THE UNIVERSITY OF NEW SOUTH WALES

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Robot Localisation Using a Distributed Multi-Modal Kalman Filter, and Friends

Oleg Sushkov

Thesis submitted as a requirement for the degree

Bachelor of Science (Computer Science)

Submitted: September 8, 2006

Supervisor: William Uther
Assessor: Professor Claude Sammut

Abstract

Any system for robot localisation needs to be accurate, resilient to false observations, and computationally efficient. In the domain of robot soccer, the prevalent methods of localisation previously included grid based localisation techniques, and more recently, Monte Carlo particle based methods. The problem with grid based localisation techniques is the discretisation of the state space, with higher levels of detail requiring increased memory and processing time. Particle based methods have been quite successful in the robocup domain, but they have difficulty with high dimensional state spaces, since this requires an exponentially increasing number of particles with the number of dimensions. This report presents a localisation method based on the Kalman-Bucy state estimation filter. We build on top of the basic algorithm, adding multiple-hypothesis tracking, distributed teammate localisation, and a way to find the optimal constant parameters for the localisation system. These changes allow our robots to be able to localise themselves as well as the ball effectively and quickly. The specific domain to which we apply our algorithm to is the Robocup Four Legged League. The biggest advantage of our system is the fact that the ball acts as a mobile beacon, allowing a robot looking at the ball to be able to correct its position hypothesis. This in effect makes our algorithm a form of Distributed SLAM.

Acknowledgements

Firstly I would like to thank my fellow teammates, Andy, Eric, Michael, and Ryan. Working with you guys for 6 months of this year has been a real pleasure, and I feel very lucky to have been given the opportunity to be part of such a great team. Will, thank you for giving us your guidance, your experience, and your support. Most of all, thank you for having confidence in us at all times, maybe even when we could barely score against an empty goal a week out from the Australian Open. The lessons that you taught us this year have been invaluable and I feel privileged to have you as my thesis supervisor. Claude, thank you for teaching us that being good is not good enough, only being the best is good enough. Thank you Brad for being our Mother for these 6 months, allowing us to concentrate on our work and not have to worry about anything else. Thank you also for trying your best to get the other teams drunk at the competition, and also getting us drunk afterwards. Thanks to CSE and NICTA for funding and supporting the rUNSWift team all of these years. Finally, I want to thank my friends and most of all my parents for being understanding, supporting, and loving through what was probably the most difficult and challenging endeavour I have ever undertaken.

Contents

Chapter 1

Introduction

1.1 Robot localisation

Knowing where you are is extremely important for any mobile robotics system. If you do not know where you are or where the objects around you are, it is very difficult to perform any meaningful actions. The main obstacle to efficient and accurate robot localisation is noise. Noise is present in every part of a robotics system, both in the sensors as well as in the actuators. In a world without noise, with perfect sensors and actuators, localisation would be a relatively simple task.

It is important to note that localisation is not necessarily restricted to determining the pose of a robot, but can also include tracking the state of other objects. Taking the robocup domain as an example, this could include the ball position and velocity and teammate tobot poses. In this case, the system is essentially localising the world state in a multi-dimensional state space.

The core concept of robot localisation is estimating the world state through sensor data. In most situtations, the world state is not directly observable in its entirety, and must be inferred from the given sensor data, and integrated over time. We say that the world is partially observeable. The different algorithms for robot localisation provide varying ways on incorporating the partial observations of the world state into an internal representation of the complete world state.

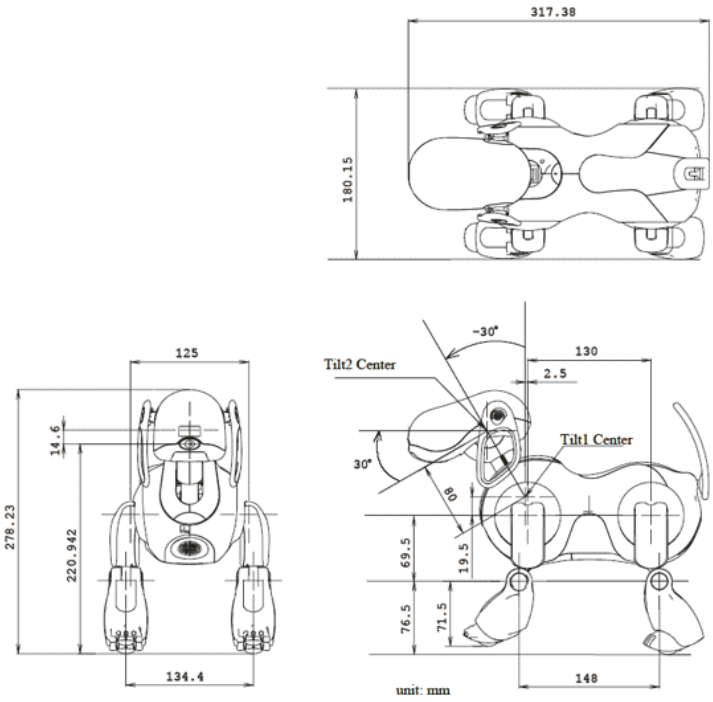

Figure 1.1: Schematic Diagram of the Sony ERS7 AIBO Entertainment Robot

1.2 Application Domain

The specific domain to which we are applying robot localisation to is the 4-legged league of Robocup[1]. Robocup is an international organisation which seeks to foster research in robotics by providing a standard performance measure, soccer. Playing soccer is a very difficult problem for autonomous robots, involving accurate sensing of the world, reliable and fast locomotion, as well as multi-agent cooperation and planning. By providing a level and competitive playing field it is possible to directly compare the performance of various systems and algorithms. Robocup consists of many leagues, each with variations in terms of the hardware used, the rules, and overall system focus. The 4-legged league[1] is unique in that it has a standardised hardware platform, the Sony Aibo entertainment robot. The 4-legged league field measures 6mx4m, has 4 unique coloured beacons placed around the edge of the field, 2 coloured goals at either end, and white field lines painted on the field. We can use all of this information in order to localise the robot, using the beacons, the goals, as well as the field lines, to determine the position and

[1]http://www.tzi.de/4legged/

Figure 1.2: A typical game of the 4-legged league, notice the coloured Goal and Beacons

heading of the robot.

Another key object on the field is the ball. We must be able to track the balls position as well as velocity to be able to chase the ball effectively and control it once the robot is close. As such, an effective localisation system must be able to track the robot pose (x and y position, as well as the angle of orientation on the fiel), as well as the ball x and y position and velocity. This is important because if the robot does not know where it is on the field, even if it has control of the ball, it will not be able to do anything effective. In addition to this, if the robot cannot currently see the ball, if it has some idea of where it is on the field, this information would significantly reduce the time required to re-aquire the ball.

1.3 State of the Art

The currently preferred methods of localisation in the 4-legged robocup league include Monte Carlo Particle Filters, and uni-modal Kalman filters. By far the most popular of these is the particle filter method[7]. The reasons for the popularity of particle filters[11] is their ability to handle non-linear observations (where the mapping from observation space to state space is a non-linear function), their quick convergence, and their multi-modal nature (being able to track

many different possible positions at the same time, termed Global Localization). The other popular method of robot localisation is the Kalman Filter[5]. This is a state estimation filter which uses a Gaussian to approximate the probability density function. The main advantage of this method is that it is very computationally efficient, being able to compute the updates in closed form.

Particle filter localisation works by keeping track of many discrete hypotheses of the state of the world, or particles. At each time step, each particle is updated based on the available odometry information. When an observation is made, the hypotheses which more closely match the observation are given higher weights, while the ones which disagree with the observation are given lower weights. The particle set is "'resampled"' after each observation, with the higher weight particles being more likely to be included in the new set. The mean robot position is estimated to be at the point of highest particle density. The main problem with the particle filter approach to localisation in the Robocup domain is the computational demands of the algorithm, since many particles are needed to approximate sufficiently accurately the true probability distribution. A related issue is the fact that in general, the number of particles needed to represent this probability distribution grows exponentially with the dimensionality of the state space.

1.4 Multiple Hypothesis Tracking Kalman Filter

We propose a system for robot localisation which is a hybrid of the Extended Kalman Filter and the Monte-Carlo particle filter. In essense, have a cloud of particles, where every particle is a Gaussian, and the weighted sum of these Gaussians forms an approximation for the underlying probability distribution for the state space. We apply observation and motion updates to each Gaussian particle in the same way as for a standard Kalman filter. Each Gaussian particle is then weighted according to how well the observation matched the hypothesis. This method combines the advantages of both the particle filter method and the Kalman filter method. We are able to approximate arbitrary probability distributions (given enough Gaussian particles), handle high dimensional state spaces, ambiguous observations, and at the same time keep computational costs reasonable.

1.5 High Level Behaviours

In addition to the work on the localisation module, we made several major contributions to the high level behaviours of our system. We concentrated mainly on perfecting the general game-play style introduced in 2005[4], which is generally 1)find the ball, 2)grab the ball, 3)turn towards the goal, 4)dodge obstacles, 5)shoot on goal. I concentrated on the first two points, as well as the support behaviour of the teammate robots. Being able to find the ball as quickly as possible is one of the most crucial skills, because if you can re-aquire the ball faster than the opponent, then you can get to the ball first and do something with it. At the competition, the efficiency of our ball tracking and searching skill was able to counter-balance the relatively low walking speed of our robots, so in effect we were more often than not the first to the ball.

Being able to grab the ball, that is catch the ball under the robots chin, quickly and reliably is an equally important skill. This is particularly true for our gameplay style, which relies on grabbing to be able to control the ball. Last year our grab was quite reliable but was at times slow and hesitant[4], which was one of the main factors contributing to our loss against Newcastle in the semi-finals in 2005. Due to the underlying improvements in our localisation and tracking modules, we have improved the accuracy of the ball position and velocity tracking, resulting in much greater reliability and speed of the grab.

The teamplay aspects of our system are an extremely important part of the overall performance. To this end, we spent alot of time on the positioning of the supporter and defender teammates, as well as the behaviour of the robots as they walk to their positions. This also included the policy for role switching, that is, deciding which of the robots will be the one to attack the ball, and which ones will fall back.

1.6 Report Overview

In this report we discuss the problems associated with efficient and robust robot localisation as applied to the 4-legged league. In addition to this we also cover the improvements made to the high level behaviours of the rUNSWift codebase. Chapter 3 covers in depth the theories behind robot localisaton, the basic Kalman Filter approach, and the way in which we significantly

improved upon the basic algorithm by combining the Kalman Filter approach with the particle filter method. In Chapter 4 we provide a broad overview of the improvements made to the various behaviours. Finally, Chapter 5 gives a summary of the work as well as suggestions for future work.

Chapter 2

Background

2.1 Methods of Evaluation

We have made considerable changes to our localisation system this year, and as such it was fairly important to evaluate the changes thoroughly in order to confirm that the changes increase the performance characteristics of the system. However, this raises the question of what exactly defines good performance for a localisation system in the Robocup domain. Accuracy in an artificial test is not the only criteria, the system must also perform well in a real competition game. To this end, our testing methods included specific artificial tests which target only the part of the system which is being scrutinised, and also we run full soccer games between new code and old code at regular intervals to make sure that a feature which performs well in isolation also performs well as part of the system as a whole.

For specific testing of the localisation module we used a series of waypoints placed on the soccer field and had the robot run between the waypoints, stopping at each one in series. We can then measure at each waypoint the difference between where the robot thinks it is and where it is located in reality. We gather this data on multiple runs to reduce the impact of noise, and with different field configurations, ie: removing certain beacons and goals, or placing teammate robots and a ball on the field. This form of testing gives concrete error estimates which allow us to determine if the accuracy of the localisation system has improved. This will be the first time such concrete testing has been done on the rUNSWift localisation system.

The other module which we were able to test in isolation effectively was the grab. A robot

must be able to perform a grab both quickly and reliably. We were able to test the reliability of the grab relatively easily, by recording the percentage of successful grabs made on a number of test runs. Following the competition we were also able to review game footage and record the percentage of successful grabs.

Most of the other modules were extremely difficult to test effectively. For example, to test the effectiveness of a new Supporter positioning scheme one must play many practice matches between old and new code. This number of games must be large enough to account for the stochastic nature of a Robocup game. Unfortunately, this was not a practical task for us to carry out, and as such we relied more on ad-hoc estimations for the performance measure of such modules.

2.2 Review of Existing Work

Much of the localisation system described in this report is built upon the foundations created in previous years by rUNSWift team members. The localisation system has undergone several major revisions, gradually improving and become less and less ad-hoc and more theoretically sound.

In 2003 the localisation system was largely re-written[2], and it was now based on the full Kalman Filter state estimation algorithm. This was a very effective system for the time, but it nonetheless suffered from some problems. Firstly, it used separate Kalman Filters to track the ball and the robot poses, this resulted in poorer estimates of the ball position and velocity due to not being able to use the correlation between the robot pose and ball position in observations in the filter. In addition to this, the system could only handle uni-modal distributions and could not deal with ambiguous observations.

