Rise of the AIBOs III - AIBO Revolutions

Jin Chen 2277936 Software Engineering
Eric Chung 2272934 Computer Engineering
Ross Edwards 2252733 Software Engineering
Nathan Wong 2273999 Software Engineering

Bernhard Hengst, Claude Sammut, Will Uther

4th November 2003

Contents

Chapter 1

Introduction

1.1 Abstract

In 2003, the UNSW United team in the Sony legged robot league was successful in claiming the title of world champion in the Robocup 2003 competition taken place in Padua, Italy. In this report we present a description of the system that has controlled our team of quadruped robot that has competed sucessfully against other teams from a wide range of institutions around the world. A complete rewrite have taken place in most modules that made up the system between this year and last year, and drastic changes have occurred in those modules that was not subject to rewrite. An important part in the development of our system was to study its performance under actual competition conditions by playing many practice matches. In this report, we will also detail the changes from previous years and the reasons that prompted us to change.

1.2 Background

The Sony legged league is one of five leagues of the Robocup competition. UNSW has participated since 1999 and have achieved runner up in 1999, 2002 and title of champion in 2000, 2001 and 2003.

Each match is played out in two ten minute halves. There are four robots in each team, and one of the robots must be designated as goalie. The winner is decided by the number of goals scored. In the event that both teams tie with same number of goals, a penalty shoot out that tests the fundamental individual

skills of each robot is played to determine a winner.

The robots used in the legged league are ERS-210 model Sony entertainment robots. In the legged league, physical modifications to the robots are disallowed, so that the sole difference between teams is in their software system.

1.3 Changes in Competition Field and Game Rules

1.3.1 Goal Box

The area where the goalie is given priority and illegal defenders are penalized has been reduced from the previous years' strip of field before the goal to the form of a goal box. This means defenders can now enter the bottom corners of their halves and assist in the defense of the goal without being called for illegal defending.

1.3.2 Goalie Charging

The goalie charging rule was changed this year by increasing penalty by a substantial amount and tightening the conditions that triggers the offense, to discourage goalie pushing. For the majority of the year, any physical contact with the goalie is forbidden and the offending robot is removed from the game for sixty seconds, and replaced at the half way mark at the end of that. As a result much of our strategy was focused in attacking the goal from a distance.

Nearing the competition, the rule was relaxed, and it was decided that the rule was to be changed from forbidding any physical contact to allowing physical contact if the attacking robot is in possession of the ball, and without ball possession the attacking robot is permitted two seconds of contact with the goalie. Any additional attacking robots that contacts with the goalie is immediately removed from game for thirty seconds.

1.3.3 Illegal Defender

Another rule designed to remove congestion from the goalie box was the illegal defender rule. In previous years the defender that enters the defensive zone of its own goalie was removed and placed at the half way mark of the field. This

year, a much heavier penalty has been applied in that offending defenders are removed from the game for sixty seconds, and then to be replaced at the half way mark of the field.

1.4 System Architecture

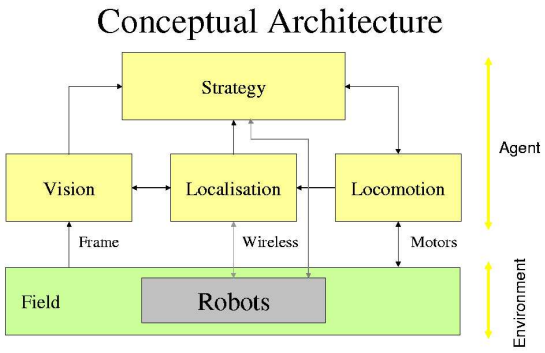


Figure 1.1: Conceptual Architecture

We have completely revamped the system architecture from previous years, which was monolithic in structure. The major problem that prompted this change was performance degradations as a result of wireless network congestion. The monolithic structure has the rest of the system tightly coupled with wireless communication modules, so that poor performance in wireless, such as in the case of congested wireless networks, will slow down the execution of the rest of the system. In addition, the previous structure was difficult to scale, and was not friendly to collaboration between multiple contributors. Conceptually, the architecture remains similar, see Figure 1.1. However, we have broken the structure into five modules, Behavior, Vision, Localisation, Locomotion and Wireless Communication, see Figure 1.2.

