b + \frac{1}{2}\right)(m_{0,1} - m_{0,0}) + \left(y_b - \frac{1}{2}\right)(m_{0,-1} - m_{0,0})\right] \end{aligned} \quad \text{(Eq 3.10)}$$

$$\frac{\partial^2 M}{\partial x^2} = \frac{1}{4}\frac{\partial^2 M}{\partial x_b^2} = \frac{1}{2}a \qquad \frac{\partial^2 M}{\partial y^2} = \frac{1}{4}\frac{\partial^2 M}{\partial y_b^2} = \frac{1}{2}b \qquad \frac{\partial^2 M}{\partial x \partial y} = 0$$

$$\begin{aligned} \frac{\partial^2 M}{\partial x^2} &= \frac{1}{4}\left[(m_{1,0} - m_{0,0}) + (m_{-1,0} - m_{0,0})\right] \\ \frac{\partial^2 M}{\partial y^2} &= \frac{1}{4}\left[(m_{0,1} - m_{0,0}) + (m_{0,-1} - m_{0,0})\right] \end{aligned} \quad \text{(Eq 3.11)}$$

Currently, the second derivatives are not used, but they are stated here in case the field localisation is extended in the future and needs to make use of them. In summary, the look-up process for a single point is as follows:

1. Calculate look-up table index and bias values for the given point's global field coordinates, using equations 3.4, 3.5, 3.6.
2. Calculate the match value using equation 3.9.
3. Calculate the first derivative of the match value using equation 3.10.

3.2.3 Calculating Overall Match

A process is required that evaluates how well a given robot position and heading (in global coordinates) matches the field configuration seen on the c-plane. Section 3.2.1 supplies the robot local coordinates of the detected points from the c-plane, and a weighting factor for each point. Section 3.2.2 supplies a process for computing the match of an individual point, given its global coordinates. The remaining task is to transform all point

coordinates from robot-local to global coordinates, and to evaluate the match of all points to obtain an overall match value.

Transforming from robot-local to global coordinates is a simple task of a rotation by the robot heading followed by a translation by the robot field position. If a point in robot local coordinates is denoted by the 2D vector $\underline{l}$, its position in global coordinates by $\underline{g}$, and the robot global coordinates and heading by x_r, y_r and h_r, then this can be represented by the formula:

$$\underline{g} = \underline{\underline{R}}\left(h_r - \frac{\pi}{2}\right)\underline{l} + \begin{pmatrix} x_r \\ y_r \end{pmatrix} \qquad \text{(Eq 3.12)}$$

where: $\underline{\underline{R}}(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$ is the matrix of a rotation of θ radians.

Once all points have been converted to global coordinates, an overall match value can be calculated. The overall match is defined as the product of the weighted average of match values for line and border points, and the weighted average of match values for field points. Since the points detected from the c-plane are stored as an array, each point will be referred to individually by using an integer subscript, such as *i*. That is:

$$O(\underline{r}) = A_L A_F \qquad \text{(Eq 3.13)}$$

where: O is the overall match value.
$\underline{r} = (x_r, y_r, h_r)^T$ is a 3-vector of the robot's global position and heading.
A_L is the weighted average of match values for line and border points.
A_F is the weighted average of match values for field points.

$$A_L = \frac{1}{W_L}\left(\sum_{i \in line} w_i M(\underline{g}_i)\right) \qquad A_F = \frac{1}{W_F}\left(\sum_{i \in field} w_i M(\underline{g}_i)\right) \qquad \text{(Eq 3.14)}$$

where: M is the match for a global position, as calculated in section 3.2.2.
line is the set of point indices that are either line or border points.
field is the set of point indices that are field points.
w_i is the weight of the point with index *i*.
$\underline{g}_i$ is the global position of the point with index *i*. It is a function of $\underline{r}$.
W_L and W_F are the total weights of line & border points, and field points.

$$W_L = \sum_{i \in line} w_i \qquad W_F = \sum_{i \in field} w_i \qquad \text{(Eq 3.15)}$$

For localisation purposes, the gradient of the overall match is also required. First, the chain rule is applied to equations 3.14:

$$\frac{\partial A_L}{\partial x_r} = \frac{1}{W_L}\left(\sum_{i \in line} w_i \frac{\partial M}{\partial x}(\underline{g}_i)\right)$$

$$\frac{\partial A_L}{\partial y_r} = \frac{1}{W_L}\left(\sum_{i \in line} w_i \frac{\partial M}{\partial y}\left(\underline{g}_i\right)\right) \qquad \text{(Eq 3.16)}$$

$$\frac{\partial A_L}{\partial h_r} = \frac{1}{W_L}\left(\sum_{i \in line} w_i \left(\underline{\nabla} M(\underline{g}_i)\right) \bullet \left(\underline{\underline{P}}\underline{l}_i\right)\right)$$

(Similarly for A_F)

where: $\underline{\nabla} M\left(\underline{g}_i\right) = \left(\frac{\partial M}{\partial x}\left(\underline{g}_i\right), \frac{\partial M}{\partial y}\left(\underline{g}_i\right)\right)^T$ is the point match gradient.

$\underline{\underline{P}} = \underline{\underline{R}}\left(h_r\right) = \begin{pmatrix} \cos\left(h_r\right) & -\sin\left(h_r\right) \\ \sin\left(h_r\right) & \cos\left(h_r\right) \end{pmatrix}$ is a h_r radian rotation.

$\underline{l}_i$ is the local coordinates of the point with index i.

● is the dot product operator.

$\underline{P}\underline{l}_i$ is the vector corresponding to the direction and speed that each point would move at, if the robot rotated.

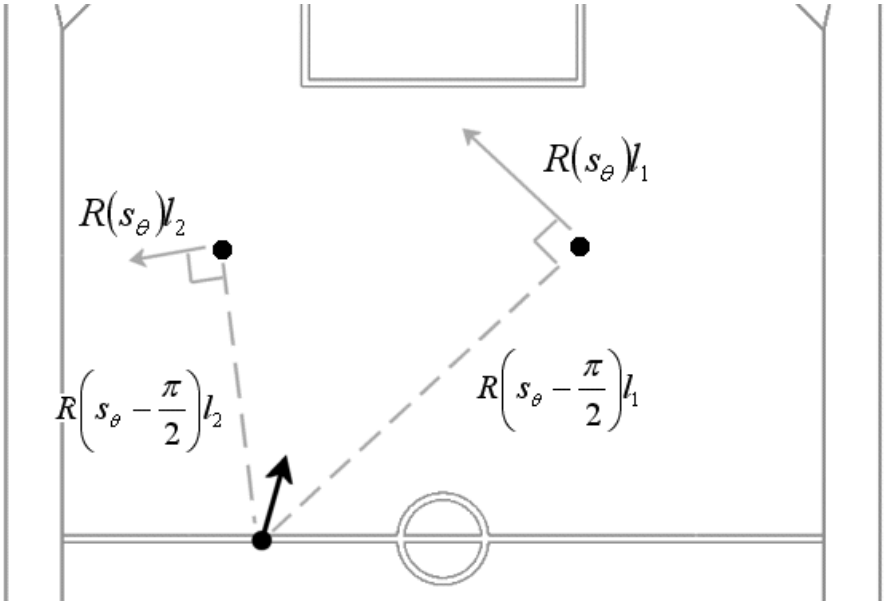

Figure 3.13: The movement of any point when the robot rotates is perpendicular to the displacement of the point from the robot, and proportional in length. Hence we can obtain the movement vector by rotating the local point coordinates by an extra 90 degrees. Moving the robot in either the x or y directions simply moves every point by the same vector.

Figure 3.13 shows how the above equations were derived. The overall match gradient is then found by using the product rule on equation 3.13:

$$\underline{\nabla} O(\underline{r}) = \begin{pmatrix} \left(\frac{\partial A_L}{\partial x_r} A_F + A_L \frac{\partial A_F}{\partial x_r} \right) \\ \left(\frac{\partial A_L}{\partial y_r} A_F + A_L \frac{\partial A_F}{\partial y_r} \right) \\ \left(\frac{\partial A_L}{\partial h_r} A_F + A_L \frac{\partial A_F}{\partial h_r} \right) \end{pmatrix} \qquad \text{(Eq 3.17)}$$

It should be noted that if there were no detected field points then A_F is set to 1 (for all r). Similarly, if there were no detected line or border points, A_L is set to 1. Doing this allows match values to still be obtained if only one point type is seen on the c-plane. In summary, to calculate the overall match at a particular robot position and heading:

1. Transform all points from local to global coordinates using equation 3.12.
2. Calculate the overall match value using equation 3.13.
3. Calculate the overall match gradient using equation 3.17.

It can be simpler to study the implementation of this process rather than the mathematics behind it, since the algorithm is less cluttered than the formulas. The process is implemented in the "calcRobotMatch" function in the file "VisualCortex.cc".

3.2.4 Stationary Mapping

If the robot is standing still then the field is not moving relative to the robot. Hence the configuration of lines, borders and field green around the robot also does not change with respect to the robot. It is possible to take advantage of this fact, so as to better utilise the field information and localise off it more accurately. As the name suggests, "stationary mapping" involves building up a map of the surrounding field features when the dog is standing still. Normally, when the robot is moving, the field localisation will only use the detected points in the current c-plane to help localise off; when a new c-plane arrives the points from the previous plane are thrown away. With stationary mapping, the old points are not thrown away (as long as the robot is standing still), instead, all newly detected points are compared to a "map", and the are incorporated into the map.