In 2004 the rUNSWift localisation system was built upon by Derrick Whaite[3]. The system was made multi-modal, with the probability distribution now being represented by a weighted sum of Gaussians rather than a single Gaussian. This allowed the system to deal with ambiguous and noisy observations much more effectively. Despite these improvements the state was still represented by the robot pose, with the ball being tracked by a separate Kalman Filter. In addition to this, the full power of the multi-modal representation of the probability distribution function

was not utilised. The multiple modes were mainly used for filtering out noisy observations and also for ambiguous field line observations. In addition to these two uses, it is possible to use the multiple-modes to more accurately approximate the true probability distribution and to reduce the inherent error from the linearisation of the state space to observation space function.

The Newcastle team is one of the teams which has taken a very similar approach to ours to localisation in the 4-legged league. One of their major advantages over our previous system was their coupling of robot pose and ball position and velocity into a single mean vector[8]. This allowed them to be able to track the ball very well, and in turn flowed on to enabling improvements in high level behaviours such as the grab. Their success in the area of ball position and velocity tracking showed us that this was a path worth pursuing.

There are many other possible approaches to robot localisation. Most of the teams in the 4-legged league now use Monte-Carlo Localisation, or particle filter localisation[7]. In 2004 10 teams used this approach out of 17 classified teams. The German Team used this algorithm to good effect in 2004 and 2005[10]. The reason for the popularity of this algorithm has been due to its ability to model arbitrary probability distribution functions, not being restricted to a fixed closed form respresentation. This makes it particularly suitable for localising from field line observations, which has become more and more important with the reduction in the number of beacons on the field. The German Team uses two separate particle filters, one for the robot pose, and one for the ball position and velocity. Despite the advantages offered by the particle filter approach, the downsides include the need for a large number of particles to properly model a probability distribution function, which in turn increases the runtime for the algorithm, a limited resource on the AIBOs. We decided that we could realise the advantages of a particle system, while avoiding the downsides, by modifying the existing localisation framework.

Chapter 3

Multi-Hypothesis Tracking for Robot Localisation

The task of robot localisation can be seen as calculating the belief distribution of the robot position with given observations and control updates. Measurement updates are observations of landmarks, in our case distance and heading measurements to the coloured beacons. Control updates are a prediction of the belief state after the robot performs some movement. At their core, almost all robot localisation algorithms are approximations to the general Bayes Filter algorithm. In this chapter, we discuss the general Bayes Filter algorithm, the assumptions it makes, followed by a detailed description of the localisation algorithm which we developed as an approximation to the Bayes Filter.

3.1 Bayes Filter

The Bayes filter is the most general form of deriving beliefs. A belief in this case is a function from states to probabilities. Using this function we can determine the most probable state of the world. The belief state is calculated from measurement and control updates. The general algorithm for calculating the belief at time t is given below.

$$\overline{bel(x_t)} = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$$

$$bel(x_t) = \eta p(z_t|x_t) \overline{bel(x_t)}$$

The first line in the above formula is calculating the belief state based on the control data, that is using the commands sent to the actuators to hypothesise a new belief. The second line is using the measurement data to update the belief state. In the above formula, x_t is the state at time t, u_t is the control update at time t, and $p(z_t|x_t)$ is the probability of an observation z given the world state x at time t.

In practice, we can only implement an approximation to the Bayes filter. This is because it is impossible to represent general belief and measurement probability functions in closed form. Instead either non-paramteric filters are used, such as particle filters or histogram filters, which try to approximate the function with descrete samples, or we can use given closed form functions such as Gaussians to approximate the belief and measurement probability functions.

One important assumption of the general Bayes filter is the Markov assumption. The Markov assumption states that past and future states are independant if the current state x_t is known. An example of a Markov system is a pendulum, if current state is the position and velocity of the pendulum, we have all the required state information to predict future states. However, if in the case of the pendulum, the current state was only described by the position of the pendulum then the system becomes non-Markov, because knowing only the position of the pendulum at time t is insufficient information to be able to predict any future states. In almost all real world robotics systems the Markov assumption does not hold, but the violations are usually small enough that Bayes filters end up being quite robust to them.

3.1.1 Localisation Problems

The task of robot localisation can vary in difficulty depending on various factors. Each of these factors will influence the choice of algorithm used for the particular situation, since some algorithms are adept at handling certain classes of problems but not others.

We can categorise localisation problems based on the type of measurements available and the knowledge of the initial state of the system. Based on these we can classify the problem into either Local or Global localisation. In the case of Local localisation, the probability distribution function has only a single mode, meaning that the localisation algorithm is only required to deal with a small pose error, with the uncertainty confined to a small region around the robot's true

pose. In the case of Global localisation, the probability distribution function needs to be able to handle multiple modes, and the algorithm used must be able to handle high uncertainty in the robot pose. This may be due to large errors in robot motion and measurements, or due to the presense of non-unique landmarks. For example, in the situation where there are two identical rooms connected by a corridor, the localisation algorithm must be able handle a probability distribution function which has a mode in each room.

The "'kidnapped robot"' problem is related to the problem of Global localisation. This issue arises if at some point in time the robot is taken from its current location and placed in a completely different location. In the case of the Robocup domain, this issue is especially prevalent, since robots are frequently taken off the field or placed back on the field in a new location. It is very important that the chosen algorithm can deal with "'kidnapping"' quickly and efficiently. This problem is magnified even further if the place where the robot is replaced has almost symmetrical landmarks when compared to its belief location.

Whether the environment is static or dynamic will also affect the choice of suitable algorithm. In the case of a static environment, the objects which are of concern for the robot are stationary during the operating cycle. A dynamic environment, however, may have moving objects which can affect the robot, or which the robot must interact with. In which case the algorithm must track not only the robot pose, but also the position of the moving objects. When applied to the Robocup domain, we can clearly see that it is a dynamic environment. Specifically, the orange ball is a moving object which we must track. The teammate robot and opponent robots may also be considered as dynamic objects, but often, as in our case, they are ignored because they are very hard to observe.

The final distinguishing feature of robot localisation that we will consider is the issue of single-robot and multi-robot localisation. The simple case is single-robot localisation, in which case observations are all made by the one robot, only one pose needs to be tracked, and there is no need for communication between robots. Multi-robot localisation offers the advantage of the availability of a greater number of sensors and thus a greater number of observations. We can use this increased observational power to improve the accuracy of the system as a whole. However,

multi-robot localisation also brings with it many challenges. The localisation algorithm used must be able to incorporate the observations of other robots in the system, which in itself brings about issues such as communication lag. In our case, we can use the observations of other robots on the team to better track the position and velocity of the ball, and can also set up correlations in order to be able to use the ball as a beacon, thus improving not only the accuracy of the ball location, but also of the robots pose itself.

3.2 Kalman-Bucy Filter Algorithm

At its core the Kalman-Bucy Filter[5] is a recursive solution to the discrete-data linear filtering problem. It allows us to estimate the state of a process minimising the squared error. The filter makes the assumption that the noise in the measurements and time updates to be Gaussian, and the uncertainty in the state estimate is also represented as a Gaussian. The process to be estimated is governed by the stochastic difference equation:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1}$$

and measurement update:

$$z_k = Hx_k + v_k$$

In the equations above, A is an nxn matrix which relates the process state at the previous time step to the current state in the absence of control input. The nxm matrix B relates the control input u to the process state, and w is the process noise, which is assumed to behave as a Normal Distribution.

The Discrete Kalman Filter consists of two distinct steps, the time update and the measurement update. The time update uses the control input to update the belief state of the system, and the measurement update uses possibly noisy observations of the system state to improve the state belief estimate. The variables which we must keep track of between time steps are the mean vector, and the covariance matrix. The mean vector is the best estimate of the world state, and the covariance matrix is the multi-dimensional measure of the uncertainty of the current estimate.

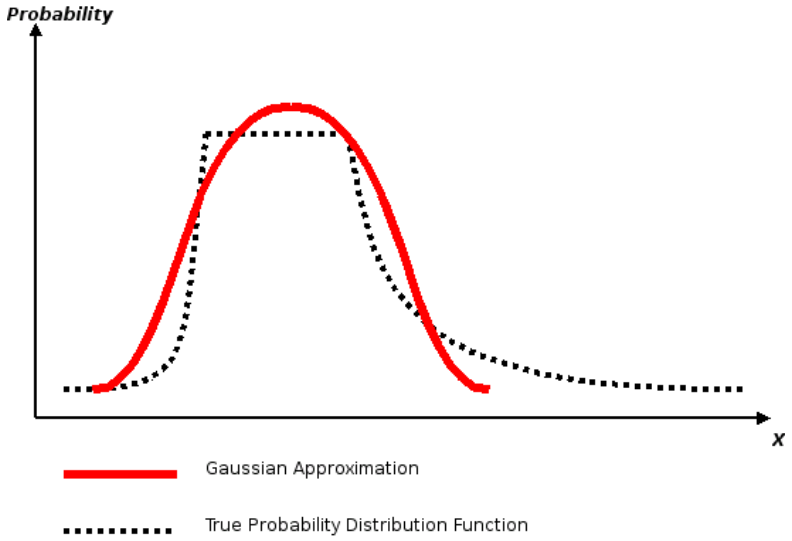

Figure 3.1: We approximate a probability distribution function with a uni-modal Gaussian distribution

The time update equations:

$$x_k = Ax_{k-1} + Bu_{k-1}$$

$$P_k = AP_{k-1}A^T + Q$$

The measurement update equations:

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$$

$$x_k = x_k + K_k(z_k - Hx_k)$$

$$P_k = (I - K_k H)P_k$$

In the time update step we must do two things, we must update the mean state estimate based on the control input, and we also need to update the covariance matrix P, that is, we need to increase our uncertainty estimate, due to the noise present in the control input.

The measurement update step is a more complicated process. Firstly we compute the Kalman Gain K, this is a measure of how much influence the observation z_k will have on the mean state estimate. For example, if we are very certain of our current estimate and we judge that the observation z_k is very unreliable, then the Kalman Gain K will be close to 0. However, if our

uncertainty estimate is very high, that is, we consider the current mean to be very unreliable, and at the same time we consider the measurement to be very accurate, then the Kalman Gain will be close to 1.

After we have computed the Kalman Gain, we can adjust the mean state estimate x_k by moving it in the direction of the Innovation Vector, $(z_k - Hx_k)$. The Innovation Vector can be seen as the direction in state space in which the mean vector needs to be shifted in order for it to more closely agree with the current state observation z_k. The more the currently observed state disagreed with the mean estimate, the greater will be the magnitude of the Innovation Vector, while if the mean estimate is in complete agreement with the observation then the Innovation Vector will be $\mathbf{0}$.

Finally, we must recompute the covariance matrix P, the uncertainty estimate. In general, observations tend to decrease the uncertainty estimate, while control updates tend to increase the uncertainty estimate. The derivation of the update of the covariance matrix is beyond the scope of this report. See the seminal paper by Rudolf E. Kalman[5].

We can apply the Kalman filter to the problem of Robot Localization as follows. The time update step is triggered by the actuator module, that is, when the robot makes a step, we can use the noisy odometry data to update the mean robot pose of the form (x, y, θ). The measurement update is triggered when the vision system detects a beacon or goal. We could use the noisy distance and heading to the landmark to form an observed state estimate z_k and then update the mean pose estimate. However, it is at this point that the algorithm breaks down, if applied to the 4-legged league domain. This is because the standard Kalman Filter assumes that the mapping between the state vector x_k and any observation is linear, in the above case, represented by the matrix H. So if we were able to observe directly, albeit with noise, the robot pose in (x, y, θ) form, then we would be fine. When observing a beacon, however, the information given implies that the robot pose can be anywhere on a helix in (x, y, θ) space. Knowing the distance to a landmark places you on a circle of a given radius around that landmark, and knowing the heading to it gives you a certain heading at every point on that circle, but they do not provide a single point. In order to deal with this issue, we need to move to the Extended Kalman Filter, which can handle non-linear mappings between observations and state.