1.5 Development

We began development on the robots by first porting all our code to the new version of OPEN-R, which is the environment that our code is executed under on the robots. At the same time we perform the port, we also separate the code into the new modularized structure as forementioned. Following complete porting and separation out of the infrastructure modules for vision, localisation,

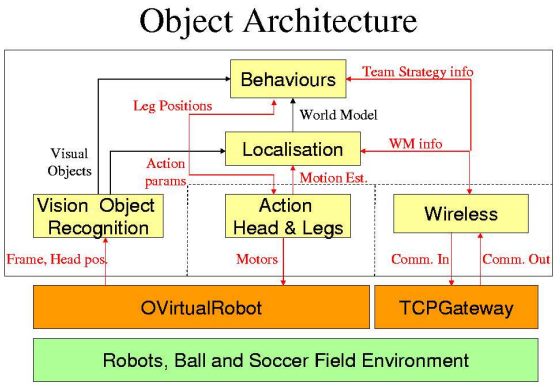


Figure 1.2: Object Architecture

locomotion and wireless, we began to play weekly matches between the new code and last year's code. From these matches, we evaluate the advantage and pitfalls in the robot's behaviour and apply fixes or improvements in the corresponding module. Once we were able to defeat last year's code consistently and convincingly, we assign the winning code the title of champion, and all subsequent matches were then played against the champion. And the champion code is replaced by the new code when it in turn has been beaten consistently and convincingly. These practice matches were vital to our development as they are the only way to evaluate whether certain ideas and strategies really do work. For skills that seem to be powerful and advantageous when tested with one robot could break down when confronted with opponents, and this year there has been more subtle adjustments than previous years, that add sum up to a considerable advantage overall. As we drew closer to the competition, practice games were held on a daily basis.

Chapter 2

Vision

2.1 Introduction

The RoboCup field and all objects on it are one of nine distinct colours. For real time performance in visual processing, we rely almost exclusively on this colour information for object recognition. Edge detection was developed this year, but its resource consumption was too high to offer real time performance, only a minimal portion of the system is executed during the competition.

The vision module is at the lowest level of all modules. It is the robot's main sensor of the outside world, and as such it plays a vital role in supplying information from which the behaviors of the robot is deduced. Hence the performance of the vision system is crucial.

The goal of the vision system is to be able to recognise objects on the field correctly, these include the recognition of beacons, goals, robots and ball. Currently this is achieved by feeding training data about the colours of objects into a learning algorithm. The algorithm outputs a colour lookup table that contains the learned colour of each Y U V value. The table is stored in the robot's memory, and on every camera frame the robot translates the camera image into colours it recognizes by using the lookup table. Like pixels are grouped to form blobs, and the blobs are analyzed to form objects. It is also necessary, to determine from vision, the distance and heading of the various objects. The methods for each object vary. Some rely on geometric transformations based on the image position of objects whilst others rely on calibration of object properties.

2.2 Colour Segmentation

Starting in 1999 and until half way through the 2001 competition, a Polygon Growing Algorithm (PGA) was used in the training of coloured samples. During the finals of 2001 it was found that Quinlan's C4.5 decision tree learning algorithm performed better under the given lighting conditions. In 2002, a Nearest Neighbor algorithm was used in place of the other methods because it was more tolerant to errors during the calibration process, so that small calibration errors still allowed for reasonable performance, and thus does not need the entire calibration process to be repeated, which is time consuming. It was also found to be more efficient in its execution because it constructs a lookup table instead of the decision tree generated through C4.5.