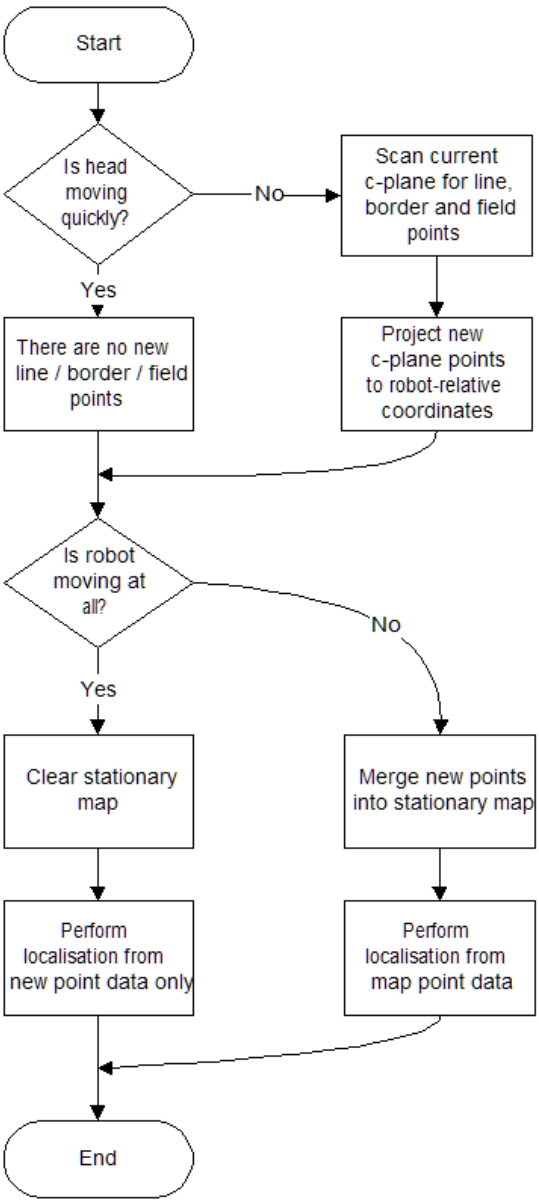

Figure 3.14: Field localisation pipeline and stationary mapper

The map is very similar to the array used to store the detected points from the c-plane, once they have been projected down into robot-local coordinates. It is an array of points that holds information on the x, y position, type and weight (or confidence) value for each of the points. Figure 3.14 shows how stationary mapping fits into the field localisation process.

The localisation process uses the stationary map data if the robot isn't moving, and if it is moving, it reverts back to the local point list for the current c-plane only. Also, if the robot starts moving, then the map is reset, as it is no longer valid. The merging process is the most complex portion of the stationary mapper.

Since the robot local point list from the current c-plane and the stationary map are essentially of the same form, a naive approach to merging would be to add all points from the current c-plane to the end of the map's point list. This would get the job done, however, with around 50 c-plane points detected per frame and 30 frames per second, the stationary map

would quickly grow to a list of thousands of points, which would cause a dramatic increase in processor usage by the field localisation, such that the robot would not be able to do any other processing. Clearly, this is not useful. A merging process is needed that generalizes the surrounding field information into a list of about 100 points (the array size for the map is 128 points, the same size as the array that stores the detected points from the current c-plane), i.e. the merging process needs to minimize redundant information by not storing duplicate or near-duplicate points.

Two points are considered equivalent if they are the same type (both are line, border or field type), and if they have approximately the same x and y coordinates. Hence for every point in the map array we define a "merge region" around it. If any of the newly detected points from the current c-plane happen to be projected into one of the merge regions, that point is merged with the map point by taking a weighted average of the positions of the new point and the map point. The merged point then gains the weight of the newly added point. The weights of the points that have just been detected are all 1, i.e. the point weighting functions described in section 3.2.1 are not applied when the stationary mapping is in use. Another way of perceiving this process is that any newly detected point falling inside a merge region is a new estimate of that map point's position. When the new point is merged with the map point, the map point's weight is increased, and this equivalent to the confidence of that map point's position increasing.

If a newly detected point does not fall within the merge region of an existing map point, then it is added to the map. If the map array is already full, then the point is discarded. Periodically, the weight of all points in the map array is reduced by 1, and any points dropping to zero weight are removed from the map allowing for further points to be added.

Note that merging a newly detected point with a map point will alter the map point's position, and so it is possible that over time, one of the map points may move into the merge region of another map point of the same type. This would be a waste of map space, since both map points are describing nearly the same thing. To stop this from happening, all merged map points are treated the same as the newly detected points, albeit with a larger weighting. They are re-inserted into the map and merged again if they lie within an existing map point's merge-region. Obviously they have to be removed from the map array before inserting again, otherwise the process would go into an infinite loop, as the existing map point is continually merged with itself.

The merge regions for line and border points are the intersection of two simple area types. For a newly detected point to be merged with one of these map points, it has to lie within a certain distance of the map point, but also has to be within a certain angle of the map point, with respect to the origin (which represents the robot's position in robot local coordinates), as in figure 3.15. So the merge region consists of a "merge radius" which defines a circle around the map point, and a "merge angle" which defines a pie slice coming out from the origin, and has the map point in the centre of the slice. The merge angle stays fixed for all map points, but the merge radius increases as the map point's distance from the origin increases.

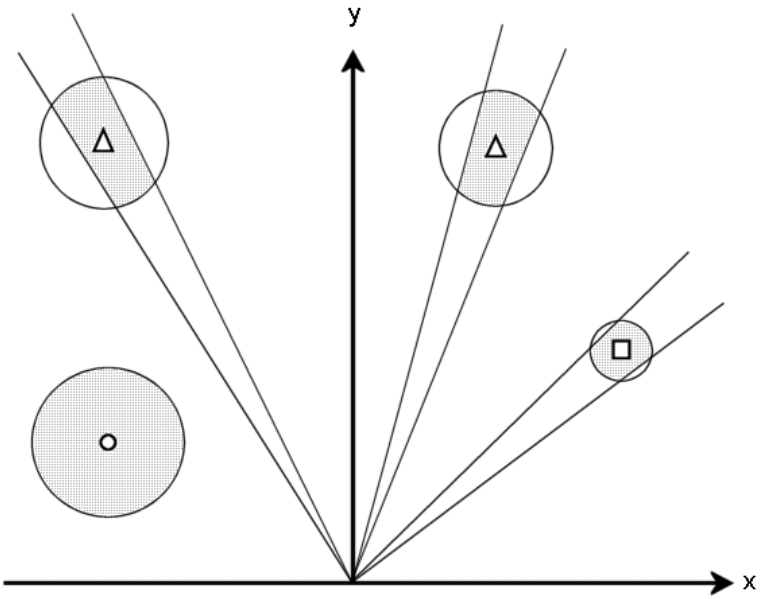

Figure 3.15: The merge regions of border (the small square), line (the two triangles) and field points (the small circle), in robot-relative coordinates. The merge radius for field points is bigger than for line/border points, and increases as the point distance increases. The merge angle is fixed to the same value for all line/border points.

This region type was chosen for two reasons:

- It is easy to check quickly. To check whether the test point lies within the map point's merge radius, simply check:

$$(p_x - m_x)^2 + (p_y - m_y)^2 \leq R_M^2 \qquad \text{(Eq 3.18)}$$

where: $\underline{p} = (p_x, p_y)$ is the test point position in local coordinates.
$\underline{m} = (m_x, m_y)$ is the map point position in local coordinates.
R_M is the merge radius.

To check if the test point is within the map point's merge angle, check:

$$\frac{(\underline{p} \bullet \underline{m})}{|\underline{p}|^2 |\underline{m}|^2} \geq \cos^2(\theta_M) \qquad \text{(Eq 3.19)}$$

where: θ_M is the merge angle.

Many values for the test can be pre-computed, while the rest involve multiplications and additions (and one division), which are quite fast operations. Speed is important for the tests, since O(n) map points need to be checked (where n is the number of map points) for every newly detected point when they are merged into the map.

- It corresponds to the error region associated with each point. Errors in a point's pixel position are equivalent to angular errors, because the camera has a fixed resolution and a fixed field of view angle, while errors in the joint sensors are angular errors as

well. An angular error that is parallel to the ground plane will be projected into the region somewhere inside the pie slice. An angular error that is perpendicular to the ground plane, i.e. vertical, will be projected to a donut shaped region about the origin. For a fixed error amount, as the projection distance increases, this donut will become thicker. For the merge region, the donut is approximated by the merge radius circle, which increases in radius as the map point distance increases.

The field points' merge regions are different to the regions for the other two point types; lines and borders. There are usually many field points detected per c-plane, and the field points only describe a general area of the green part of the field (so they do not need to be particularly accurate in there position), and so their merge regions consists of only the merge radius circle, but not the pie slice.

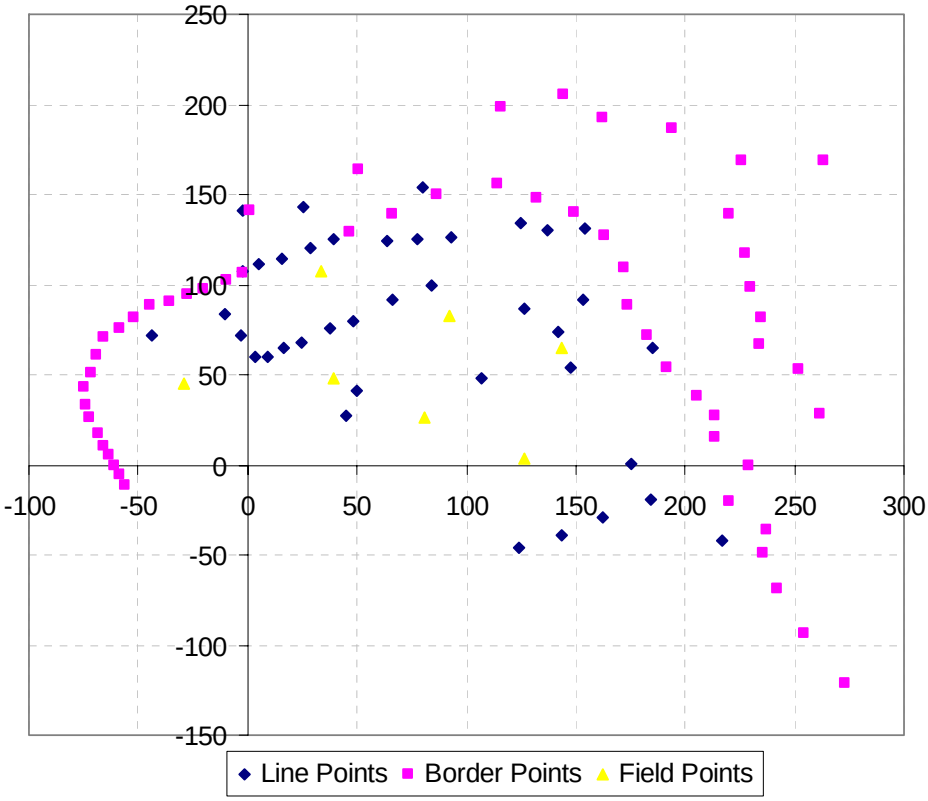

Figure 3.16: A map built by the stationary mapper when the robot was standing outside the new a goal box corner, facing towards the goal. This figure is in robot-relative coordinates, with units in centimetres. Note how projection accuracy decreases with distance, making the far border into two separate lines.