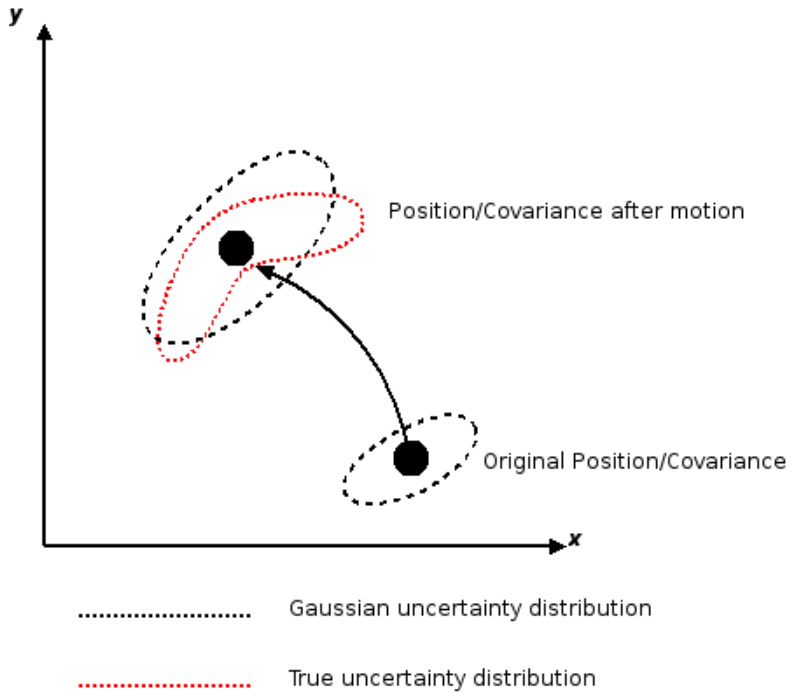

Figure 3.2: Gaussian approximation to non-linear motion update

3.3 Extended Kalman Filter

A Kalman Filter uses linear Gaussians over the state space to estimate the probability distribution function. However, as noted in the previous section, if a measurement or motion update has a non-linear nature, the classic Kalman Filter algorithm cannot handle this kind of situation. A solution to this problem is to linearise the function from state space to observation space around certain point such that we would now have a linear Gaussian approximation. Simply put, we can take the derivative of the non-linear function at a point $\mathbf{x}$ and use the "'line"' (or hyperplane) which has the given gradient and passes through the point $\mathbf{x}$ as the linear approximation. When applied to the Extended Kalman Filter, we must compute the multi-dimentional derivative of the non-linear function, which is called the Jacobian Matrix. Take the function $\mathbf{F}$ which maps an n-dimensional state space onto an m-dimensional observation space:

$$\mathbf{F} = \begin{pmatrix} f_1(x_1, ..., x_m) \\ \vdots \\ f_n(x_1, ..., x_m) \end{pmatrix}$$

The corresponsing Jacobian Matrix would be:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{pmatrix}$$

Each of the elements of the Jacobian Matrix is a partial derivative of the non-linear function F. We can now use this linearisation to be able to incorporate non-linear observations and time updates into our Kalman Filter. The time update equations becomes:

$$x_k = f(x_{k-1}, u_k)$$

$$P_k = AP_{k-1}A^T + Q$$

In the above equation, f is the function which updates the mean state estimate position, it may be non-linear, and the matrix A is the Jacobian of this function. The measurement update equations become:

$$K_k = P_k^- J^T (JP_k^- J^T + R)^{-1}$$

$$x_k = x_k + K_k(z_k - Jx_k)$$

$$P_k = (I - K_k J)P_k$$

and we can now assume that the state to observation equation is no longer linear, becoming:

$$z_k = j(x_k) + u_k$$

If we consider beacon observations to be two dimensional observations, being heading and distance to the beacon, we can derive a Jacobian matrix which is a derivative of the mapping between robot pose state space and observation space. The mapping between state space and observation space is as follows:

$$distance = \sqrt{(x_{robot} - x_{beacon})^2 + (y_{robot} - y_{beacon})^2}$$

$$heading = \tan^{-1} \frac{(y_{beacon} - y_{robot})}{(x_{beacon} - x_{robot})} - \theta_{robot}$$

If we now compute the first derivative of this function from state space to observation space, we get the following Jacobian matrix:

$$\mathbf{J} = \begin{pmatrix} -\frac{\frac{180}{\pi} * (y_{robot} - y_{beacon})}{(x_{robot} - x_{beacon})^2 + (y_{robot} - y_{beacon})^2}, & \frac{\frac{180}{\pi} * (x_{robot} - x_{beacon})}{(x_{robot} - x_{beacon})^2 + (y_{robot} - y_{beacon})^2}, & -1.0 \\ \frac{(x_{robot} - x_{beacon})}{\sqrt{(x_{robot} - x_{beacon})^2 + (y_{robot} - y_{beacon})^2}}, & \frac{(y_{robot} - y_{beacon})}{\sqrt{(x_{robot} - x_{beacon})^2 + (y_{robot} - y_{beacon})^2}}, & 0.0 \end{pmatrix}$$

A more thorough derivation and explanation of the Extended Kalman Filter can be found in a other papers[6].

3.4 Multi-Modal Localisation

The Extended Kalman-Bucy Filter is a powerful algorithm for robot localisation. However, one of its major downfalls is the fact that it can only approximate a uni-modal probability distribution function. So, for example, if the robot knew it was in one of two positions, say, it knew it was next to one of the two goals, the probability distribution function for the pose of the robot would have two local maxima, each centred near one of the goals. The Extended Kalman Filter, which uses a single Gaussian to represent the pose and uncertainty, would be unable to model this situation sufficiently well. This hypothetical situation would be much better modelled by the sum of two separate Gaussians, each of which is centred on one of the modes. In fact, it is possible to approximate any probability distribution function arbitrarily accurately using an infinite sum of weighted Gaussians.

To incorporate multiple modes into the localisation algorithm we use an array of uni-modal Gaussians, each with an associated weight. We can then view the full probability distribution as a weighted sum of Gaussians.

$$P(\underline{x}) = \sum_{i=0}^{N} w_i G_i(\underline{x})$$

This weight ranges between 1.0 and 0.0 and is a measure of the probability that that particular Gaussian represents the state of the system. When an observation update is made the weight of a Gaussian is updated in such a way that the Gaussians which match the observation well have a higher weight than the Gaussians which disagree with the observation. Given an observation covariance R, Jacobian J, and the covariance C of the Gaussian prior, we can calculate the

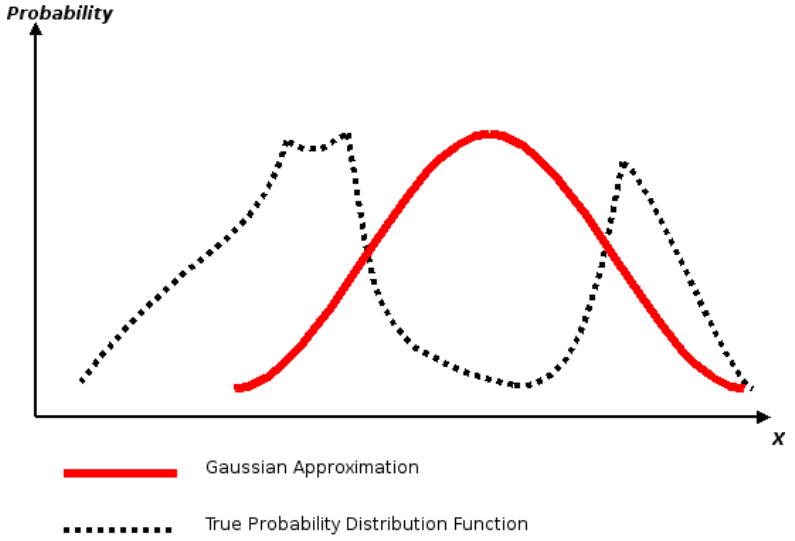

Figure 3.3: A single Gaussian approximates a multi-modal probability distribution poorly

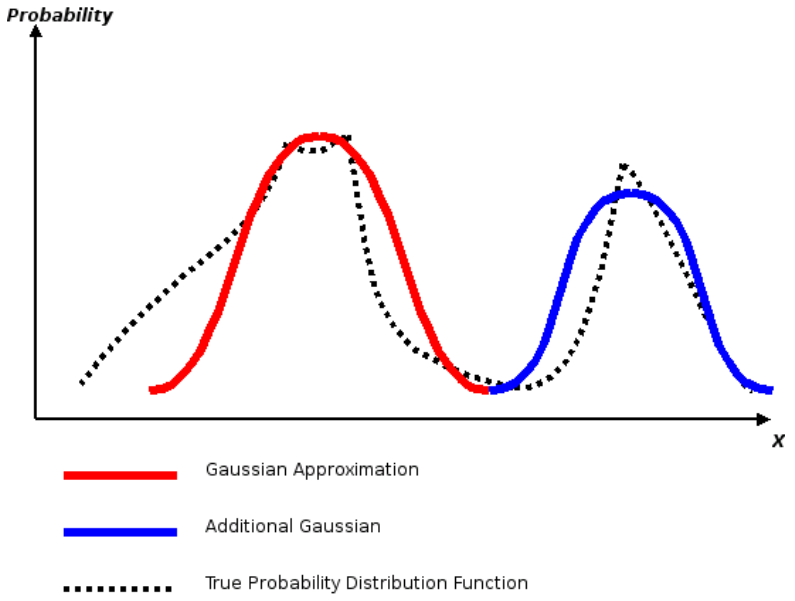

Figure 3.4: A weighted sum of Gaussians provides a far closer approximation to the true distribution

combined covariance E. Combining this with the innovation vector v we calculate the weight scalar S, which allows us to update the weight of the Gaussian distribution.

$$E = R + J^T C J$$

$$S = exp(-\frac{1}{2} v^T E^{-1} v)$$

$$w_i = S * w_{i-1}$$

The sum of the weights of all of the Gaussians in the distribution must sum to 1.0 in order to make the entire distribution a valid probability distribution (one which integrates to 1.0 from $-\infty$ to ∞). In order to maintain this property a renormalisation must happen every time the weight of a Gaussian is modified. In addition to this, Gaussians which have a very low weight are removed from the distribution array. This is because they represent extremely unlikely modes, and for performance reasons we cannot keep track of unlikely modes. Another method used to reduce the number of Gaussians that form the distribution is merging similar Gaussians. If two Gaussians have a similar mean and similar covariance matrices, then one of them is removed and the other becomes the average of the two. To calculate the global maxima of the weighted sum of Gaussians distribution a simple approach is taken, the maxima is assumed to be the mean of the Gaussian of highest weight. This is not strictly correct, but is a good enough approximation, seeing as a correct solution for finding the global maximum of a sum of Gaussians is alot more involved and does not provide enough of a benefit for it to be used in our system. See Derrick Whaite's undergraduate thesis[3] for further information on the way in which our localisation system has been modified to be multi-modal.

3.4.1 Algorithm

Our system is based heavily on the previous years system. The largest difference is the changed mean vector, we now track a 16 dimensional state space, rather than a 3 dimensional one.

$$\mathbf{MeanVector} = \begin{pmatrix} robot\ xpos \\ robot\ ypos \\ robot\ theta \\ ball\ xpos \\ ball\ ypos \\ ball\ dx \\ ball\ dy \\ teammate_1\ xpos \\ teammate_1\ ypos \\ teammate_1\ theta \\ \vdots \\ teammate_3\ xpos \\ teammate_3\ ypos \\ teammate_3\ theta \end{pmatrix}$$

In addition to this, we now have a non-identity transition matrix which maps the previous timestep state to the current one in the absense of control input. We use this to estimate the movement of the ball, by additing the velocity of the ball to its position.

$$\mathbf{MotionMatrix} = \begin{pmatrix} \mathbf{I_{3,3}} & & & & \\ & 1.0 & 0.0 & 1.0 & 0.0 \\ & 0.0 & 1.0 & 0.0 & 1.0 \\ & 0.0 & 0.0 & friction & 0.0 \\ & 0.0 & 0.0 & 0.0 & friction \\ & & & & \mathbf{I_{9,9}} \end{pmatrix}$$

The addition of a distributed nature of the localisation algorithm, we must be able to communicate a robots observation and motion information to its teammates. Ideally we would do this by sending the distance and heading information of the observation to all of our teammates

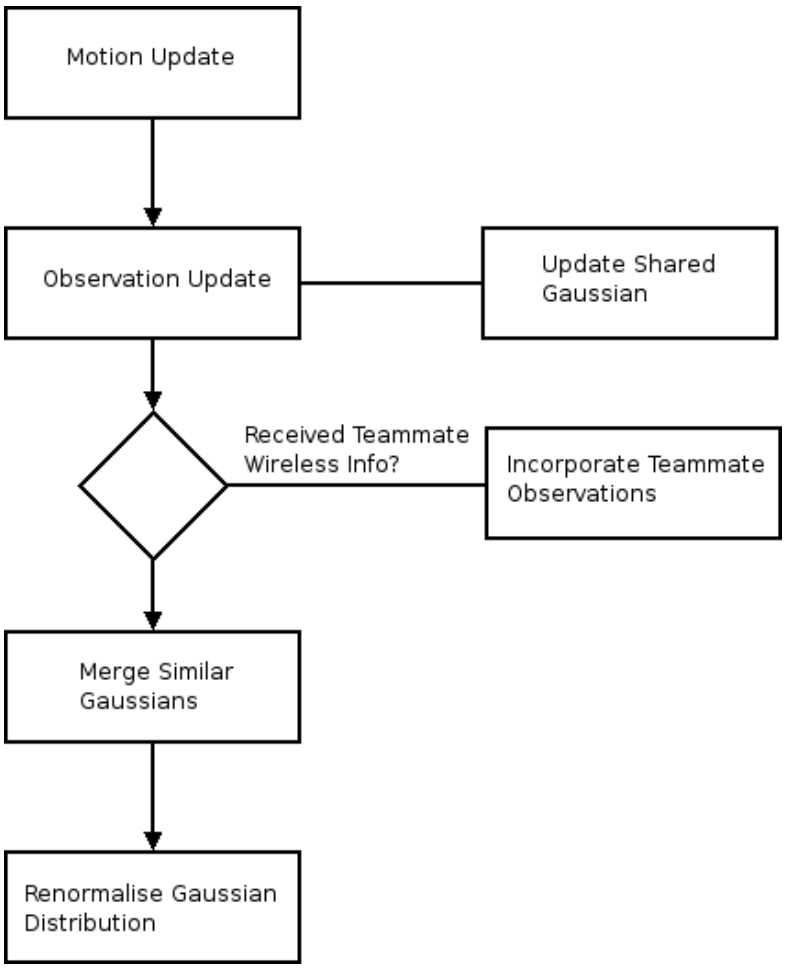

Figure 3.5: This diagram provides an overview of our localisation system

every frame. However, we cannot do this due to the latency and limited bandwidth of the wireless communication. As a result, we instead aggregate the observations and send the result every 5 frames. We do this by keeping a separate Gaussian which we update like normal, but every 5 frames we broadcast it over wireless and reset the Gaussian.