The Nearest Neighbor algorithm was used during 2003 up until several weeks prior to the competition. As early as following the 2003 Australian Opens, we have come to realize that the Nearest Neighbor algorithm does not adapt well to lighting changes. The Australian Opens calibration was performed prior to the competition in ideal lighting, and it was a calibration that had been tested and found satisfactory for we had conducted our development on it up until the Australian Opens. And perform satisfactorily it did during the initial pool matches and in the semi-finals. However, during the finals of the Australian Open, the robots behaved erratically due to the change in lighting condition as a result of crowded spectators around the competition field, the severe performance degradation prompted us to search for other methods. Prior to the Australian Opens, a brief attempt was made to solve lighting sensitivity issues by experimenting with a different colour segmentation scheme that allows a single point in colour space to take several values, and the most appropriate value determined during run time. However the significance of the problem was not realized until after the Australian Opens, and so the project was terminated. After the Australian Opens there was not enough time to develop an alternate colour segmentation scheme and conduct thorough tests to convince ourselves of its reliability. So we turned to existing methods that have proven to work. Due to time limitations, we had only time to experiment with one other method, which is what was used in the finals of 2001, C4.5. The results indicate that C4.5 was more robust to lighting changes. We evaluate this by switching off sets of lights that were on during the calibration process and analyze both the behaviors of the robots and their segmented images under the lighting change in the environment. A possible explanation in the better performance of C4.5 is that the Nearest Neighbor algorithm may suffer from over fitting. The algorithm only expands samples that have been classified, and fails to generalize into regions of colour space that was absent in the training data. It still does extend into neighboring colour spaces, so it can still function under subtle lighting changes that normally occur around the field, but not noticeable by human eyes. However, the generalization still leaves much of the colour space unfilled, and thus when a lighting change of severity noticeable by human eyes occurs, it fails, as in the case with the finals in the Australian Opens. C4.5 is better at generalizing into color space that has not been represented in the training data. In fact, since C4.5 simply cleaves the colour space according to the training

data, if we leave it unchecked every point in the colour space will belong to a colour. Thus introducing a potential danger, in that noises are given meaning from over generalization. To prevent this, we introduce a placeholder colour to hold the regions in colour space that are noise, and rely on it to prevent C4.5 from labelling noises as information.

To create the decision tree, we supply two files to C4.5. The first is a names file that defines the parameters and the values that they can take. The second is the data file containing all training data. The output of C4.5 is a decision tree, which we parse into C++ code. All possible combinations of colour values are then fed through the decision tree to produce a look up table identical to what was used in 2002. A full look up table of YUV each with 256 possible values for each Y, U and V would require a storage size of 16MB, far too large for the robot's memory. We compress the number of possible values for Y, U and V to 128 possible values, and thus reducing the size to 2MB. We lose information in that a 2x2x2 region in colour space has been compressed into a single point, however points occupying similar regions share similar colour anyway so we do not lose much information. An area where this compression have performance impact is in the boundaries between colour regions, compression would mean blending and blurring of these boundaries that could contribute to the noise in the biundary of objects. Yet this trade off is neccesary so that the entire table can fit inside the robot's memory, and colour segmentation can be performed through table look up rather than through a decision tree, and thus saving caluable processing time on the robots.

2.2.1 Calibration Images

Before we start to capture training data, it is important to first settle on a camera white balance setting that is best at separating out the various colours. We do this by taking sample images and have their YUV values compared. We have found white balance settings to impact greatly on the outcome of colour segmentation. Colours that are difficult to distinguish on a particular setting could be well separated on another setting. There does not seem to be specific rules that we can follow, and generally we experiment with all settings to see its effect, as their effects are rather unpredictable. In general, we begin with fluorescent setting if the environment is subject of many fluorescent lights, such as in our robotic laboratory. We have found the indoor setting to be ideal for dim lighting, and outdoor setting to be ideal under strong lights. Although the competition games are played indoors, we have found outdoor setting to out perform the other settings due to the brightness of the standard light set up for the competition. The outdoor setting was used in the competition at Padua and fluorescent is used in the laboratory. We usually stay with high shutter speed to reduce motion blur, however we have found a slower shutter speed produces noticeably lower noise levels, and that is the preferred setting in circumstances where there is limited time for calibration, such as for demonstration purposes. In Padua we have found the shutter frequency under fast setting to coincide with