Although it is limited to use when the robot is not moving, when it is running the stationary mapper is very effective at localizing the robot to the correct field position. If the robot performs some head pans while not moving, it gains field data on a large angular range in front and to the sides of the robot. This is usually enough information to localise it to within a single point on the field, or perhaps its 180˚ rotational equivalent. In a Robocup match, there are not many times when it can be used by the forwards since they move too much. The goalie, however, is not moving all the time and it can make good use of it, especially when facing the nearby field corners. The stationary mapping was also very useful for the global localisation problem for the localisation challenge.

3.3 Field Line Updates

3.3.1 Match Gradient Ascent

The final stage of field localisation is to find a robot position and orientation that maximises the match between what is expected to be seen by the robot, calculated from knowledge of the field layout, and what field configuration is seen on the c-plane. The match value and gradient can be calculated for any robot position & orientation on the field, so the goal is to find the best of these with as few match evaluations as possible; since each evaluation requires the sum up to 128 separate point match calculations, it can be quite a computationally expensive operation.

The match gradient vector indicates the direction of greatest increase of the match value, so following the gradient vector will take us to the nearest local maximum of the match value. This "gradient ascent" method used in the rUNSWift 2004 code. It is assumed that the current highest probability position and orientation are somewhere in the vicinity of the actual values, and so hopefully the nearest maximum corresponds to the global maximum.

The implementation of the gradient ascent is quite simple. Moving in the gradient direction will always increase the match value, however, the gradient value changes as position and orientation changes. Since it is impossible for a computer to continuously check the gradient while updating the position, the movement has to be divided into discrete intervals. Starting at the current best estimation of the robot's position, the match value and match gradient are calculated. The gradient is multiplied by a movement rate scalar and added onto the current position and heading vector. The match value and gradient are then calculated at the new position. If the match value has improved, the new position is kept and the movement rate is increased slightly. If the new match value is worse, then the current position is reverted back to the old one and the movement rate is decreased. The process is then repeated at the current position.

The main problem here is that any one movement may have been too large, passing over a maximum, and then decreasing to a lower match value. To solve this, the movement rate is decreased whenever the match rate decreases. If the movement rate is too slow, it will take many steps to reach the maximum, but the number of steps needs to be limited to a certain number (50, in this implementation) so that performance is acceptable. Hence it is possible that the gradient ascent never reaches its maximum. To help prevent this happening, the movement rate is accelerated if the match value is increasing.

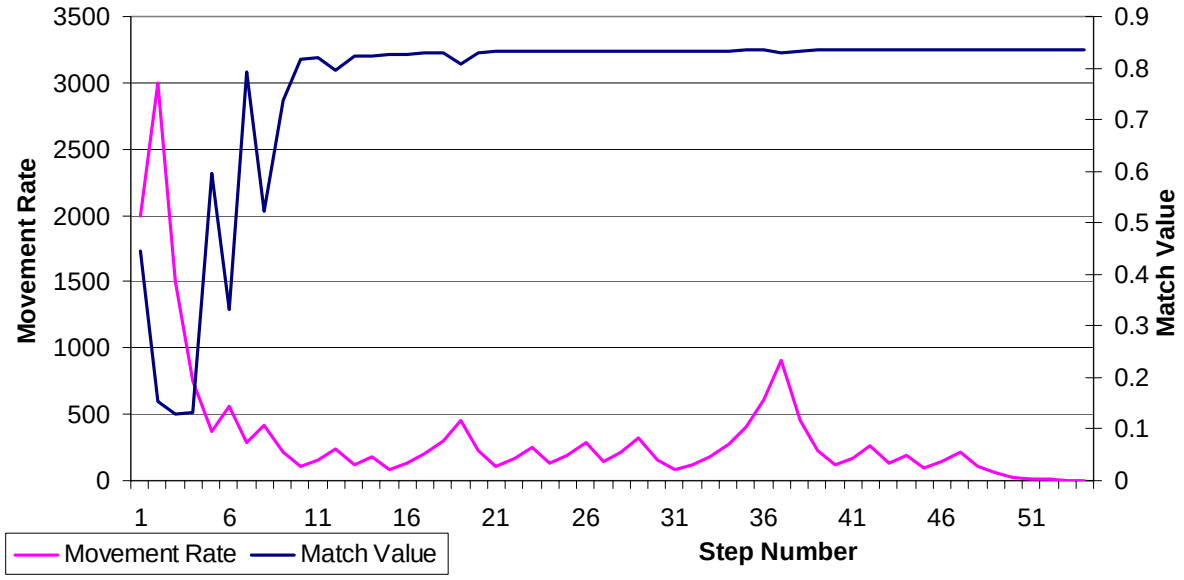

Figure 3.17: The movement rate usually drops quite quickly at the start of the ascent, levelling off for a period, then dropping again to zero once a local maximum has been reached.

The gradient ascent has an early exit mechanism. If the movement rate is very small, it shows that many past attempts at movement have resulted in a smaller match value, indicating a local maximum is nearby, and it is a waste of time to continue searching. Hence if the movement rate drops below a threshold value, the algorithm quits before its normal 50 steps are completed.

Because of processing time constraints, it is only possible to perform the gradient ascent for one of the modes (the mode with highest weight value) in the multi-modal Gaussian self location distribution. This is one of the disadvantages of using this method. Despite this, the method has proved to be effective at finding high match values, thus providing useful localisation information.

3.3.2 Symmetry Generation

After the gradient ascent has finished running, it returns an updated position with a match value that is greater than, or equal to in very rare cases, the match value at the starting position. This updated position and orientation tends to be located at a local maximum of the match value, but there is no guarantee that it is the global maximum, since there could be many local maximums for any given point configuration. As a precaution, some code was added, before the final field line update, that generates a list of field positions and orientations that are similar to the update position output by the gradient ascent. This code is called the "symmetry generator".

The symmetry generator works by defining "windows" around the some of the line features on the field, e.g. the centre circle, the 45˚ corners of the field border, the corners of the goal box. Windows are considered equivalent if the line features inside them are a rotation of each other; see figure 3.18 for examples of equivalent windows used by the symmetry generator. It then takes the median of the final global position of all line and border points after the gradient descent has been performed (here, the median is defined as the

closest line / border point to the mean of those points). If the median lies inside one of the windows, then alternate update positions are generated from all the other equivalent windows around the field. The update positions and orientations are rotated and translated accordingly so that the features inside each equivalent window appear the same.

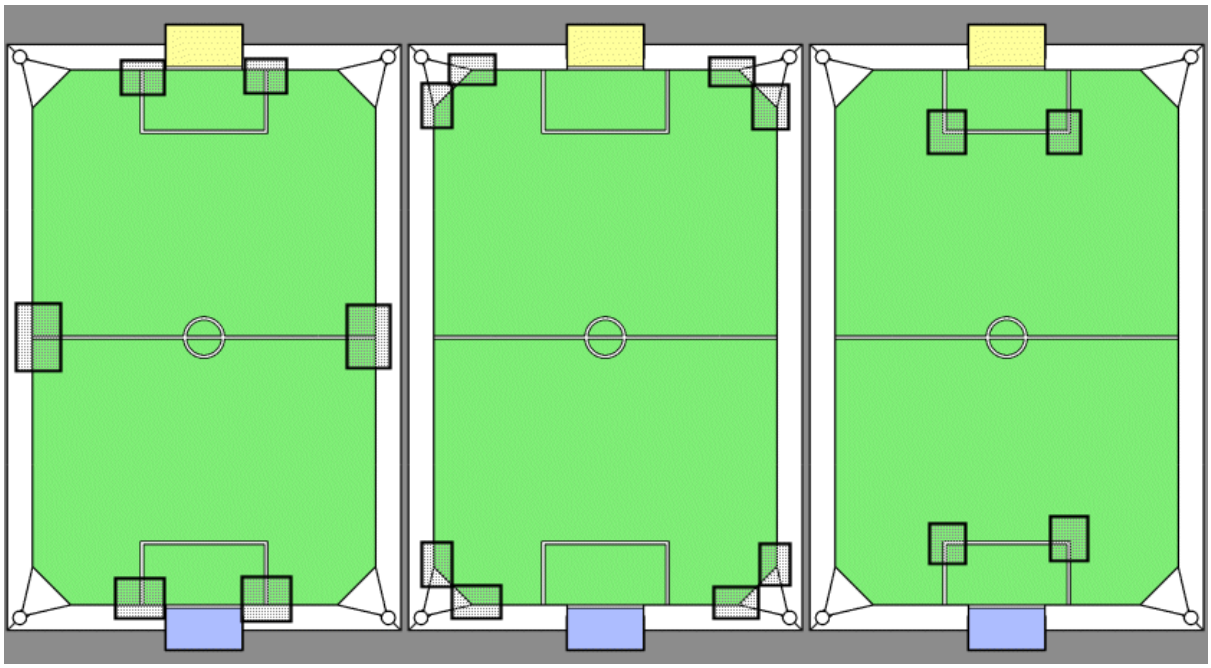

Figure 3.18: Examples of equivalent windows.

If the median does not lie inside a defined window, a simple 180˚ rotation about the field centre will generate an update position and orientation equivalent to the original, since the playing field is rotationally symmetric about the centre, when the resolving beacons and goals are ignored (which they are for the field localisation).

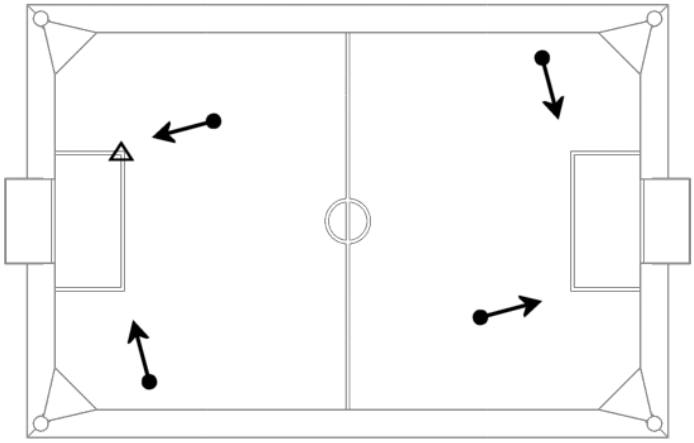

Figure 3.19: An example of the generated symmetries. The triangle represents the median line / border point.