3.4.2 Noise Filtering

Using the power of a multi-modal filter we can improve the robustness of the system to spurious observations. In the 4-legged league it is very likely that the vision system will make false classifications of objects which are part of the background, such as spectators wearing coloured tshirts. These may be classified as beacons, or goals, or the ball. This is a different type of noise in the system to what a standard filter is designed to handle, that is, noise that is centered on the mean. We need a way to be able to reject spurious observations, otherwise they will significantly reduce the accuracy of the localisation system, more so than standard "'noisy"' observations. Take as an example a seeing a spectator wearing an orange tshirt in the crowd, which the vision system classifies as a very large (and thus close) ball. The variance of this observation will be small because the closer the ball is the more certain we are about its observed distance. This results in our estimate of the ball position shifting significantly towards the observed "'phantom"' ball position, despite the observation being false.
Our solution to this problem is to say "'this observation may or may not be correct"', and as such when we apply the observation to every Gaussian in the weighted sum distribution we also make a copy of the Gaussians which do not have the observation applied, but we scale their weights down by a constant factor. This constant factor can be seen as the probability of a false observation. This means that for every observation, the system doubles the number of Gaussians which make up the distribution. These are later culled if there are too many or their weights are too small. An observation is said to be a phantom observation if the weight of the Gaussian with it applied is lower than the weight of the Gaussian without the observation applied.

This technique works because the more an observation disagrees with the current state, the lower the weight will be of the resulting Gaussian after applying the observation. So if an observation is made which is extremely unlikely, and so is probably a false one, the resulting

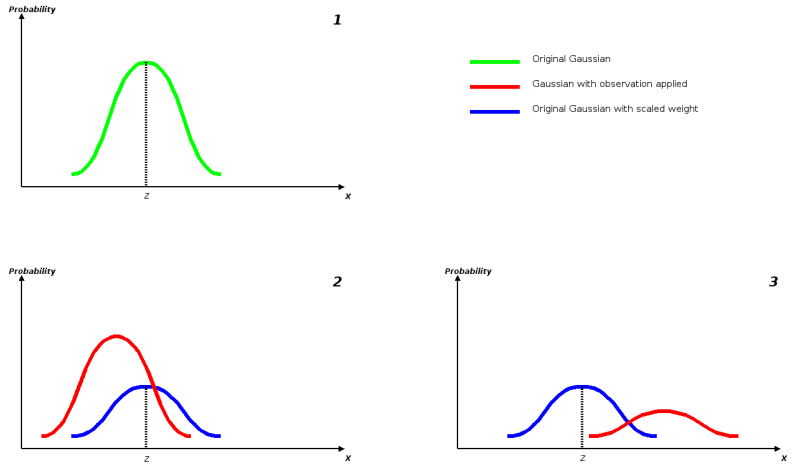

Figure 3.6: The Gaussian is split upon receiving an observation update, one has the observation applied, the other doesnt

Gaussian will have a lower weight than the Gaussian without the observation applied.

The effectiveness of this approach was demonstrated to us when we accidentally swapped two of the beacons around and put the robot into the Ready State, whereby it has to position itself at a kick-off position. This results in 4 valid landmarks (2 goals and 2 beacons), and 2 invalid landmarks (the 2 swapped beacons). Despite expecting the robot to localise poorly due to the contradictory observations, the robot localised extremely well, rejecting almost all observations which were of the switched beacons. This robustness to false observations was extremely important at the competition due to the fact that there were spectators close to the field at "'eye level"' who were wearing coloured shirts. Without this noise filtering in the localisation system, we would have been far less well localised.

3.4.3 Multiple Linearisation Points

One of the sources of error in an Extended Kalman Filter comes from the fact that we are approximating a possibly non-linear probability distribution with a linear Gaussian function.

This is one of the advantages of a particle filter approach to localisation, particle filters do not have to linearise a non-linear distribution, since they can approximate any probability distribution function. However, using a sum of multiple weighted Gaussians to represent our function, we can reduce the error from the linearisation process by better approximating the

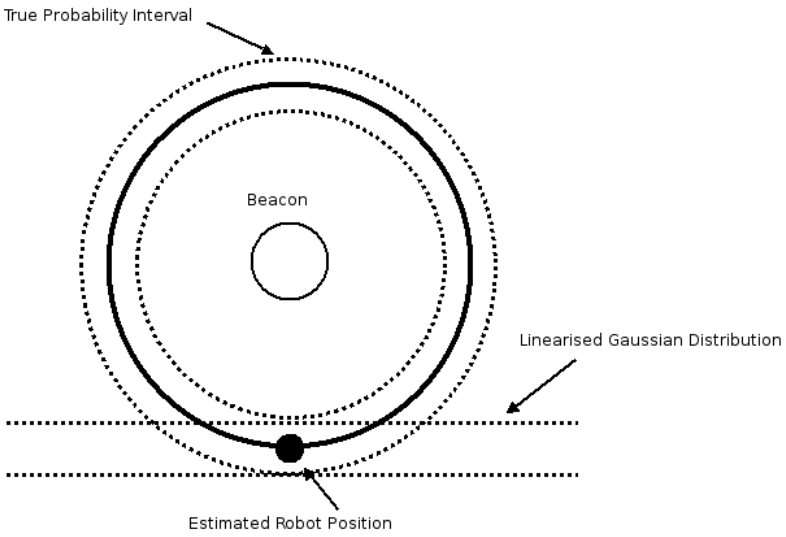

Figure 3.7: The standard Extended Kalman Filter linearises around only one point, the mean of the estimates world state

non-linear function.

The way in which we do this is every time a single beacon (note that for the purposes on this section, the ball should be considered as being a beacon) is observed, a copy is made of every Gaussian in the distribution, with the mean of each copy being offset such that it is the same distance from the observed beacon, but is rotated around by a given angle. The weights of these new displaced Gaussians are also scaled down. It is important to note that the displaced Gaussians can only be generated when only one landmark is observed. This is because it is not consistent with the observations to rotate a Gaussian around a beacon if there are multiple observed beacons. Every time there is a single beacon observation, only one displaced copy is made per existing Gaussian, whereas we would need 2 to maintain symmetry. This is done with the aim of improving the speed of the localisation module, spawning two additional Gaussians would have been too expensive, so instead we alternate whether to rotate the added displaced Gaussian clockwise or anti-clockwise around the beacon. In our implementation we chose to rotate the displaced Gaussian by 16 degrees, and the weights are multiplied by 0.1. The end result of this is a better approximation of the probability distribution function through a reduction in the inherent error introduced by the linearisation process.

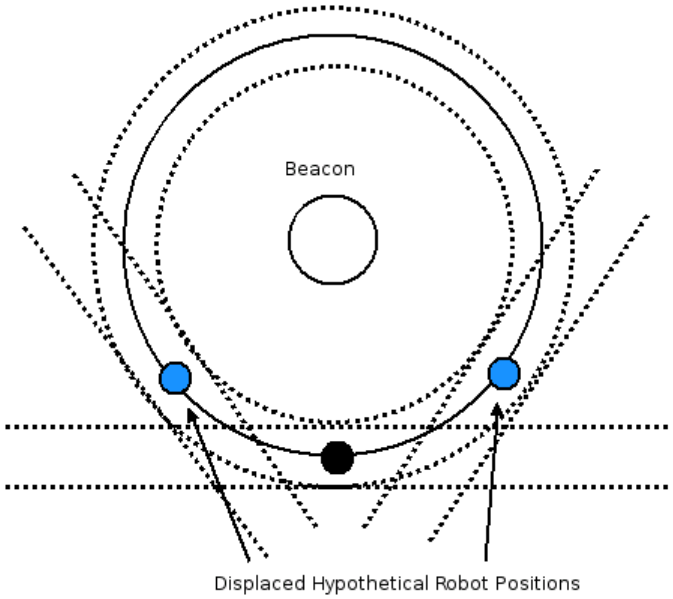

Figure 3.8: Our system adds low weight Gaussians rotated around the beacon, meaning we linearise around multiple points

3.5 Incorporating Teammate Observations

This year we have implemented distributed observations of the world state into the Kalman Filter localisation. We can do this very easily seeing as our world state mean vector contains the pose of all of the robots on the team. Our algorithm involves keeping track of a separate "'Shared Gaussian"', which we update as per normal, but it does not form part of the main weighted sum of Gaussians probability distribution function. This Shared Gaussian is sent across wireless to all teammates, followed by "'resetting"' it. Resetting the Shared Gaussian involves essentially increasing the variance to an extremely large number, and setting the mean to that of the main weighted sum of Gaussian distribution. We need to do this because otherwise an observation incorporated into the Shared Gaussian would be sent to the teammate robots multiple times. In addition to sending the Shared Gaussian mean vector and covariance matrix, we also send the cumulative odometry information, which we also reset every time a wireless packet is broadcast.

When a wireless packet is received, we can incorporate the teammate observation into the main probability distribution as a direct observation. A direct observation essentially means there is no need to linearise the function mapping observation to state space, since it is already

a linear one. This is the case in this situation because the beacon observations have already been linearised by the sending robot. The data sent is of the form of a 7 dimensional mean vector and a covariance matrix. What the receiving robot must do is have a matrix which maps the teammate pose and ball position/velocity estimates into its own world estimate.

$$\mathbf{A} = \begin{pmatrix} \mathbf{0_{7x3}} \\ \mathbf{0} \\ \mathbf{I_{3x3}} \\ \mathbf{0} \end{pmatrix}$$

$$\mathbf{B} = \begin{pmatrix} \mathbf{0_{3x4}} \\ \mathbf{I_{4x4}} \\ \mathbf{0_{9x4}} \end{pmatrix}$$

$$\mathbf{H} = \begin{pmatrix} \mathbf{A}|\mathbf{B} \end{pmatrix}$$

The matrix $\mathbf{H}$ is the mapping between the wireless teammate observation and our world state estimate. The matrix $\mathbf{B}$ is constructed such that the placement of the $3x3$ identity matrix corresponds to the index of the robot id from which the packet was received, such that our idea of where that teammate robot is positioned is updated accordingly inside the mean vector.

Overall the inclusion of teammate observations has greatly increased the accuracy of ball position and velocity tracking this year. We demonstrated the effectiveness of incorporating teammate observations by testing to see if a robot could grab a ball under its chin with distance observations turned off, but with teammate robots on the field. This essentially means that the robot is relying on heading observations from other robots on the field to the ball to determine the position of the ball. To the surprise of some of the spectators at the time, the robot actually managed to grab a ball several times, which would have been impossible had we not implemented the distributed observations into the localisation system.

In addition to this, not only the accuracy of the ball tracking is improved by this scheme, but also of the robot pose itself. This is because our scheme allows for the ball to become essentially a moving beacon which the robots can use to localise from. For example, if a very well localised

robot is looking at the ball, it can transmit a very accurate and certain position of the ball to its teammates. Following this, a poorly localised teammate robot can look at the ball, and knowing its global coordinate position, localise from it as it would from a normal beacon. This allows us to reduce the frequency of what is called "'active localisation"'. This involves a robot looking aways from the ball to glance at a beacon to relocalise itself. This has been a necessity in the past because if the Attacker robot is chasing the ball for a prolonged period of time it will become severely mislocalised having seen few if any beacons. The downside of active localisation is that it forces the robot to look away from the ball, possibly losing track of it after it finished glancing at a beacon. But with our system the ball itself can be viewed as a beacon, and as such the rate at which a chasing robot becomes mislocalised is greatly reduced, allowing it to focus more on the ball and less on looking around for beacons.

3.6 Localisation Simulator

Robocup is a very competitive environment, one where time is an extremely limited resource. As such, at the beginning of the year, we decided that it would be advantageous to write a very simple simulation environment where we could test some of the proposed changes to the existing localisation module. The main aim was to get an idea of the benefits of having a distributed Kalman Filter, incorporating teammate observations using wireless. The simulator was written to model two moving robots, 4 beacons, and a moving ball. It was set up such that the noisiness of the observations could be varied, packet loss could be simulated, the frequency of observations, the variance of the motion model, etc etc. Using this we could simulate a large range of situations very quickly and easily, as well as measure the error of the system, since we had easy access to the ground truth as well as the world state belief generated by the localisation system. We conducted several experiments using this simulator to gauge the effectiveness of the changes we made to the base algorithm. The basic setup was 4 beacons, each at a corner of the field, 2 robots moving in circles and constantly scanning by moving their heads in a circle, and a ball moving left and right across the field. The experiments were conducted by taking the sum of the distances between the mean belief vector and the ground truth vector over every frame for 10000 frames. These limited experiments revealed the advantages of a distributed localisation

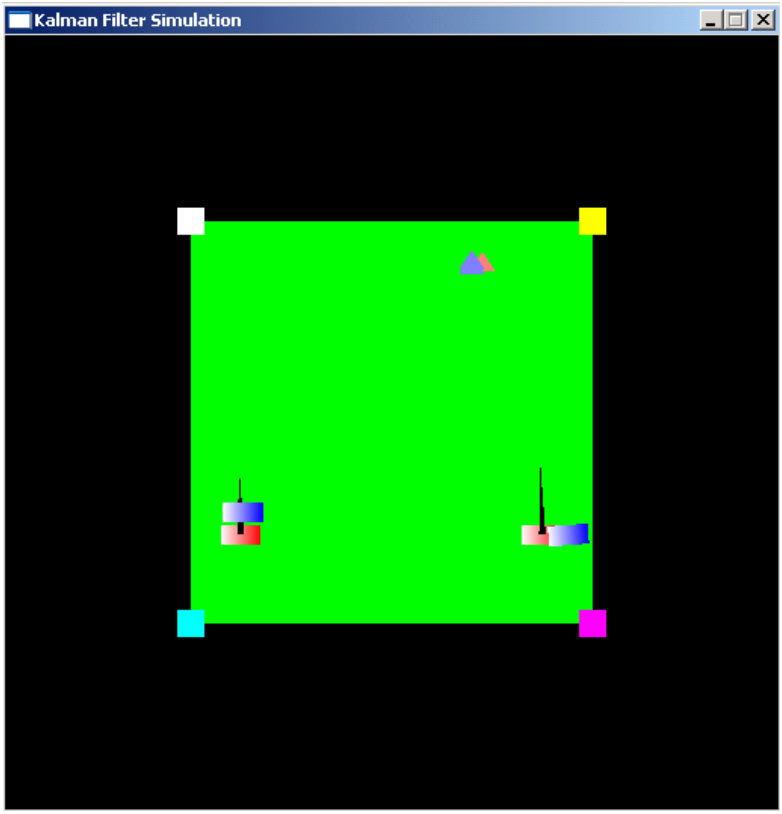

Figure 3.9: This is a screenshot of our simulator, the triangle is the ball, the rectangles are the robots, the squares at the edges are the beacons. Blue objects represent the estimated pose of the robots, the red objects are the actual poses of the objects

system. Varying the distance variance of the observations gave the following results:

Distance Observation Variance	Error With Communication	Error Without Communication
0.5	18.135	42.1
2.0	20.2	42.9
8.0	46.0	91.7

It is important to note that this simulator is extremely simple and makes many crucial assumptions. Firstly, and possibly most importantly, the noise in the observations and motion model is modelled to be Gaussian noise in the simulator, which matches perfectly the assumptions made by the Kalman Filter algorithm. This is definitely not the case in a real 4-legged league environment. We also model the wireless communication between the robots as being instantaneous and lossless (no packet loss). This again is not the case in the real world, and most definitely not the case at the competition site where the wireless communication is subject to severe interference. Despite these assumptions which are not upheld in a real game situation, the improvements that were apparent due to the introduction of a distributed Kalman Filter warranted its implementation in the system proper.

3.7 Machine Learning Optimal Parameters

One of the requirements for a localisation system is a model for the way in which the world behaves. For example, when we use odometry to estimate the change in the mean of the robot pose, we must have an idea of how far the robot actually moves when given a certain set of walk parameters. Another example is the rate at which a rolling ball will slow down to a stand still, we need this to be able to predict the position of the ball at some future time given its current velocity. In the past, many of the constants which define our world model have been either calculated by hand using sample measurements, or have been approximated in an ad-hoc manner. Taking the variance of a beacon distance observation as an example, in the past an expert user has had to take a number of measurements of the beacon distance from different areas of the field, and then use statistical means to calculate the variance. This is a very time consuming process and can be prone to error, automation of this process would be of great help,

especially when arriving at a new venue where time is limited. Some of the variables, such as say the rate at which the robot pose variance increases, have been essentially "'guessed"' in the past. In fact, there were a large number of such variables in our localisation system, many of which are crucial to the overall performance of the module. If an optimal parameter set could be found for these variables, then we could expect the accuracy of the localisation to increase as compared to guesstimated values.

This year we have developed an automated process for finding the optimal set of parameters to describe the world model. Firstly, let us look at the meaning of the "'weights"' of each Gaussian in the probability distribution. The weight of a single Gaussian in the sum of Gaussians distribution signifies how well the estimate of the state of this Gaussian agrees with the sensor measurements. These weights are recalculated every time an observation of the state is made, so for example, whenever the robot sees a beacon. As such, if a Gaussian agrees well with an observation, its corresponsing un-normalized weight will be higher than if it disagrees with the observation. This makes it possible to use the total product of the un-normalized weights of the topmost Gaussian as a fitness function for the localisation system. However, a problem arises if we just try to keep track of the cumulative product of the weight, due to the fact that the weight is between 0.0 and 1.0, resulting in the cumulative product quickly going to 0.0 due to the limited precision of floating point numbers.

$$0.0 < w_i < 1.0 \implies \prod_{i=0}^{n} w_i \to 0.0$$

However, we are looking for a fitness function which can compare the accuracy of various policies with each other. As such, it would be suitable if we could transform the cumulative product of weights in such a way that it was mathematically stable while keeping the relative ordering of policies the same. One such function we could use is the natural logarithm function. It is a strictly increasing function, so it would be suitable for keeping the relative ordering of the cumulative products of weights, and it also is mathematically stable.

$$\ln(\prod_{i=0}^{n} w_i) = \sum_{i=0}^{n} \ln(w_i)$$

We can use this result to now instead keep a cumulative sum of the natural log of the weights. The sum will always be negative, the more negative it is, the more the observations disagree

with the estimated state, perfect agreement is signified by a cumulative sum of 0.0.

Now that we have a suitable fitness function, we develop a procedure for estimating the world model. Firstly, we intend for this procedure to be offline, that is, the learning of the optimal parameters happens on a workstation, rather than a robot. What we needed to do is to play a sample game and record all of the input data into the localisation module into a log file. These inputs include such things as the walk-engine control input, the beacon and goal observations, etc etc. Using this log file, we can effectively "'play back"' the game from the perspective of the localisation module, all offline. We then came up with a large number of variables, which were previously mostly ad-hoc "'magic numbers"' in the localisation module, which we can say approximate the world model. In total, we had around 50 such parameters, resulting in a 50-dimensional search space. Now, when we play back the game through the localisation module, different parameters will result in varying fitness. This provides us with the ability to apply a well-known function minimisation technique to find the best set of parameters. Since this is an offline algorithm, and as such we were not extremely concerned with speed or efficiency, we used an existing implementation of the Simplex Method for this task, without investigating other, possibly faster methods.

It is important that this method provides accurate results. One of the ways that we verified the correctness of this general approach is to compare the values it returns for variables which can be calculated by hand, and confirm that they are a close match. Our method involved taking values which have been calculated in the past, for example the distance variance for a beacon observation, and use the paramter optimisation to compute linear shifts for these values. What we found is that the linear shifts generated are very close to the line $y = x$, which implies that the values computed by hand agree very well with the optimality criterion of the optimisation algorithm.

The main areas to which we applied this technique is to replace what were previously ad-hoc "'magic numbers"' with values derived from this technique. Doing this has increased the overall performance of the localisation system greatly. In future, it may be feasable to apply this method to many more aspects, one of this is odometry calibration. Currently, the mapping between raw walk commands and the actual walking motion of the robot is calculated by hand, using multiple trials. This is a very time consuming process, and one which can very easily induce mistakes.

If we view the mapping between raw commands and actual motion as just another aspect of the world model, it should be possible to apply this technique to derive the relationship very accurately and quickly.
See Ryan Cassar's undergraduate thesis[15] for a description how this system can be used for automatic odometry calibration.

3.8 Results

This year we have gone to some effort to provide concrete results for the effectiveness of the changes we have made to the localisation system. The hardest part of evaluating the effectiveness of the system is having a ground truth from which to measure the error estimate of the robot pose. Some teams such as the German Team have the man-power to be able to develop systems such as an overhead camera which can accurately track robots and provide ground truth. Unfortunately, our small team cannot spare the time for such a project. As a result, we came up with a trial run between waypoints test. Essentially, this includes placing several waypoints on the field at known locations, and running a robot between them in order while the robot is scanning from left to right with its head. The robot then stops at each waypoint, allowing an operator to measure the distance between the robot at the point where it believes it is at the waypoint, and the actual position of the waypoint. This gives us an error estimate, the lower the distance measured, the better the accuracy of the localisation system. Unfortunately, this is form of testing has the downside that it is difficult to measure the accuracy of the heading, so we take only the (x, y) part of the robot pose into consideration. There are multiple scenarios to which we applied this methodology to test the various aspects of the localisation system.

3.8.1 Base Test

The first test which we ran is a base test. This involves 4 waypoints, each at a corner of the field, and all of the beacons and goals. The robot starts near the center circle and runs between the waypoints in increasing numbered order. We use this test to compare the accuracy of the previous localisation system, and the new localisation system, specifically testing the multiple-linearisation modification, under near ideal conditions.

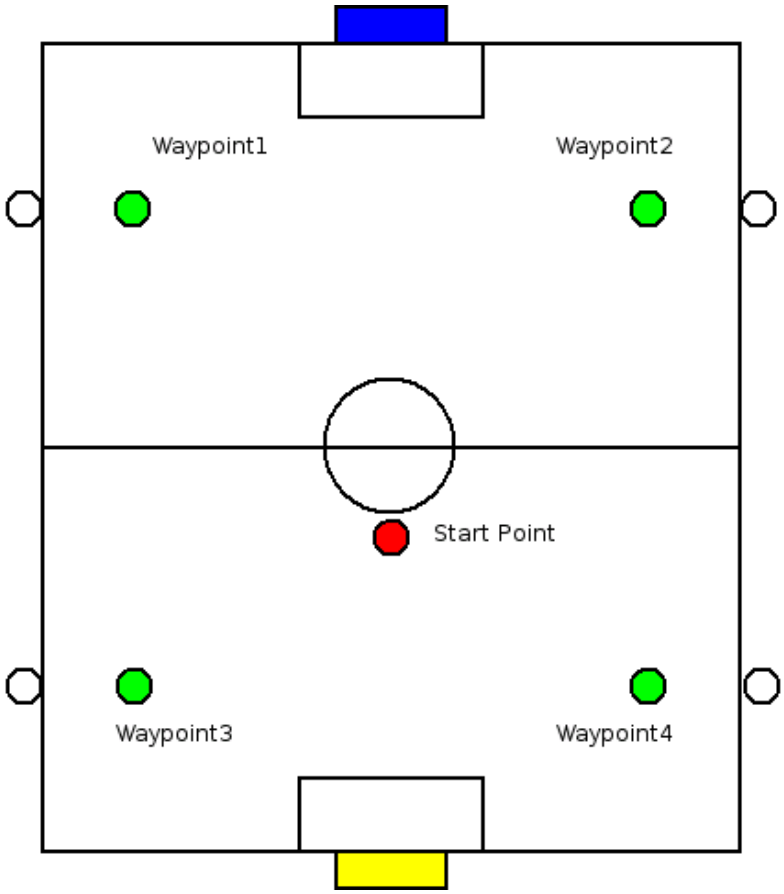

Figure 3.10: This figure shows the layout of the Base Test, the waypoints are visited in order, all beacons are present

Full 2005 Localisation System			
Waypoint1	**Waypoint2**	**Waypoint3**	**Waypoint4**
42.0	26.0	13.0	180.0
17.0	18.0	17.0	20.0
38.0	20.0	7.0	82.0
37.0	20.0	58.0	15.0
29.0	10.0	62.0	17.0
Average Error(cm): 28.8			

Full 2006 Localisation System			
Waypoint1	**Waypoint2**	**Waypoint3**	**Waypoint4**
35.0	9.0	33.0	6.0
27.0	15.0	29.0	16.0
50.0	15.0	16.0	14.0
28.0	8.0	1.0	19.0
46.0	23.0	16.0	30.0
Average Error(cm): 21.8			

3.8.2 2 Beacons Test

This test is very similar to the Base Test, except we now removed 2 beacons, making it much harder for the robot to localise as it receives far fewer observation updates. This we hoped would show a larger disparity between the old and new localisation systems, allowing the better approximation of the probability distribution function due to the use of the multiple linearisation points to have a greater effect on the relative accuracy of the system.