The symmetry generator also generates position and heading variances which are used by the kalman filter for the final field update. It does this by supplying variance fixed values depending on the line feature present in the window used to generate the symmetries. If the line features inside the window are not resolvable in one direction, e.g. the window contains only a straight section of line or border, then the variance in the irresolvable direction is set to

infinity. In all other situations, all variance values are set to a constant. Similarly, if the median point did not lie inside any of the windows, all variance values are to the aforementioned constant.

Another, possibly more mathematically correct, method of determining the update variances would be to evaluate the second derivative of the match value at each update location and estimate a variance based on that. However, the previous simple method has proved easy to implement while still being effective, so other methods were not tried.

3.3.3 Kalman Update Conditions

After passing through the symmetry generator, the single update position and orientation gained from the gradient ascent becomes a list of update positions and orientations, each with an associated variance for the x and y position coordinates, and the heading h. The match value of each of these positions is calculated, and only those positions that pass the following threshold are put into the kalman filter:

$$mo \geq v \qquad \text{(Eq 3.20)}$$

where: m is the match value for the update position and heading.
o is an off-field penalty.
v is the minimum match value.

Occasionally, some of the generated symmetries are positions that are not actually on the field. These positions can still receive good match values despite there off-field positions if the global coordinates of the line / border / field points still lie in high match regions. Since they do not improve the robot's self localisation accuracy, it is better to avoid using them. This is done by multiplying the match value by an off-field penalty o, where:

$$o = \exp\left(-0.02 \times d^2\right) \qquad \text{(Eq 3.21)}$$

where: d is distance from update position to the nearest on-field point.

The minimum match value v is calculated so that when the head is moving more quickly, the required match is less. This is because the image distortion caused by head movement would likely result in lower overall match values, while also supplying less accuracy.

$$v = 0.5 + \frac{0.3}{1 + 50 \times h} \qquad \text{(Eq 3.22)}$$

where: h is the head movement speed in μrad / frame.

To account for the lower accuracy as a result of head movement, all update variances are multiplied by a factor s:

$$s = 1 + \left(50 \times h\right)^2 \qquad \text{(Eq 3.23)}$$

So increased head movement increases the update variances, meaning the measurements are less trusted. Recall from section 3.1.3 that scanning for field points doesn't occur if the head is moving too fast, and so no updates are performed in this situation.

The update positions that pass the threshold in equation 3.20 are input into the kalman filter as a "direct" position update. Here the "direct" refers to the fact that the update, or observation, values are in the same space as the state values, and so the jacobian matrix used in the extended kalman filter correction is the identity matrix.

The new multi-modal kalman filter described in section 2.1 now comes into use. As a result of the symmetry generation, there are several update positions and orientations, and these correspond to a multi-modal observation distribution. To perform a multi-modal kalman correction update, each of the observation modes are treated separately from each other. The observation modes are applied to every original state mode individually, and a copy of the original modes is also kept and scaled by the noise scalar mentioned is section 2.2.3, so that if there are m state modes and n observation modes, the resulting number of state modes is $m(n+1)$. The resulting modes are kept in order of mode weight, so once the kalman state's mode array is full only the most likely modes will survive, the lower weight modes are discarded. This is the final step of the field line localisation using the gradient ascent method.

3.3.4 Gaussian Mode Sampling

An alternate method of field line self localisation, named the "mode sampling" method, was developed prior to completion of the gradient ascent method. This other method made use of the same point detection and match calculation functions described in sections 3.1 and 3.2, but applied the update in a completely different way.

Mode sampling examines each Gaussian mode of the kalman state and generates a set of 27 sample points arranged in a 3×3×3 regular grid around the mean of the original mode, i.e. the point at the centre of the grid is located at the mode's mean. The sample points in the grid are arranged to be one standard deviation away from each other along each of the 3 axes, and are aligned along the 3 basis directions for the global coordinate space, such as in figure 3.20. The sample point arrangement is an approximate representation of the one-variance volume of the Gaussian mode. Note that it does not approximate the Gaussian well if there are two or three highly correlated variances, i.e. if the mode is elongated on an axis diagonal to all basis vectors. A better approximation to the mode volume could be made by positioning the sample points along the mode's major and minor axes, however, calculating the coordinates of the sample points would then involve having to solve a third order eigenvalue problem, which would be too computationally expensive to perform for each mode.

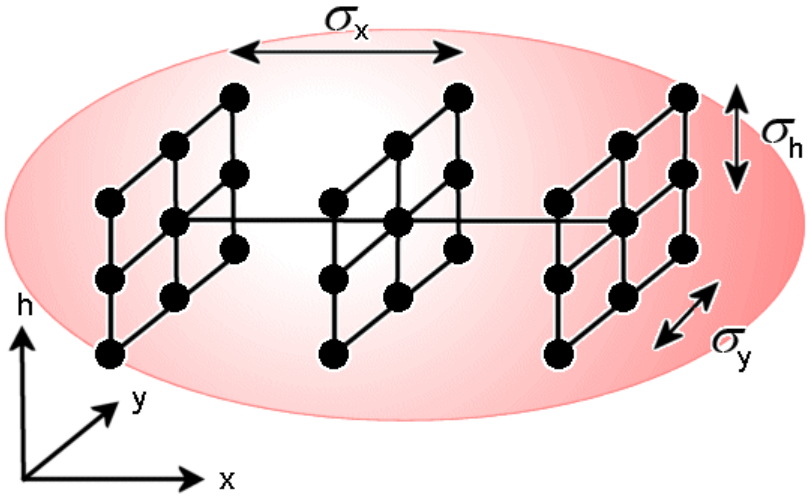

Figure 3.20: The arrangement of sample points. The lines joining the points are merely to make the diagram more clear, and hold no significance.

It is more convenient here to use a different coordinate system. Instead of the global coordinate system, sampling space coordinates are used, which have the origin at the mean of the current mode, and axes aligned with the global coordinate system, but scaled so that 1 unit length in sampling space corresponds to one standard deviation of the mode in that direction. Hence the coordinates of the sample points in sample space will be:

$$\underline{s} = \begin{pmatrix} i \\ j \\ k \end{pmatrix} \qquad \text{for} \qquad i, j, k \in \{-1, 0, 1\}$$

The match values at each of the sample points are then calculated. The actual update of each mode involves "reforming" it according to the match values at the sample points; the mean can be shifted, the variances altered, and the mode can even be split into new modes. The first step of reforming is to find the local maximums in the sampling grid. Here we define a sample point as being a local maximum if it has a higher match value than all the neighbouring sample points along the basis directions (i.e. not diagonal neighbours).

Each local maximum becomes a new mode, with the maximum point and its basis aligned neighbours determining the mean and variance of the mode. The new mode's mean and variance are calculated by treating the sample points along each basis axis separately as a one dimensional case. So along each basis direction, there will be either two or three match values from which to calculate a new one dimensional Gaussian distribution.

Consider the case of a one dimensional Gaussian distribution calculated from three equally spaced points with known function values:

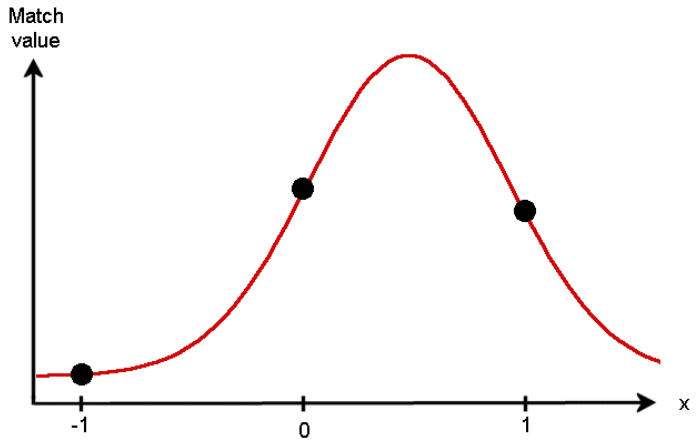

Figure 3.21: A One dimensional Gaussian with three known points.

The Gaussian distribution of unknown amplitude A, mean x_0, and variance α needs to be fitted to these three points:

$$M(x) = A\exp\left(\frac{-(x-x_0)^2}{2\alpha^2}\right) \qquad \text{(Eq 3.24)}$$

This can be solved to get values for α and x_0, while A is not important:

$$\alpha = \frac{1}{\sqrt{Q}} \qquad x_0 = \frac{1}{2} - \frac{R}{2Q} \qquad \text{(Eq 3.25)}$$

where: $\quad Q = \ln\left(\frac{m_0^2}{m_{-1}m_1}\right) \qquad R = \ln\left(\frac{m_0^2}{m_1^2}\right)$

Now consider the case where only two points are used to fit a one dimensional Gaussian. This would occur when the local maximum is not in the centre of the sampling grid, but on one of the edge points. Here, the fitting is solved for the case where the +1 sample point is the maximum, but the -1 case is very similar to derive:

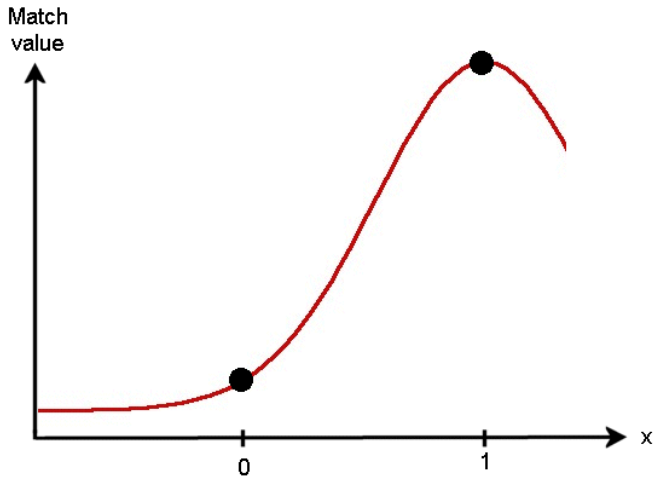

Figure 3.22: One dimensional Gaussian with two known points.

The equation is simplified by assuming the Gaussian's mean is at the point of the maximum, so only a value for α is required. Solving for α gives:

$$\alpha = \frac{1}{\sqrt{S}} \qquad x_0 = 1 \qquad \text{(Eq 3.26)}$$

where: $$S = \ln\left(\frac{m_1^2}{m_0^2}\right)$$

The mean and variance for each basis direction are calculated, and then converted from the sample space to the global space coordinates to get a newly formed Gaussian mode. This process is repeated for all the local maximums in the sampling grid. When all new Gaussians have been formed, they are applied as a direct kalman update to the single state mode that was reformed. The reforming and direct kalman update is then done for all other modes in the original state array.

The mode sampling localisation method has a few nice properties:

- It is applied to all modes in the self position mode array, unlike the gradient ascent which only gets applied to the most probable mode.
- It will automatically calculate appropriate variances for line information. So if the robot can only see a straight field line, its variance will spread out along that line.
- It will automatically split into separate modes if there is more than one possible location within the variance volume.
- Although accuracy is limited to the separation of the sampling grid, as more updates are applied, each mode's variance will shrink, and so the sampling grid will also shrink, increasing the accuracy when it is needed.

Despite all these positive aspects, the mode sampling method just didn't seem to work as expected. The robot's self localisation would jump almost randomly around the field, even with localisation help from the beacons and goals. It might be that the method's lack of support for non-axis aligned variances was causing problems. It might also be possible that the variances calculated for the update modes were far too small or too unstable. There could be a bug / bugs in the code. Whatever the reason, the mode sampling method failed to work, and so was abandoned when the gradient ascent method was found to work much better.

4 Behaviours

For the rUNSWift 2004 code, the Python interpreted programming language replaced C++ as the language used to control the top level "behaviour" of the robot (See [13]). As part of the change, all the existing behaviours had to be ported to the new language. Many of the fundamental skills were ported directly, i.e. a direct translation of the code from C to Python, while other skills and behaviours were redesigned and re-written as part of the porting. Two of the behaviours that were redesigned were the localisation challenger and the goalkeeper, not because they were bad, but because of the changed field environment and quality of sensor information available to them.

The localisation challenger, in particular, had more effective field line localisation available to it, while goal information which was available the year before would not be available now, as the goals were covered up during the challenge. For a full description of the challenge, consult "The Almost SLAM challenge" (Part 4) of [5]. Basically, the challenge involves making the robot move to five points on the field without help from normal beacons and goals, but with a new set of coloured landmarks posted at random positions around the field at the start of the challenge. The robot gets one minute at the start of the challenge with both the normal beacons and the new unknown beacons, to localise itself and memorise the positions of the new landmarks, then the regular beacons are taken away and the robot is transported to a different section of the field. It then gets two minutes to travel to the 5 points given to it in a file stored on its memory stick, and the robot that does this the fastest and most accurately wins the challenge (there is a scoring system for ranking purposes).

The goal keeper was also redesigned from last year's keeper. A couple of reasons for this were that the rUNSWift 2003 goalkeeper's localisation relied heavily on looking at the two central beacons, which were removed for the 2004 competition, and the kicks and special actions available to the robot had changed completely. This report will describe the initial simple implementation of the goalkeeper, but it should be noted that the system described here is only the base of the final code used in the 2004 Robocup tournament in Lisbon. Some code removal and many modifications were made by the team to this base code before competing.

4.1 Localisation Challenger

4.1.1 Detection of New Landmarks

The new unknown landmarks for the localisation challenge consisted of blank white A4 sheets of paper with other pieces of coloured paper stuck to them in various configurations. These A4 sheets were then attached to the outer barrier wall that was all around the field, with each sheet representing one landmark. As part of the rules, at least three of the landmarks had to contain a patch of pink at least 10cm across. Apart from this constraint there were few other limitations on the appearance of the landmarks; they could be of any colour other than black or white, including compositions of multiple colours, they could range in size from 10 to 50 centimetres, there would be at least six of them. It was decided that the localisation challenger would use only the pink patches from the new landmarks for two reasons:

- Pink was the only colour certain to be in the new landmarks.

- As part of the existing beacon localisation process, detection of pink blobs on the c-plane was already performed in the vision module, so this code could be used in the implementation.

In the vision module, a list of pink "half-beacons" is available. These half beacons are a list of areas that are formed by joining close-by solid pink blobs on the c-plane. The pink half-beacons that are close to blue or yellow half-beacons and pass certain sanity checks are used to form the beacon recognition in the vision module, but not all the pink half-beacons get joined into a full beacon. The landmark detector is basically a "pink detector" that takes these pink half-beacons, or pink areas, and makes sure they pass the following tests:

- Both dimensions (width & height) of the pink areas have to be greater than 4 pixels.
- The number of pink pixels in the area needs to be greater than 30.
- The density of pink, i.e. number of pink pixels divided by the size of the whole area, needs to be greater than 85%.
- The pink half-beacon must not be part of a regular beacon.

The first three tests are designed to eliminate unwanted noise caused by very small patches of pink in the background or on the field. The density check in particular is highly effective at removing most noise from the background, whilst the legitimate pink from the landmarks, which have very high density values, are left there. The final check is to stop the pink sections of the normal field beacons from being mapped as "new" pink landmarks.

The pink detector compiles a list of all the pink areas that pass these tests and calculates a robot relative heading to each of them, taking into account the area's c-plane position, and the robot's head angles. A distance to the pink areas cannot be calculated since the rules state that the pink patches will be at least 10cm across, not exactly 10cm across, so the size of each pink patch is unknown. This list of pink patches is passed to the localisation, or "gps", module.

4.1.2 Pink Mapping

The first part of the SLAM challenge is a period of one minute to calculate and remember the positions of the new landmarks around the field. The rUNSWift localisation challenger completed this first stage by following the schedule shown in figure 4.1, which has a duration of around 50 seconds.

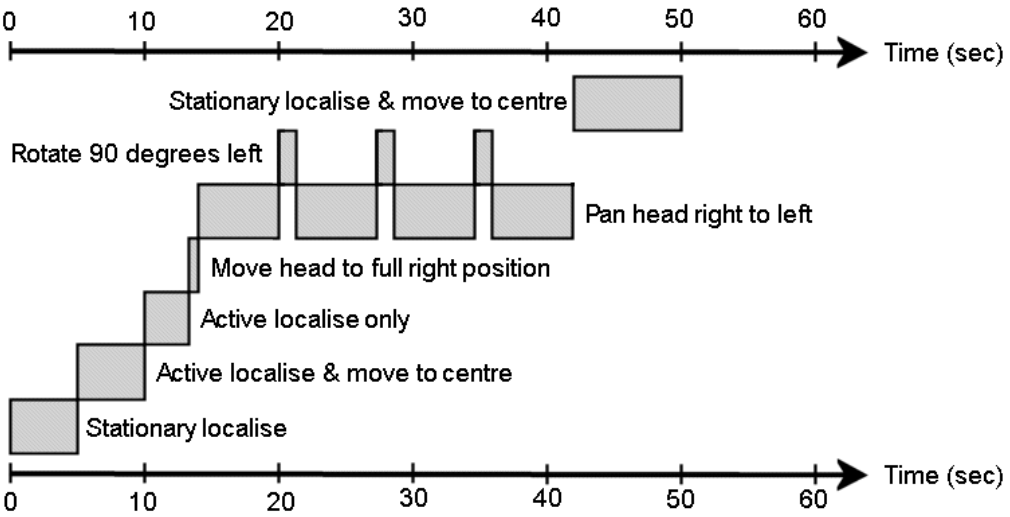

Figure 4.1: The mapping schedule.

All of the actual mapping is done in the four slow head pans when the dog is located in the centre of the field. When the pink mapping is enabled during these head pans, the localisation module takes the list of pink areas detected by the vision module, and using the robot-relative angle of each pink area, finds the intersection of the line cast out from the robot's current global position and heading estimate with the quadrilateral representing the outer wall of the Robocup field, as in figure 4.2. The outer wall quadrilateral needs to be defined by its corners' global positions, which were measured before the challenge took place, and entered into the code.

Doing this gives a global coordinate location and a distance to each of the pink areas, but also allows some extra checks to be performed on the pink areas:

- The elevation angle of the pink region is checked to make sure it is not too high or too low. The minimum allowed elevation is constant, but the maximum is inversely proportional to the distance, so that closer pink blobs are allowed to have a greater elevation angle, while further away blobs need to be closer to the horizon.
- The line cast in the pink blob's direction must not pass too close to the known global positions of any of the regular beacons. This is easily done by calculating the perpendicular distance from each beacon position to the line, and making sure it is more than 25cm.

The first check here makes sure that any unwanted background information that managed to pass the checks in the vision module will not be mapped. The second is a back-up check in case the pink from a regular beacon was not eliminated from the list obtained from the vision module (This can happen if the vision module failed to classify the regular beacon).

Once the global coordinates of each blob are found, the pink positions can be recorded. The recording process is actually quite similar to the "stationary mapping" described in 3.2.4, with a list of pink locations kept (this time in global coordinates), and a confidence value for each. This list is called the "pink map". Each pink location in the map gets a circular "merge region" of radius 25cm around it, and if any following pink locations

fall within the merge region, a weighted average (weighted by confidence, where any newly detected pink area has a confidence of 1) is performed, and the confidence value is increased by 1. Any newly detected pink areas that do not fall within a merge region are added to the map list, with a confidence of 1. An example of this process is shown in figure 4.3. Every 4 seconds, the confidence values of all locations in the pink map are decreased by one. This discourages any noise regions from entering the map.

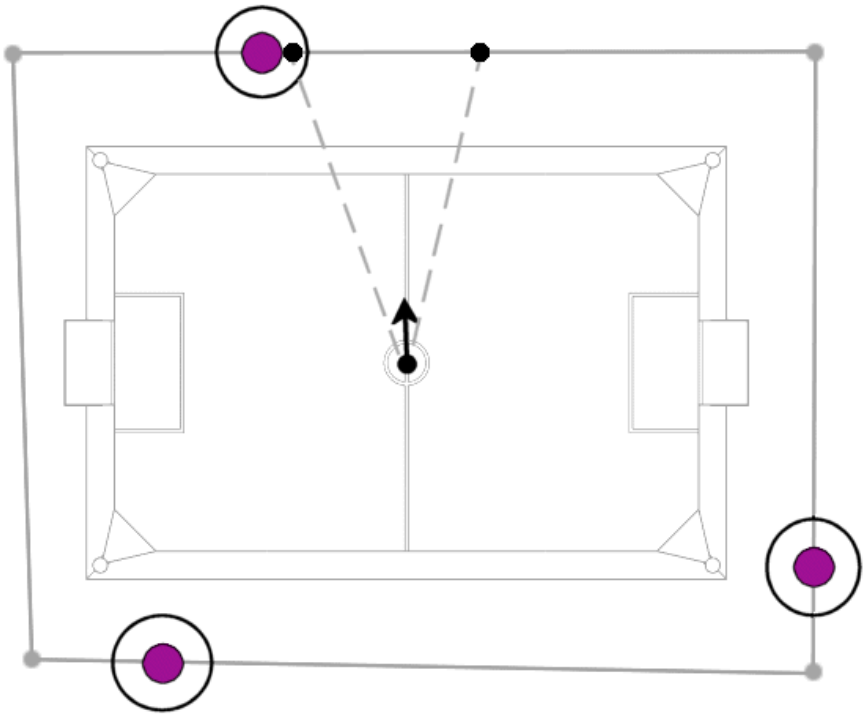

Figure 4.2: The robot has seen two pink blobs at angles shown by the dashed lines. The mapper calculates the intercept (shown by black dots) of these lines with the outer wall, which is defined by any four vertices (i.e. the outer wall doesn't have to be rectangular). One of the intercepts lies within the merge radius of an existing pink region (shown by the larger circles), and so is merged with it by taking a weighted average of their positions. The other intercept doesn't lie within a merge region, so will create a new region.