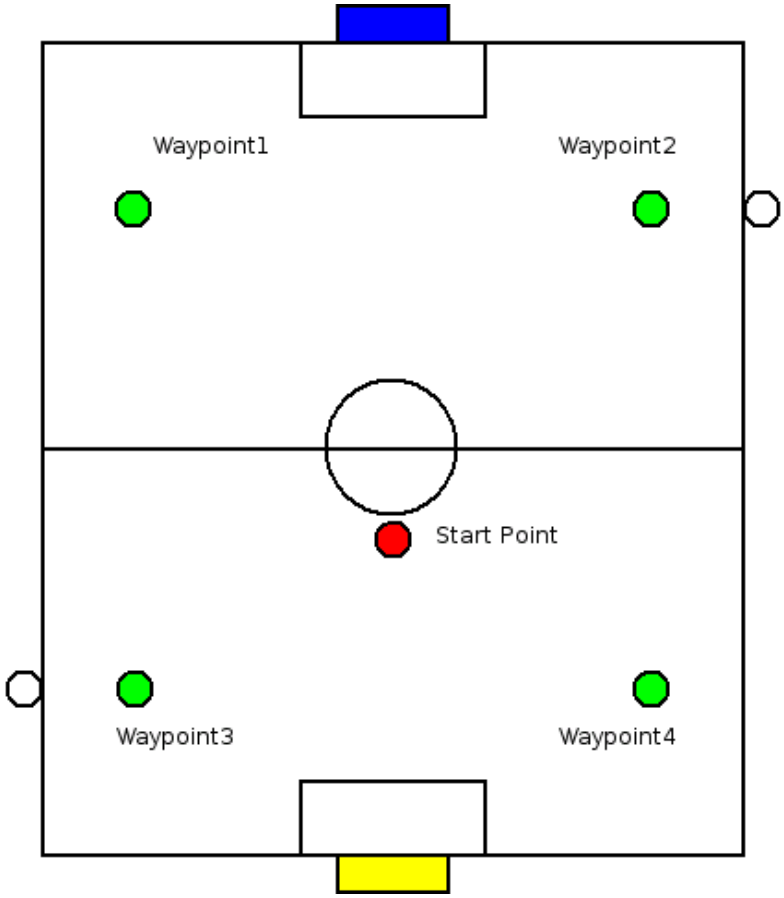

Figure 3.11: This figure shows the layout of the 2 Beacon Test, 2 of the beacons are removed

Multiple Linearisation Disabled			
Waypoint1	**Waypoint2**	**Waypoint3**	**Waypoint4**
88.0	20.0	22.0	30.0
37.0	20.0	17.0	33.0
21.0	44.0	15.0	15.0
44.0	49.0	10.0	25.0
100.0	38.0	40.0	39.0
Average Error(cm): 35.35			

Multiple Linearisation Enabled			
Waypoint1	**Waypoint2**	**Waypoint3**	**Waypoint4**
40.0	20.0	36.0	13.0
30.0	1.0	28.0	16.0
10.0	14.0	43.0	17.0
40.0	20.0	38.0	13.0
15.0	10.0	40.0	19.0
Average Error(cm): 23.15			

3.8.3 Distributed Ball Localisation Test

This test focuses on showing how the ball can become a beacon if we have teammates which are observing it and sending us their observations. In this test, we remove 2 of the beacons such that the test robot is relying more on the ball for its localisation than the beacons, and we have a teammate robot positioned off to the side, standing still and scanning, seeing beacons and the ball. This test should demonstrate the effectiveness of the distributed nature of our filter.

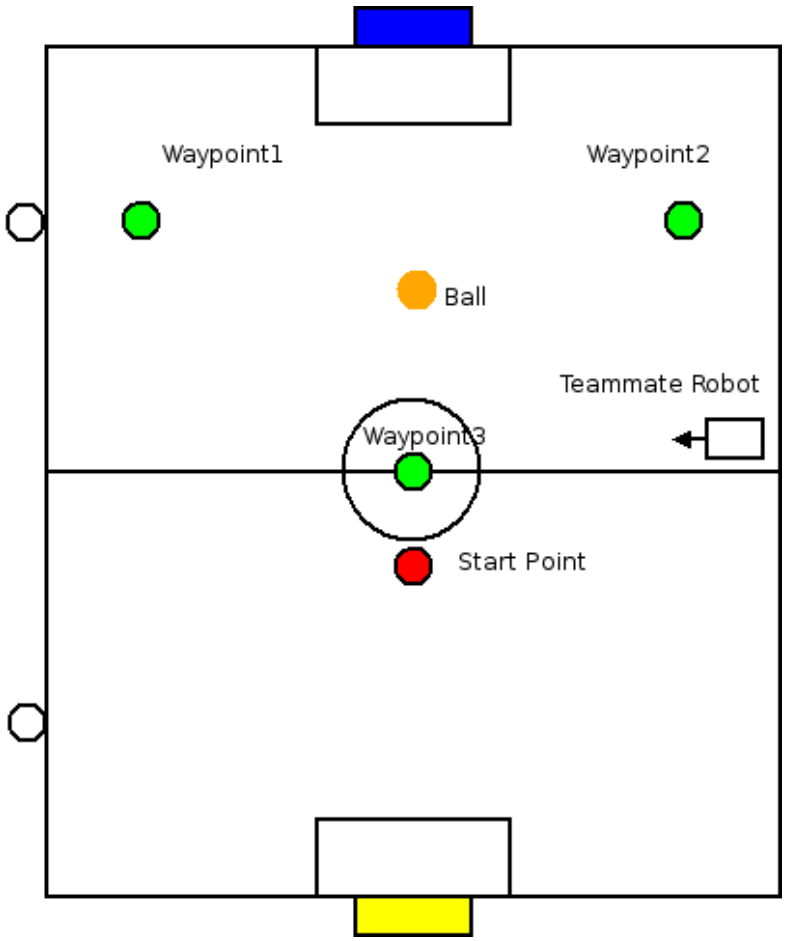

Figure 3.12: This figure shows the layout of the Distributed Ball Localisation Test, 2 beacons are removed, a teammate robot is present localising the ball

Distributed Localisation Disabled		
Waypoint1	**Waypoint2**	**Waypoint3**
34.0	22.0	25.0
20.0	23.0	15.0
31.0	31.0	24.0
23.0	8.0	9.0
19.0	15.0	23.0
Average Error(cm): 21.47.0		

Distributed Localisation Enabled		
Waypoint1	**Waypoint2**	**Waypoint3**
29.0	9.0	28.0
3.0	18.0	22.0
13.0	10.0	36.0
14.0	35.0	7.0
17.0	18.0	27.0
Average Error(cm): 19.1		

3.8.4 Noise Filtering

Our system is multi-modal in nature, allowing it to consider observations as possible false and possibly true. This results in spurious observations having a far lower effect on the accuracy of the system. If the robot observes a landmark which does not make sense for the current estimated world state then it is possible for the system to deal with this by using the Gaussian without the observation applied as the mean Gaussian instead. The setup for this test is similar to that of the Base Test, except that we swap around the Yello on Pink and Pink on Blue beacons. This result in any observation of either of these beacons as being "'false"', and hopefully the localisation system will cull them.

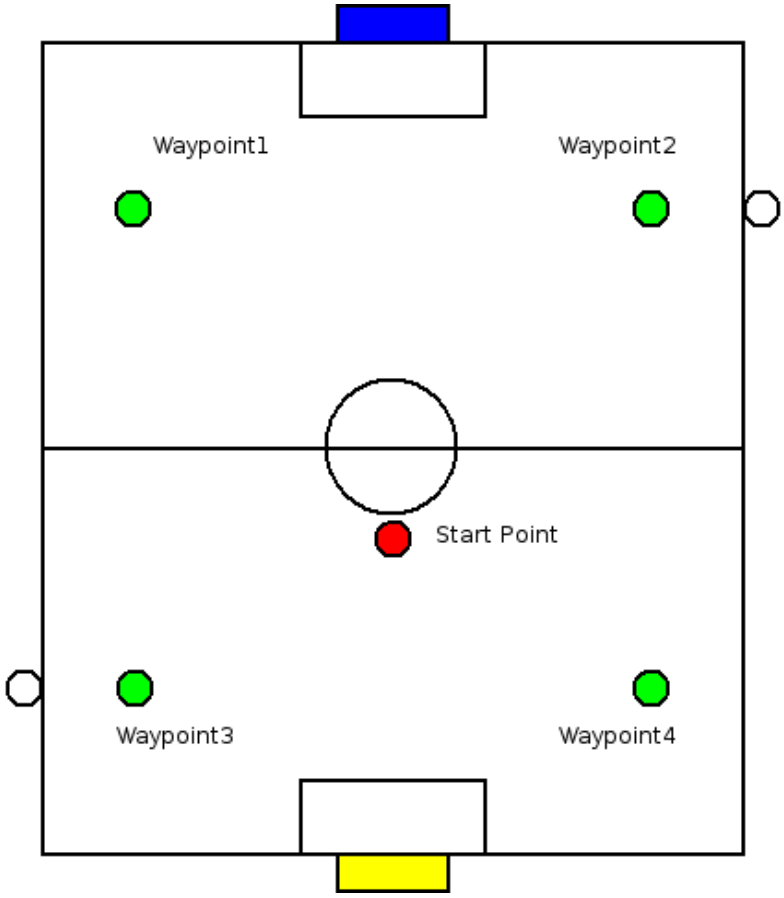

Figure 3.13: This figure shows the layout of the Noise Filter TEst, 2 of the beacons are swapped around

Noise Filtering Disabled			
Waypoint1	**Waypoint2**	**Waypoint3**	**Waypoint4**
150.0	400.0	180.0	111.0
93.0	140.0	200.0	142.0
168	350.0	188.0	116.0
Average Error(cm): 186.0			

Noise Filtering Enabled			
Waypoint1	**Waypoint2**	**Waypoint3**	**Waypoint4**
19.0	28.0	38.0	14.0
35.0	39.0	22.0	12.0
39.0	15.0	42.0	12.0
Average Error(cm): 26.25			

3.9 Discussion

The above results show that our improvements this year have had a great effect on the overall accuracy of the localisation system. The effectiveness of the Noise Filtering part of the system is staggering. It should be also noted that these experiments only measured the (x, y) pose error, whereas the localisation system tracks much more data than those 2 dimensions, including the ball position and velocity. Unfortunately we were unable to concretely demonstrate the improved accuracy of the ball tracking, but it is evident through the speed with which we can re-aquire, track, and grab the ball, all of which rely on accurate ball tracking. It should be noted that this report does not describe the line localisation part of the system, see Michael Lake's undergraduate thesis[14] for a description of how the localisation system can use line detection to correct its pose estimate.

Chapter 4

High Level Behaviours

The previous chapter described one of the low level foundations on which the high-level behaviours are based, the localisation module. In this chapter, we describe some of the new skills we have developed, and the improvement made upon existing skills. The development process for high-level behaviours is an extremely ad-hoc one, since it relies more upon the experience of the developer to be able to tell what is a "'good"' behaviour, while something like localisation or vision, one can objectively judge the performance of the module.

4.1 Behaviour Overview

The rUNSWift behaviour architecture is separated into distinct levels, which defines how "'high level"' the behaviour is. At the top of the heirarchy are the player modules, such as the pForward and the pGoalie players. In a standard game, 3 of the robots will be running the pForward player module, while the goalie will be running the pGoalie module. The robots cannot switch dynamically between player modules, they are locked in. At the next level of the heirarchy are the roles, such as Defender, Attacker, or Supporter. At any point in time, each Forward has a given role, and can dynamically switch between roles depending on certain criteria, such as distance from the ball. This year we have paid alot of attention to the dynamics of role switching. The next level consists of "'skills"', that is, behaviours which are used by all of the roles, for example, finding the ball or avoiding running into fellow teammates. Finally, there is a set of helper modules, such as math functions.

4.1.1 Finding and Intercepting the Ball

Being able to re-aquire the ball visually after having lost it, as well as being able to chase the ball once you have aquired it, is arguably the most important skill in Robocup. This is because the effectiveness of these two combined skills determines which team will be able to get to the ball first and thus affect gameplay to their advantage. In previous years rUNSWift has relied on having a pure walking speed advantage over almost all of the teams at the competition. This year, however, rUNSWift had in fact one of the slower walks out of all of the teams, but nonetheless we managed to be the first team to the ball in the vast majority of situations. There are two main features which were added this year to the FindBall module which have greatly improved its effectiveness, Variance Scan and Velocity Intercept.