The pink mapping process proved to be quite accurate. It could detect the location of any pink in the extra landmarks to an accuracy usually within 15cm of the actual position. It did not suffer from any noise in the map, i.e. there were never extra pink regions in the map that didn't exist, or existed only in the background behind the barrier wall. If anything, the noise checks were a little too strict, since sometimes small, far away pink areas would not be present in the final map. Also, sometimes the pink map would contain two separate pink regions for the one landmark (but these were still close to the actual landmark position). This is thought to be caused by seeing the same landmark during two different head pans.

4.1.3 Localisation Using the Map

Near the end of the minute in the first stage of the SLAM challenge, the robot pauses itself and switches from "pink mapping" mode, to "pink localizing" mode. While the robot is paused, the regular beacons are removed from the field, and the goals are covered in white so they are indistinguishable. The robot is then moved to a random spot on the field, and the second stage of the challenge is begun when the robot is unpaused. While in the second stage

of the challenge, all localisation is done using the field lines and the pink map recorded in the first stage.

Localisation off the pink blobs seen is a similar process to localizing off the normal beacons. When localizing off beacons, a kalman correction update can be performed by using the global position of the beacon to calculate an expected angle and distance, which is compared to the angle and distance given in the visual observation (see section 3.2.7 of [7]). The pink updates are similar, but there is less information available. The global positions of all the pink landmarks are still available from the pink map. However, the pink blobs are indistinguishable from one another, so seeing a pink blob means that the blob could correspond to any of the recorded pink landmarks in the map. Fortunately, this is not a problem for the new multi-modal localisation system, which was constructed to handle these situations. Secondly, the observation of a pink area provides no distance information, so any updates using the pink are restricted to angle space only. This is also not an issue, as the beacon localisation already has code that implements a kalman update using only angle information.

The exact pink localisation process is as follows. Given the list of pink blobs from the vision module, the pink localiser checks all blobs and extracts the largest one (i.e. the blob with the greatest number of pink pixels). This will be the only pink blob used for localisation for that frame. It is unknown which pink landmark in the map that this blob refers to, so all landmarks in the map are used. Each mode in the self localisation state array is then split into N identical modes, where N is the number of pink landmarks in the map, and each of these modes are kalman updated separately under the assumption that the pink blob detected is actually a different one of the landmarks from the pink map. A copy of the old state mode is also kept, but scaled by the noise probability scalar from section 2.2.3, so any pink noise used for localisation wouldn't make a serious impact on the robot's position information. If the initial self location mode array contained M modes, and there were N pink landmarks in the map, the number of modes after updating would be $M(1 + N)$.

4.1.4 Global Reorientation

The most critical stage of the SLAM challenge is for the robot to be able to reorientate after it has been moved in the pause between the first and second stages of the challenge. This is the global localisation problem, where the robot must determine its position with no prior knowledge of where it is, which is significantly more difficult to solve than the local localisation problem, where the robot only needs to ensure that its localisation estimate is kept accurate by observation information.

At the beginning of the second stage of the slam challenge, the rUNSWift 2004 challenger attempted to re-orientate itself simply by spending 5 seconds standing still and performing several full left to right head pans, at two different tilt angles. This was enough for the robot to completely relocalise itself. It is important to understand what was happening in the localisation system during that period of apparent inactivity on the part of the robot.

By standing still while head panning, the robot activates the stationary mapping feature of the field line localisation. By doing full 180° head pans, the stationary mapper is able to build up a point map of a significant area in front, and to the sides, of the robot. The field localisation is then able to more precisely relocalise the robot than it normally would if it were moving. During the initial standing still period, the pink landmark localisation system

is also working, and this performs the duty of resolving any symmetries that are inherent in using only the field line information.

Before the challenge was run at the Robocup tournament, it was unclear whether this simple method would be sufficient for reliable global localisation, and so another method was developed, called the "One time edge localise". The method obtained its name from the fact it is very computationally expensive and so should only be called once, since it will cause the robot to "drop frames" (this is where the processor is spending all its time doing a particular task, and so doesn't have time to retrieve the frame data from the camera, leading to the robot missing video frame information). The idea behind the one time localise method is to perform the match gradient ascent (described in section 3.3.1) for many different starting positions and headings distributed across the field, in the hope of finding the maximum overall match value. After running all the gradient ascents, the one time localiser puts the highest match positions and orientations into the self location mode array, with each mode weighted according to its final match value after the ascent (overwriting any old modes that were present in the array).

For a method such as this to be effective, sufficient field information needs to be available, otherwise the match function in global coordinate space will feature many local maximums of similar weight, e.g. consider 10 border points arranged in a line segment 15cm long in front of the robot: there are many field positions where this would produce a maximum match value, and all the maximums would have about the same match value. This is where the stationary mapping facility is useful, it drastically cuts down the number of match maximums inside the global coordinate (position and heading) space, if the robot collects a wide variety of data (i.e. if the robot takes a good look around while standing still).

The strategy was, like before, to have the robot stand still and perform head pans after the second stage had just begun. After allowing some time (5 seconds) for the stationary mapper to build up a map, the one time localise would be executed, and the robot would continue from there. Unfortunately, this code was never used by the localisation challenger because it wasn't tested enough before the SLAM challenge was run at the Robocup tournament (the code was completed minutes before the deadline for submission of memory sticks for the challenge). It effectively has not been tested, and so it is unknown whether it works or not.

4.1.5 Movement Between Points

Once reorientated at the start of the second stage, the robot must move to the five pre-determined points as quickly and accurately as possible. The five points can be visited in any order, and so it is sensible to calculate the shortest path between them. There are heuristic searches, such as A*, which can supply a relatively short path quickly, however, considering there are only 5 points, it was decided to spend some extra time calculating and obtain the optimal path by an exhaustive search. This extra time is unnoticeable to any human observer of the challenger, since it is significantly less than one second.

It is assumed that the time taken to traverse a particular path is proportional to the total length of the path. This assumption is not always valid because the robot has to take time turning around to head for the next point, however it reasonably accurate in most cases. Once the distance of all possible paths have been calculated, and the shortest one chosen, the robot is free to move towards the first point on the path.

The method of movement used by the robot is significant. The prediction update of the kalman filter used for self localisation uses forward, left and turn values supplied from the actuator control module, as explained in section 3.2.6 of [7]. These values have been calibrated to match the walking speed of the robot, but the calibration is done so that the values only match the actual walking speed when only one of the forward, left or turn amounts is dominant, and the others are zero. When combinations of forward, left and turn are used at the same time by the robot's high level behaviour code, the actuator control assumes that the actual walking speed must then be a linear function of the 3 values, when in reality it is not. Hence, the odometry values sent to the prediction update can be wrong when a combination of forward, left and turn are used at the same time by the behaviour. Bad odometry can throw off the localisation of the robot, and because it is desirable to keep the self localisation accuracy as large as possible, the localisation challenger only tends to use one of the possible walk parameters (forward, left and turn) at any one time.

Hence, when the SLAM challenger needs to travel to a point, it will rotate on the spot till it is facing its destination, and then walk forward at maximum speed. If the robot needs to make small angular corrections while it is walking, then these are performed while walking forward, however if a large angular correction is needed, the robot will stop and rotate on the spot again till facing its destination.

4.1.6 Active Localisation

In order to keep the robot's position estimation as accurate as possible, the amount of useful information input into the localisation system must be maximised. To do this, the robot must actively move its head to observe field line features and landmarks on the barrier walls that will help resolve its position. It is not helpful for the robot to simply look down at the ground and hope that a line or border that it may localise off will appear in its view. Instead, the localisation challenger is programmed to keep watch on certain field features, shown in figure 4.3. These features are separated into two types, primary features and secondary features.

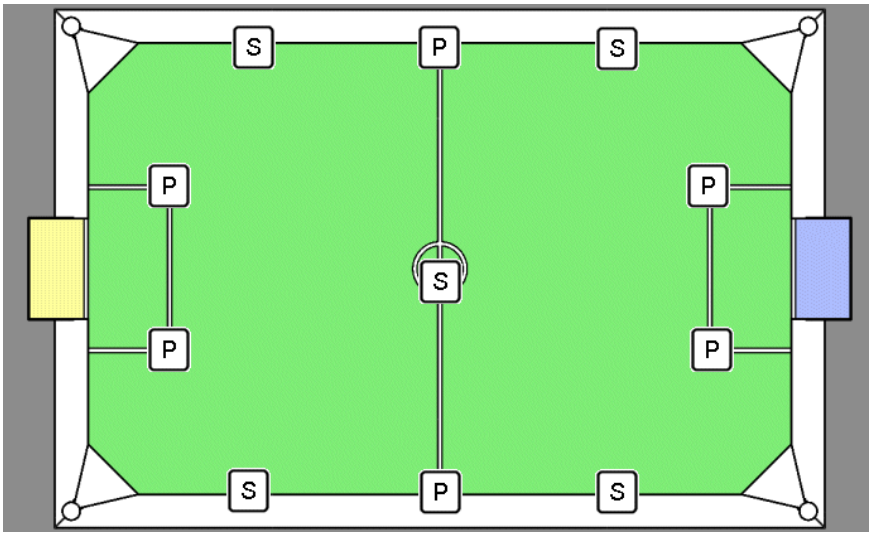

Figure 4.3: Field features. Features marked with P are primary, those marked with S are secondary.