Variance Scan

One of the actions performed by the robot after it has lost sight of the ball is to go into a scan, that is, moving its head left and right, intending to re-aquire the ball. In the past, this has been full scans from left to right and back. However, this is not the best way of approaching the problem, since we can use the "'gps ball"' as a good idea of where to focus our search, gradually increasing the scan arc length as the uncertainty of the ball position grows. So for example, if a robot is tracking the ball, and then another robot obscured the ball for a short amount of time, the subject robot will do small scans centered on the gps ball position, thus allowing it to reaquire it more effectively than if it straight away panned its head 90 degrees to the side to begin a full range scan. The question then becomes how do we decide how wide a scan arc should be at any point in time. We do this by calculating the maximum variance of the ball position, which can be derived from the covariance matrix. The maximal variance is simply the largest eigenvector of the covariance matrix.

$$\textbf{Covariance} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

$$maxVar = \frac{-(a+d) + \sqrt{(a+d)^2 - 4(a*d - b*c)}}{2}$$

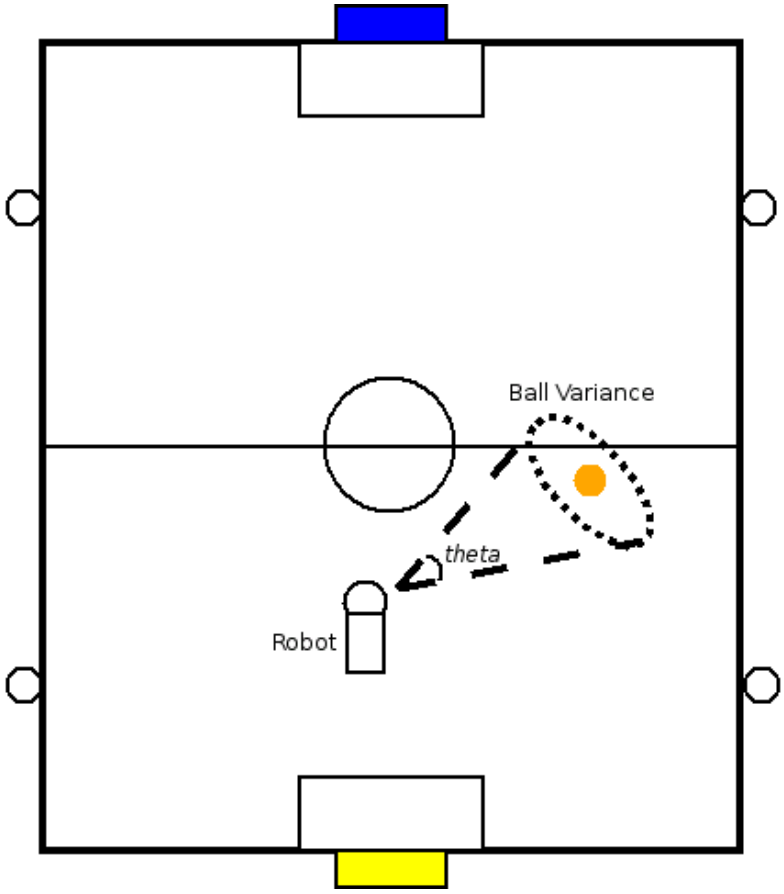

Figure 4.1: We scan around 1 standard deviation around the ball

We can then say something like, "'we are 95% sure the ball is within this circe on the field"', where the circle radius is equal to the maximum variance. Using the distance to the gps ball we can calculate the required angle of the scan needed to cover the 1 standard deviation circe centered on the global ball. This is illustrated in diagram 4.1.

Velocity Intercept

With the improvements to the localisation module this year, we have been able to greatly increase our ability to track both the position and also the velocity of the ball. This increase in accuracy has allowed us to be able to "'intercept"' the ball. In other words, instead of just running straight at where the ball is at the current point in time, we use the velocity information to approximate the optimal point to run towards to reach the moving ball the fastest. The first step is to approximate the movement of the robot, which we do by only taking into account

its forward and sideways motion, ignoring turn as it would make projecting the robot position forward in time difficult. Using the maximum forward and sideways speed of the robot we can approximate where on the field the robot can be in a given amount of time. Following this, we project the ball forward based on its current velocity, a frame at a time. For each frame forward we calculate whether the robot can reach the point at which the ball will be at that time in the given time. The first such position found is the position which the robot aims to reach, instead of the current ball position. If however there is no such position found, that is, either the ball is rolling too fast or too far away, we aim at the position which the robot can get closest to in the given time.

In addition to being able to get to the ball as quickly as possible, it is also important that once the robot reaches the ball it is in a position to effectively control it. In other words, the robot must be able to "'grab"' the ball, trapping it under its chin and being able to move it accurately. Through experimentation we've found that the most reliable way to grab the ball is when the ball is rolling towards the robot, such that the robot is facing the ball in the opposite direction of the ball's velocity. As such, we add additional sideways velocity to the walk proportional to the robot's distance away from the line which passes through the ball and is parallel to the ball velocity. This is illustrated in diagram 4.2. For more information on the algorithm used for intercepting a moving ball, see Ryan Cassar's Undergraduate Thesis[15].

4.1.2 Role Switching

One of the challenges of Robocup is being able to coordinate a team of 4 robots effectively. Having all of the robots running for the ball is not an effective tactic, since the robots will get in each others way, and also leave gaps in defense. Instead we only have one of the robots attack the ball, while the other two stay back and try to be in a good position to pick up the ball if it pops out of a scrum. The difficulty is in allocating each of the robots the most suitable role, and being able to do it quickly. If the role switching policy is not done well then a situation can arise where either the best robot for a particular role does not have that role allocated to it, or we can have a robot osciallting between two different roles, introducing stutter which slows down a robot significantly. This year we have built a layer on top of the old method of role switching to reduce the instance of role oscillation, as well as incorporating the ball velocity to

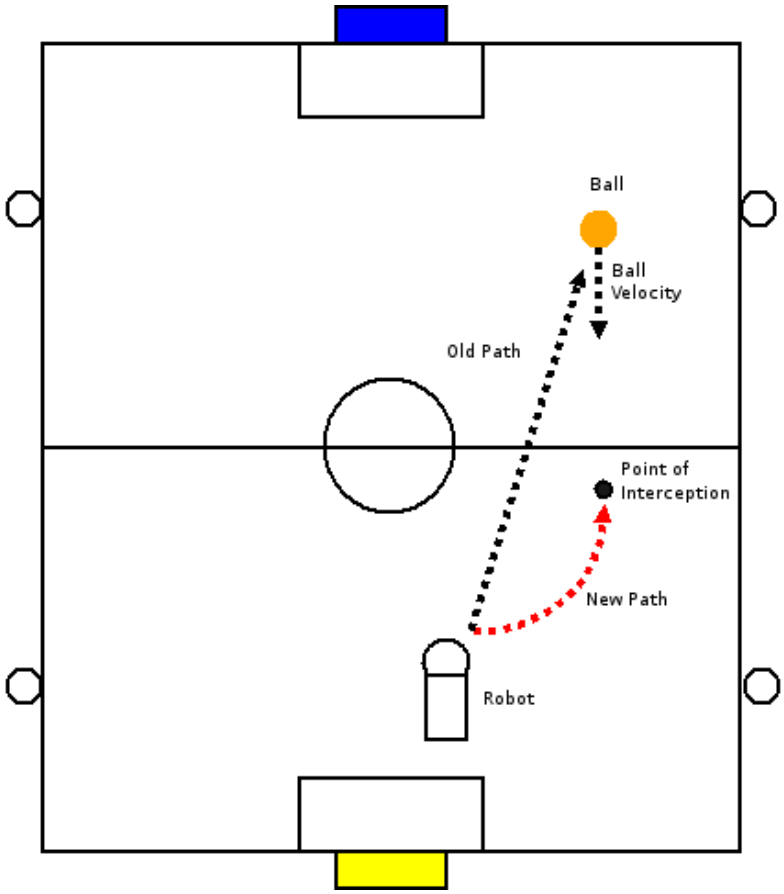

Figure 4.2: This illustrates how velocity intercept and velocity get behind alter the way a robot approches a moving ball

improve the performance of the role switching.

The role assignment for each robot is done based only on the information available to each robot at any point in time, that is, there is no "'captain"' robot which assigns all of the roles for the whole team. Instead, every robot broadcasts to its teammates information such as its estimated time to reach ball, its position on the field, and its current role. Using this data on every teammate, a robot can choose its optimal role, so for example, if it is the closest robot to the ball, it will choose to be an attacker. This layer of the dynamic role assignment is described in the "'Road To Robocup"' by Nobuyuki Morioka[4].

Since the role assignment depends largely on noisy inputs, such as ball position for example, we need to have some hysterisis and possibly filtering in the system to prevent role oscillation. In the past, this was done by giving a time "'bonus"' to a robot which was in a certain role, so for example, and robot which was currenty an attacker had a slightly artificially lowered time to reach the ball variable. However, this method of hysterisis can still be overriden by sufficient noise in the input variables. As a result, we have implemented an additional layer of hysterisis by keeping a history of roles assigned to the robot for the last N frames. We then assume the role which occurs the most times in the history, in a way we view the history of role assignment as a set of "'votes"', and the eventual outcome of the role assignment process if based on which role has the most votes. Using this framework we can then add different kinds of hysterisis very easily. For example, when a role switch occurs, say from Attacker to Defender, we can artificially add some number of "'votes"' to the role history for the new role. In this way we can impose effective restrictions on the maximum frequency of role changes. In addition to this form of hysterisis, we can also add a "'current role"' bonus. That is, whatever the current role is gets N extra "'votes"' when counting up which role we should choose next, regardless of the history. As a result, there is a kind of intertia in the roles, so it becomes slightly harder to change roles, hopefully introducing stability into the system.

Despite the new features of the role assignment this year, it was the module which hurt our gameplay the most. This is because of the implicit reliance of the system on a reliable wireless connection between teammate robots. At the competition venue, however, the wireless latency and reliability was very poor, none more so than in the final match against the NUBots. We encountered periods in the match where one of the robots was effectively stuck in a particular

role, such as Defender, and would not attack the ball despite being the closest robot. I believe this was due to the fact that a robot will effectively only switch roles if it receives a packet from a teammate. For example, if a robot grabs the ball, it will broadcast a time to reach ball of 0, indicating it is the closest robot to the ball and thus should be the attacker and the other robots should back off. However, if after this point the wireless goes down for a period of time, then even if the ball rolls away towards another teammate, that teammate will not realise it is now the closest one to the ball. The role assignment needs to be more aggressive, so to cope with poor wireless issues in future we may have to allow situations with multiple attackers.

4.1.3 Player Positioning

Player positioning refers to where on the field a player will try to place itself given the position of its teammates and the ball. So, for example, the Defender should try to position itself in a defensive manner between the ball and its own goal, while a Supporter should position itself close behind the Attacker and the ball such that it can quickly re-aquire the ball if the Attacker loses it. We spent alot of time this year experimenting with various role positioning schemes of various complexity[13]. One of the schemes which was thoroughly explored was using a set of preset positions for given situations, combined with linear interpolation. However this policy model did not seem to be effective and in the end we chose another extremely simple yet effective. We will describe both the Supporter and Defender positioning schemes very briefly, the Attacker positioning is not relevant since the attacker always runs for the ball, and the Goalie positioning is described in Ryan Cassar's undergraduate thesis.

Supporter Positioning

Our design for the Supporter positioning was intended to be such that when the ball was upfield, the Supporter could recover the ball if it is cleared from a scrum back towards our own goal. Conversely, if the ball is in our defensive half, the Supporter should be in a position to receive an upfield kick. As such, we implemented the following algorithm:

1. Define a "'Pivot Point"' on the field.
2. Calculate the normalised vector from the Ball to the Pivot Point.

3. Position the Supporter at the point 120cm from the Ball in the direction of the Pivot Point.
4. If the Ball is close to the far out lines, move the position closer to the centre of the Target Goal.

Defender Positioning

The Defender positioning is fairly simple, we position the Defender directly between the ball and the goal a given distance away from the ball. However, when the ball gets close to the goal, to avoid the Defender entering the goal box and thus being penalised for being an illegal defender, we add a sideways vector to the Defender position which becomes larger in magnitude the closer the ball is to the goal. This has the effect of pushing the Defender away so it does not become entangled with the Attacker.

4.2 Grabbing the Ball

Grabbing the ball is a fundamental skill, on which our entire gameplay style is based on. Our gameplay style consists of getting to the ball, grabbing the ball, controlling the ball, and kicking the ball. Many of the teams in the 4-legged league shun this style of gameplay, preferring to use paw kicks and head kicks to control the ball, as opposed to turning and running with the ball under the robots chin. The advantage of the grabbing style of play is that when the ball is under the robots chin, the other team cannot disposses our robot of the ball, as well as the fact that it is much easier to line up the robot and the ball in the correct direction to kick. The disadvantage of our style of play is that it heavily relies on being able to grab the ball quickly and reliably. This is a hard task, needing accurate vision to see the ball accurately to determine its distance and heading, a good localisation system to be able to track the ball and its velocity, and finally, the high level grab behaviour needs to take full advantage of these underlying systems. This year we have been able to significantly improve the grab due to two main factors, the improved ball position and velocity tracking as part of the localisation module, and the re-working of the high level behaviour.