The primary and secondary features are distributed evenly over the field, so that it almost all locations and orientations it will have at least one, and preferably two, features within its possible field of view. The primary feature locations correspond to points on the field that contain complex line markings such as corners or intersections. The robot will calculate, using its current position estimate, which field features (primary or secondary) are within its possible field of view by taking all features within 80° of its heading, greater than a distance of 15cm and less than a distance of 200cm. It will then set the head angles to observe each of the features by alternating between them every 40 frames (about 1.3 seconds). Periodically it will also look up to the horizon, so that it may see a pink landmark on the wall.

For each goal point the challenger has to move to, it chooses a primary field feature to face towards when near to the goal point, so that it is guaranteed to have a nearby feature to localise of for final positioning accuracy. The primary feature chosen is the closest one to the goal point that is more than 15cm away from it. When heading to a goal point, the robot does not actually head straight for the goal point, instead it aims for a point slightly behind it, along the line that passes through the goal point and the chosen field feature, such that the destination is on the opposite side of the goal point to the field feature, as in figure 4.4. This ensures that when it reaches that destination and turns towards the actual goal point, it is facing the chose field feature. This feature also helps avoid cases where the challenger performs wild rotations to keep it facing the chosen field feature as it approaches the goal point (for example, if it passes nearby the chosen field feature on its way to the goal point).

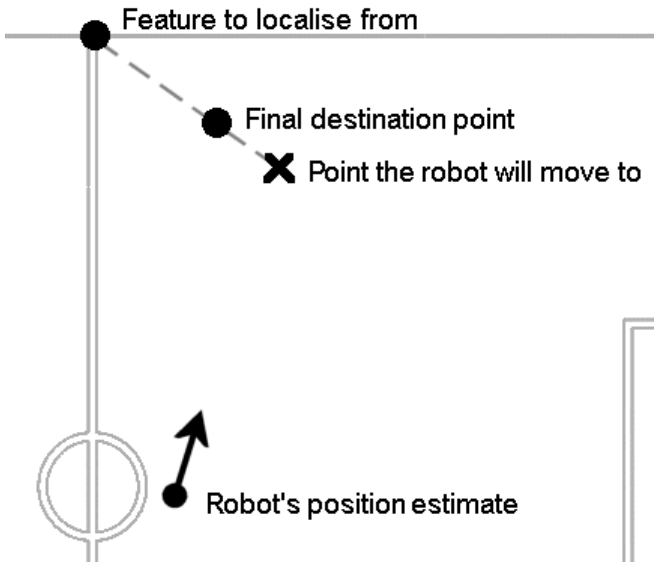

Figure 4.4: Destination point vs. the point the robot will actually move to.

4.2 Goalkeeper

4.2.1 Positioning

The aim for a goalkeeper is to form a barrier between the ball and the goal. Since it is the quickest route, most attacks at goal from the opposition will be in a straight line from the ball to the goal, so it would be sensible to position the goalie along the line from the ball to the goal. Although the goal is not a single field point but an area, we consider the goal position to be at the centre of the goal mouth. Positioning the goalie at the two extremes of this line, would have it sitting in the centre of the goalmouth all the time, or chasing after the ball, way out of the goal area. Neither of these possibilities would make an effective goalkeeper, so a compromise was chosen. The goalkeeper needs to stay quite near the goal mouth, so that it doesn't have to move too far if the ball is hit across-field, but it should move itself to a position in the goalmouth that is nearer to the ball.

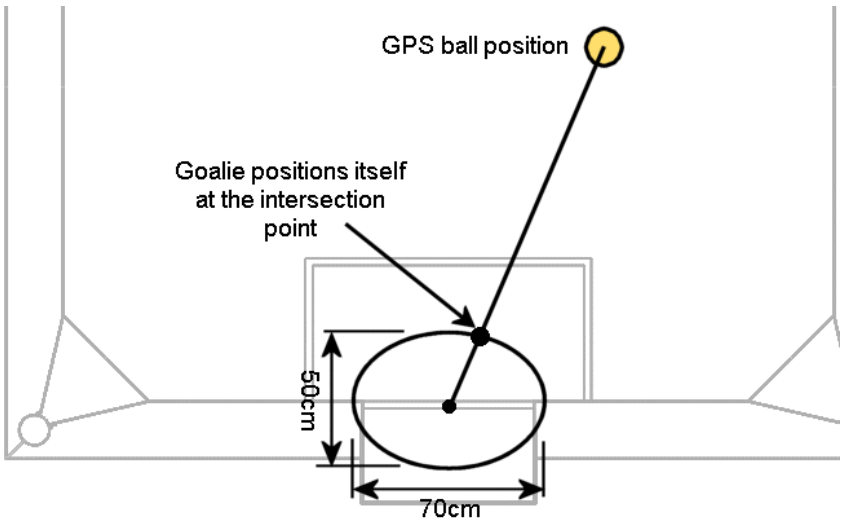

Figure 4.5: Goalkeeper position point.

The solution implemented in the original goalkeeper was to position the keeper at a point that is the intersection of the ball-goal line, and an ellipse that just surrounds the goal mouth, as shown in figure 4.5. The global ball position is obtained from the gps module, so the intersection point can be calculated in global coordinates. If the ball has not been seen for a while, the goalkeeper defaults to a position centred in the goal.

The robot's own position, taken from the gps module, is then compared to the intersection point, and if the Euclidean distance between the two points greater than a threshold, the robot will move towards the intersection point. This movement threshold, or "laziness" factor was added so that the goalkeeper would stand still when movement was unnecessary, since stopping occasionally provides better detection and distance estimates of field objects such as beacons and the ball, as well as conserving battery life and reducing heat build-up. Once the robot starts moving towards the intersection point, it stops once the Euclidean distance drops below another threshold, the "accuracy" factor, which is smaller than the laziness distance. This hysteresis is to stop the robot continually switching between moving and not-moving states.

As implemented, the goalie positioning was adequate, but not perfect. When defending from front on attacks, if the ball was hit hard enough at a gap to the side of the

goalie, the robot would not react fast enough to the changing intersection point, and would sometimes stand around while watching the ball roll past it into the goal. This situation wouldn't arise in a game very often however, since situations where an opposition robot has the time to set up a fast kick accurately directed towards the gap either side of the goalie are exceedingly rare. Most teams wouldn't even have a kick at their disposal that is fast and accurate enough for this.

Another problem was that when the ball was off to the side of the goal, the intersection point would not be in the ideal position, i.e. the goalie would be out of the goal, aligning itself along the barrier wall, when a better position is to be placed diagonally across the corner of the barrier wall, as shown in figure 4.6. Since the barrier wall is diagonally sloped it is possible that the opposition team can force the ball around the goalie by pushing it up the side of the barrier wall, while the diagonal positioning across the corner would not allow this.

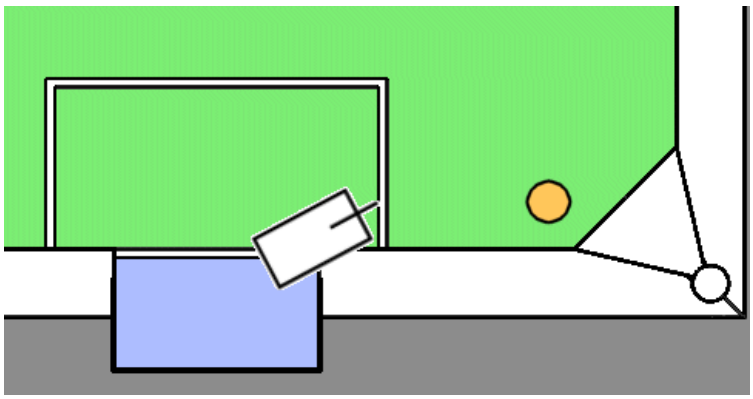

Figure 4.6: Ideal defensive position for balls off to the side.

4.2.2 Attacking

As well as forming a barrier for the goal from the ball, the goal keeper should also try to move the ball away from the goal area ("clear" the ball) whenever it gets the chance. So as to not compromise its role as a barrier, the goal keeper should only attack when it is sure that it can get to the ball first, or when clearing the ball doesn't require it to move substantially from the defence position. Because the original goal keeper was only a basic implementation, only the second of these conditions were used, i.e. when deciding whether to attack the ball the original goalie would not consider the positions or intentions of opponents or its teammates, it would only check whether the ball was in the immediate vicinity of the goal.

The attack mode would be triggered if the "gps ball", which is the global coordinates of the ball supplied by the localisation (gps) module, was found to be within the goal box, or if the visual ball, which is the ball information such as robot-relative distance and heading supplied by the vision module, was within 15cm. The two different triggers were used because the gps ball position is the result of a kalman filter process, and so can take some time to adjust to the ball's position if it is moving quickly. The visual ball, on the other hand, represents the most recent information of the ball from the current video frame, and so is better suited to reacting to fast moving balls.

Upon activation of the attack, the goalie would perform a "paw-kick", which involves running towards the side of the ball, such that one of the robot's front legs aligns with the centre of the ball. As part of its running action, the aligned leg will take a step forward, and

the ball will be kicked as part of that forward step. To break off the attack, one of two conditions must be satisfied. One trigger is that the ball is more than 15cm away from the nearest point inside the goal box, which would signify the goalie has successfully cleared the ball, and so can return to ordinary defending. The other trigger activates if the visual ball is at an angle of greater than 40° from the straight ahead direction of the robot, which would mean that the ball is no longer in a suitable position for the robot to perform a paw-kick, so the attack has failed and the goalie should retreat.

In the game situation, the goalie attacks were not successful mainly due to the type of kick used. If the ball was approaching head on, with no opposition robots behind it, then the clearance kick would work, however most goal attacks come from the sides with an opposition robot directly behind the ball, pushing it towards goal. In this case the goalie would attack with the paw kick, basically running continuously at the ball, try to push it, and the opposition robot behind it away from the goal. This scrum situation, together with the fact that not much localisation information would be available since the goalie would be focusing on the nearby ball, would usually cause the goalie to become mis-localised, and it would continue chasing the ball even if it were cleared from the goal box area, eventually leaving the goalkeeper well out of position.

The attacking was later improved with the use of a different kick, the side-swipe or "U-Penn" kick, which allowed the goalie to avoid many scrums with the opposition, however this was not part of the simple original goalkeeper implementation.

4.2.3 Head Control

The goalkeeper's head really only has two tasks. One is to track the ball, the other is to look at objects that will help its self localisation. Head control in the original goalkeeper involved tracking the ball, and occasionally looking up to the field beacons (active localisation). The trigger to active localise was time based, and would only fire if the ball was greater than 25cm away. Once done active localizing, the head would return to the angular position of where it had last seen the ball. If the ball is not seen in this position, the robot would look at the field position where the gps ball was located. If the ball is still not seen after this, then a ball search is initiated, which involves panning the head from full left to full right positions at a head tilt that enables the horizon to just be seen at the top of the c-plane, and then a quick pan at a tilt angle such that the goalkeeper can see any balls at its feet, or below the head.

The active localisation method used by the goalkeeper is slightly modified from the one used by the forwards. The active localisation skill will choose a beacon to localise off based on the robot's self position covariance, i.e. if the robot is unsure of its position in a certain direction, the active localise will choose a beacon in that direction to give it the most useful information. Normally, the beacons considered for selection by the active localiser are only the ones in front of the robot, as in figure 4.7; this is to ensure that the robot does not try to view a beacon that may be behind it, which is impossible. However, this is not always good for the goalkeeper's localisation, since when it is facing forward it can mean that the two closest beacons to it are always excluded from selection, and so the goalie only localises off the two far beacons, which is less accurate. To make sure the close-by beacons are used, every second active localisation will consider beacons in a wider angular range about the robot, as shown in figure 4.7.

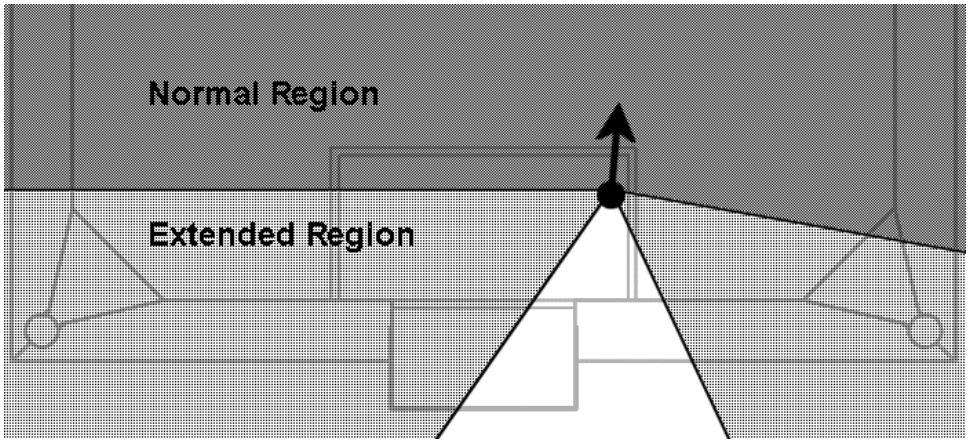

Figure 4.7: Beacons considered when active localizing.

This simple head control method proved effective at keeping track of the ball and keeping the goalkeeper localised whenever defending. When the goalkeeper decided to attack however, the goalkeeper sometimes didn't active localise for long periods of time, due to the localise trigger's conditions, causing it to get lost during the attack. This was more of a problem in the attack method than the head control, though.

4.2.4 Rear Wall Avoidance

Due to the large amount of time the goalkeeper spends near the edge of its own goal, there are often times when it tries to move somewhere, but gets stuck on the edge of the rear barrier wall to either side of the goal. For example, after the goalie has moved out of goal to clear a ball and is returning to its defence position is a usual case for this to occur. Getting stuck on the rear barrier wall usually involves the robot trying to turn in the direction of its destination and moving backwards at the same time, but unable to because one of its legs is being blocked by the wall. To help avoid this happening, a movement method was devised that cancelled all movement in a direction towards the rear of the field if it was likely that one of the legs would hit the rear wall.

To determine if a leg is going to hit the rear wall, a simple model of the leg positions, as shown in figure 4.8a), is used. This model assumes the legs are placed at the corners of a rectangle, with the centre of the top edge of the rectangle assumed to be the position of the robot's neck, i.e. this point is the local coordinate system origin. The rectangle corners are then transformed to global coordinate space by rotation and translation, and if any of the transformed legs lie inside an area that is 2cm out from the rear wall, as shown in figure 4.8b), then any movement in the global y direction is cancelled, by only taking the x component of the movement.

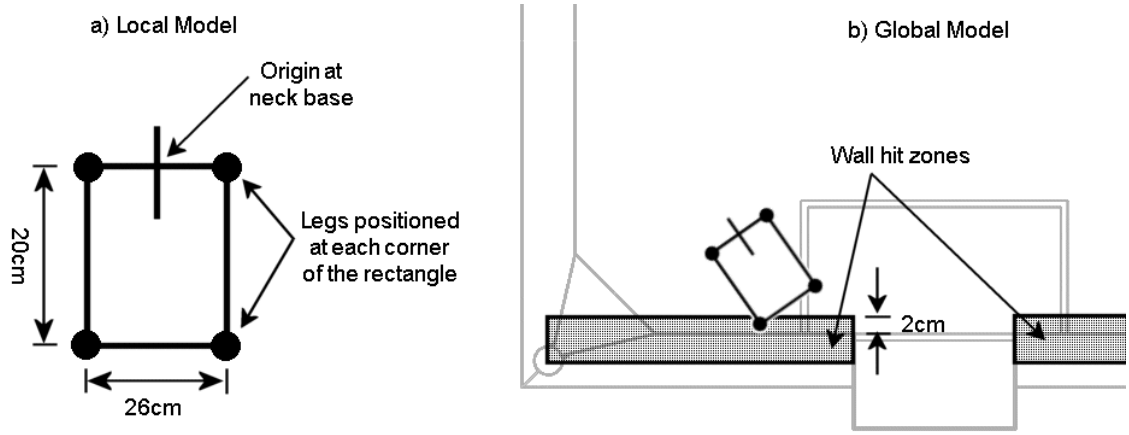

Figure 4.8: a) Local and b) Global collision model of the rear wall avoidance system. The four leg points from the local model are rotated and translated to their global Positions. If any of the points lie within the "wall hit zones" on either side of the Goal, the robot's movement in the global y-direction is clipped.

5 Conclusion

5.1 Results and Future Work

Overall, the self localisation system used by the rUNSWift 2004 team was very accurate, and made good use of all localisation information available. For a robot on its own in the field, the system has proved itself to be reliable and not too computationally expensive, as proven in the localisation challenge, where it easily claimed first place, reaching four of the five destination points, when the closest other team only reached two. It was the only localisation challenger that gave the impression of actually knowing where it was. This is not to say that the system is flawless.

The single biggest problem with the system at the moment is its reaction to scrum or restricted movement situations for the robot, i.e. when the robot is hitting another robot or a wall while trying to walk somewhere. After this occurs, the field line localisation can sometimes work against the system, keeping the robot's self variance small by localizing on some local maximum that doesn't represent the robot's actual position. Since the variance is kept small, the robot's higher level behaviour never knows to perform an active localise, and the robot will continue dribbling the ball, sometimes in completely the wrong direction, towards its own goal.

Another lesser problem with the system is the computational time it takes to perform the gradient ascent. This is of lesser importance because it doesn't make a critical difference to the localisation accuracy, and because the single step gradient ascent method used at the moment is very crude and simple, its speed can be readily increased by using more efficient methods. At the present, the computational time consumed is not enough to make the robot "drop frames".

Possible improvements that should be considered are:

- Improvement of the stuck detection. Currently the stuck detection is in a very primitive and unreliable state. With a proper analysis and testing, it is probable that the stuck detection could be significantly improved, thus providing better motion updates in scrum situations. If the stuck detection were sufficiently reliable, the field line localisation system could be disabled for a short time after the stuck condition, this way the self variance could grow for some time, and the behaviours would know to perform an active localisation. The field position could be repaired using beacon information alone, then the line localisation could be re-enabled.
- Combining field line localisation and beacon / goal localisation into a single kalman update. Currently the two systems run separately, and apply two different observation updates to the kalman filter. Combining them into one update may help stop the two systems conflicting with one another.
- Implementing a more efficient match maximization method than the current gradient ascent. The conjugate gradient method, for example, could be more efficient than the current method. If the efficiency were improved enough, it might be possible to perform the match maximization on several of the Gaussian modes in the distribution, not just the highest probability one.

5.2 References

[1] The Official RoboCup Website
http://www.robocup.org/

[2] RoboCup Legged Robot League Website
http://www.openr.org/robocup/

[3] AIBO Software Development Environment Website
http://openr.aibo.com/

[4] Sony Four Legged Robot Football League Rule Book
http://www.tzi.de/~roefer/Rules2004/Rules2004.pdf

[5] Technical Challenges for the RoboCup 2004 Legged League Competition
http://www.tzi.de/~roefer/Rules2004/challenges2004.pdf

[6] Team rUNSWift Website
http://www.cse.unsw.edu.au/~robocup/

[7] "Rise of the AIBOs III – AIBO Revolutions" – rUNSWift 2003 Report
by J. Chen, E. Chung, R. Edwards and N. Wong.
http://www.cse.unsw.edu.au/~robocup/report2003.pdf

[8] "Visual Feature Detection for Robotic Soccer"
by R. Sheh.
http://thesis.ece.curtin.edu.au/Thesis_2003/Raymond%20Sheh%20-%2009922721/

[9] "Thesis B Report - The Sony Legged Robot League"
by D. Lam
rUNSWift 2004 research papers.

[10] "Thesis B Report - The Sony Legged Robot League"
by J.Xu
rUNSWift 2004 research papers.

[11] "The Sony Legged Robot League – COMP 4911 Thesis B Report"
by T.Wong
rUNSWift 2004 research papers.

[12] "Thesis B Report – rUNSWift 2004 The University of N.S.W. 2004 RoboCup Team"
by C. K. Chan
rUNSWift 2004 research papers.

[13] "RoboCup Project Report"
by K. Pham
rUNSWift 2004 research papers.

[14] "An Introduction to the Kalman Filter"
by G. Welch and G. Bishop
http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf

[15] "Active Global Localisation for a Mobile Robot Using Multiple Hypothesis Tracking"
by P. Jensfelt and S. Kristensen.
IEEE Transactions on Robotics and Automation, Oct 2001, vol 17, no 5, pp 748-760.

[16] "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots"
by D. Fox, W. Burgard, F. Dellaert and S. Thrun.
Proceedings of the National Conference on Artificial Intelligence, 1999.

[17] "Fast and Robust Edge-Based Localization in the Sony Four-Legged Robot League"
by T. Röfer and M. Jüngel.
7th International Workshop on RoboCup 2003

[18] "Summer Research 2004 Technical Report"
by K. Pham
rUNSWift 2004 research papers.