4.2.1 Grab Approach

The Grab skill consists of several distinct phases. Firstly, when the ball is far away from the robot, we call sFindBall in order to get closer, this has the intention of closing in on the ball as quickly as possible, giving priority to speed rather than lining up with the ball. Once the robot is within some threshold distance of the ball, a close in approach is used instead. This is the most crucial aspect of the Grab. The close in approach must do two things, it must be fast and not introduce unnecessary hesitation and stutter, and it must also approach the ball in such a way that it is lined up with the robot in order for the Grab motion to be able to trap the ball under the head reliably. These two goals are in many ways contrary to one another. It is quite easy to come up with an approach policy which is very slow, yet very reliable, and conversely, one with is very fast but very unreliable.

The task of the Grab Approach is to provide a policy for the forward, left, and turn walk commands given the position and velocity of the ball relative to the robot. In 2005 this was implemented using something akin to a shallow decision tree, a series of ad-hoc if-else statements which partitioned the input parameters into a small number of categories, and decided the walk commands for each. The main disadvantage of this approach we found is that it involves a large number of parameters to tweak, as well as the fact that there are obvious discontinuities in the policy, that is, if the position of the ball relative to the robot changes by a small amount, it is possible for the resulting walk policy to change drastically. The ultimate result of this is stutter in the Grab Approach, and thus lower relibility.

This year we implemented a very simple, very effective approach policy. In essence, the sideways walk component is always 0, and the turn component is equal to $0.9 * relativeBallHeading$. The forward component calculation is a little more involved. Essentially, the forward speed component is $maxForward$ if the ball is straight ahead. Otherwise, the forward speed is lowered as the $relativeBallHeading$ gets further away from center. Firstly, we modulate the forward speed by $cos(relativeBallHeading)$, so the more misaligned the ball is, the slower the robot walks forward. We refined this process even further by making the forward speed more sensitive to the ball being misaligned the closer the ball was to the robot. This is done by multiplying the $relativeBallHeading$ by a value which grew larger the closer the ball was. These features

resulted in a very robust policy which could both reliably and quickly approach the ball, and had the added benefit of having relatively few variables to tweak.

4.2.2 Grab Trigger

The Grab Trigger determines when the robot should go from the Grab Approach into the Grab Motion. The Grab Trigger relies hevily on the accuracy of the vision and localisation modules to provide data on the position and velocity of the ball. If the Grab triggers too soon, the ball will be knocked out of the way by the tip of the head, and if the Grab triggers too late the ball will hit the chest of the robot and bounce off. In previous years only the position of the ball was considered, with the velocity ignored due to the poor accuracy of the filtered ball position and velocity. This year we use the filtered ball position and velocity as much as possible.

The basic algorithm involves defining an area in front of the robot in local coordinates which will trigger a grab if the ball enters it. We then project the ball position forward in time by up to 6 frames, and if the ball is estimated to enter the trigger region during this time the Grab Motion is triggered.

4.2.3 Grab Motion

The Grab Motion is a prescripted sequence of movements the robot performs, mainly with the head joints, in order to trap the ball between its chin and chest. We break up the action into 3 distinct sections. For the first 3 frames of the Grab Motion, the robot walks forward with the head slightly forward but still higher than ball height with the mouth closed. Next for 3 frames the robot pulls its head down, with mouth still closed but the pan is set so that the head is aligned with the ball if it is off-center. Finally, for the last 6 frames the robot opens its mouth to lock the ball under the chin, and straights the head pan joint to center. The current state of the motion is determined by a counter which is incremented every frame. We can speed up the Grab Motion by incrementing the counter twice if we detect the ball is close to the robots chest using the Infra-Red sensor. This goes a long way to preventing the ball bouncing off the chest due to a miscalculation of the velocity.

4.2.4 Results

Our method of evaluation involved counting the number of attempted grabs versus the number of successful grabs as seen in the final match at Bremen against the Newcastle Team. The number of successful grabs made by rUNSWift was 35 in the first half and 4 failed. The number of successful grabs made by the NUBots in the first half was 31 and 12 failed. This shows that our grab is extremely reliable and strong. What let us down in the final game was our role assignment and dodging around opponents while the ball is grabbed. But the grab itself was very reliable and quick.

4.2.5 Conclusion

The Behaviours this year have improved significantly, specifically the find ball behaviour and the grab behaviour. The first has allowed has to be the first to the ball on most occasions, making up for our slow walking speed, while the latter has allowed our team to be able to control and keep possession of the ball effectively. This was instrumental to our success at Robocup this year.

Chapter 5

Evaluation

In this chapter, we discuss the results of the Robocup competition, and the effects that the changes discussed previously in this thesis have had on these results.

5.1 Results

This year rUNSWift has achieved great success, winning the Australian Open and coming runner up in the World Open, losing to a fellow Australian team from Newcastle in the final. Overall this is an improvement over the previous year which was placed 3rd at the World Open.
At the Australian Open we defeated Griffith in our opening game 10-0. It was difficult to derive any useful information from this game since it was so one-sided. Our 2nd game was against the Newcastle team, which was runner up in 2005 at the World Open, and so we expected a very tough match. The final score was 2-1 in favour of Newcastle, so our expectations were well founded. However, we noted that in the second half, 2 of the robots were in low gain mode, which meant that they walked very slowly and could not effectively contribute to the game. This bug was noted and fixed for the final game. The finals were against Newcastle again, and again we expected a very tight game. This time we won 2-0, partly due to the fact that our robots did not go into low gain mode. The other major contributing factors to our victory was our speed to reach a moving ball, which was due to the effectiveness of the velocity intercept code and the ball velocity tracking, both of which were introduced this year. The reliability and speed of the grab was also contributing factors, despite the fact that the grab was still a little

bit less effective than that of Newcastle.

The World Open posed many additional challenges to the Australian Open. We had played very few games outside of our lab, and so we anticipated that adapting to a new field would pose a big challenge. When we arrived a the competition venue we soon discovered that our walk was extremely ill suited to the surface of the field. The robots were slipping quite severely, and as such our walk ended up being one of the slowest of all of the present teams. Surprisingly however, despite this fact, our major advantage throughout the competition was our speed to re-aquire and reach the ball. This was mainly due to the velocity intercept skill and the find ball skill, both of which benefited from improved tracking of the balls position and velocity, the find ball skill in particular benefitting from the distributed nature of the filter. A robot could be very certain of the position and velocity of the ball, even if it was occluded, because another robot on the team could see the ball and transmit its observations via wireless. In addition to this, the field surface actually benefitted our grabbing style of play. This is because the carpet was fairly thick and so the ball stopped quickly and rolled predictably. This meant that the grab became suddenly much more reliable when compared to that on our very hard and flat home-field. In the competition our level of play was far above that of any other team, excluding Newcastle who have a very similar style of play.

First Round Robin	
rUNSWift vs FC-Twaves	9:0
rUNSWift vs TJ-Ark	9:0
Second Round Robin	
rUNSWift vs Wright Eagle	5:1
rUNSWift vs Asura	9:1
rUNSWift vs Eagle Knights	10:0

Quarter-Final	
rUNSWift vs Upennalizers	8:0
Semi-Final	
rUNSWift vs German Team	6:0
Grand Final	
rUNSWift vs NUBots	3:7

Our ability to aquire and control the ball meant that we had much of the possession, possession which we used effectively. In stark contrast, most of the other teams forewent the grabbing style of play and as a result were unable to effectively control and keep possession of the ball. We reached the final only having 1 goal scored against us up to that point (we did however also score an own goal). At the semi-final stage we defeated the 2005 champions the German Team 6-0, which shows the gap between the top two teams and the rest of the teams in the competition.

The final match against Newcastle again promised to be very close and entertaining. The first half ended with Newcastle leading 2-1, despite our team almost scoring an equalising goal seconds before time ran out for the half. During the second half we experienced severe problems with dynamic role switching. We knew beforehand of the existance of a bug whereby a robot will stay in a Defender or Supporter role and not switch into Attacker despite being the closest robot to the ball. This results in said robot constantly backing away from the ball. This bug rarely manifests itself, but when it does, it stays with a robot until it is rebooted. When it happened to one of our robots, our gameplay essentially fell apart, allowing Newcastle to score several quick goals. We called a time-out with 2 minutes to go in the match, during which we rebooted all of the robots. This "'fixed"' the role switching bug, resulting in our team scoring 2 very quick goals with out gameplay back on par with that of Newcastle. However in the end Newcastle won 7-3, winning the 4-legged league competition for the first time.

5.2 Discussion

Our performance in both the World Open and the Australian Open this year has been very good. Early development of the rUNSWift system this year was hampered somewhat by our inability to play practice matched due to not having enough working robots. It is also interesting to note that it appears as though it is getting harder and harder to improve upon last years code, suggesting that the limits of the current hardware platform are being reached. This year, we have only won 2 practice matches against our previous best code, both of which happened very close to the respective competitions. We defeated the 2005 rUNSWift code in a practice match 1 week before the Australian Open, and we beat our Australian Open code only a week before

leaving for Bremen. This tells us that we needed for all of the developments we were working on to come together before we could beat the previous best code. Incremental improvements are harder and harder to come by.

Chapter 6

Conclusion

In this thesis, we have shown that robot localisation can be improved dramatically in the 4-legged Robocup domain using a multi-modal, distributed Kalman Filter. This algorithm allows the localisation system to be more robust to noise, reduce intrinsic errors due to such processes as linearisation, and be able to track the ball more effectively due to the ability to incorporate teammate observations. In addition to the localisation module we have also shown how various changes to the high level behaviours have helped rUNSWift win the Australian Open and come 2nd in the world in this years Robocup competition.

6.1 Future Work

This year we have made great improvements to the Vision and Localisation modules. Much of the high level behaviours, however, remain very ad-hoc, with no reasonable explanations existing as to why we went with a certain policy rather than another. Probably the greatest area of improvement in our current system is in the Locomotion and Behaviour modules. One suggestion is a greater use of Machine Learning algorithms in the high-level behaviours, possibly using offline tools such as a simulator to speed up development. Our wlak speed was fairly low comparatively this year, but in the end it was not the reason why we lost to Newcastle in the final, and in the preceeding matches we demonstrated that our slow walk was not a significant factor. We comfortably beat teams which could walk up to 15-20cm/s faster than us.

Bibliography

[1] Robocup Technical Committee, *Robocup Four Legged League Rule Book* (http://www.tzi.de/4legged/pub/Website/Downloads/Rules2006.pdf, 2006).

[2] Jin Chen, Eric Chung, Ross Edwards, Nathan Wong, *Rise of the AIBOs III - AIBO Revolutions* (Undergraduate Honours Thesis, University of New South Wales, 2003).

[3] Derrick Whaite, *Thesis B Report* (Undergraduate Honours Thesis, University of New South Wales, 2004).

[4] Nobuyuki Morioka, *Road To Robocup 2005: Behaviour Module Design and Implementation, System Intergration* (Undergraduate Honours Thesis, University of New South Wales, 2005).

[5] Rudolf E. Kalman, *A New Approach to Linear Filtering and Prediction Problems* (Transactions of the ASME–Journal of Basic Engineering, 1960).

[6] Greg Welch, Gary Bishop, *An Introduction to the Kalman Filter* (SIGGRAPH, 2001).

[7] Raul Lastra, Paul Vallejos, Javier Ruiz-del-Solar, *Self-Localization and Ball Tracking for the Robocup 4-Legged League* (Department of Electrical Engineering, University of Chile, 2005).

[8] Michael Quinlan, Steven Nicklin, Kenny Hong, Naomi Henderson, et al, *The 2005 NUbots Team Report* (http://www.robots.newcastle.edu.au/publications/NUbotFinalReport2005.pdf, 2005).

[9] Rudy Negenborn, *Robot Localization and Kalman Filters, On finding your position in a noisy world* (Masters Thesis, Institute of Information and Computing Sciences, 2003).

[10] Thomas Rofer, Tim Laue, Michael Weber, et al, *German Team 2005 Report* (http://www.germanteam.org/GT2005.pdf, 2005).

[11] Ioannis Rekleitis, *A Particle Filter Tutorial For Mobile Robot Localisation*

[12] Andrew Owen, *The 6th Sense: I see red people, as balls* (Undergraduate Honours Thesis, University of New South Wales, 2006).

[13] Eric Huang, *Road to Robocup 2006, rUNSWift 2006 Behaviors and Vision Optimizations* (Undergraduate Honours Thesis, University of New South Wales, 2006).

[14] Michael Lake, *Turning heads at the World Cup and making our mark with field line localisation* (Undergraduate Honours Thesis, University of New South Wales, 2006).

[15] Ryan Cassar, *Four Legs and a Camera: The longest of journeys starts with the first step* (Undergraduate Honours Thesis, University of New South Wales, 2006).