the frequency of power supplied to the lights, thus introducing strobes across our images. The effect was eliminated by using mid shutter speed, however we have found the strobing to impact minimally on object recognition, and so we stayed with fast shutter speed for it is with the level of motion blur under fast shutter speed as assumption that our object recognition system is developed. It may be possible to lower our shutter speed back to mid because it offers fewer noises, and better image qualities. However quite severe motion blur can take place under such a setting and we have not been successful in dealing with motion blur of such severity.

The gain setting of the camera was found to be least significant in image quality. There does not seem to be much advantage in shifting between the three. Both low and mid produce comparable images, calibration obtained from images taken from these settings can be used interchangeably without large impact in our laboratory. High gain produces significantly brighter images. Mid gain is the setting we use.

We try to capture as much of the visual representation of objects as possible in the training data. The data is captured by taking still images of various objects at varying distances, orientations and set up from the robot's camera in 176x144 resolution. Heavier emphasis is placed on the ball since it can be at any position on the field and it is subject to heavy shadowing from both other robots, and the robot itself. See 2.1 and 2.2 for a standard listing of the calibration images. The standard set used in 2002 was found to be insufficient, and the new complete set of 41 images almost double the 25 images used in previous years. Specifically, much heavier emphasis was placed on problems such as distinguishing field green and beacon green, phantom ball in red robots, phantom ball in yellow goal, phantom ball in shadowed boundary, ability to recognise occluded balls such as those in possession or in scrums and the change in colour of shadowed objects. Field green is only classified in the images 13-17, to avoid over sampling from affecting beacon green. This is a much bigger problem when using Nearest Neighbor algorithm, however we find that C4.5 is also affected, though to a smaller extent.

However, that too was not enough, and performance degradation due to crowds was still witnessed during our quarterfinal match in the competition. As a last minute patch, we took calibration pictures during the half time break of the semifinals match when the field was still crowded, the pictures were then integrated into the classification by manually overriding what is learned with a manual over learning tool.

2.2.2 Calibration Process

Calibration is performed manually by labelling pixels in an image to their colour class, see Figure 2.1. There are various utilities in the calibration tool to semi

No	Setup	Purpose
1 - 6	Close up of each beacon	Large samples of beacon colours
7 - 12	Mid range beacon (approximately 1.5m away) with ball near the beacons	Distant beacon colours, ball colour in different sections of the field, as it is common for lighting to be non-uniform
13 - 16	Two additional mid range images for each middle beacon	Extra image to calibrate between beacon green and field green
17	Green field with robot head close to maximum negative tilt	Calibrate shadows on the green field, as shadows often appear under the ball and under robots
18 - 19	1m from Goals, with ball 30cm in front of goal	Goal colours, fringe colours between ball and goal
20 - 21	Robot paws on the goal line looking into goal, with ball inside the goal	Closeup goal colours and ball colour under goal reflections
22 - 25	Red and blue robots in goalie position in front of their respective goals. Take pictures of both front and side view of robots.	Robot colour, fringe colour between red robot and goal
24 - 27	Red and blue robots holding ball, both front and side view	Robot colour, ball colour while in possession by another robot
28	Ball under chin, similar to last phase of ball grabbing action	Ball colour while in possession
29	Ball in scrum, surrounded by many robots. Ball almost entirely shadowed	Ball colour while under heavy shadowing
30 - 31	Close ball touching white boundary, and close ball on the slanted corners	Fringe colour between ball and boundary, also the colour of orange reflection and orange shadow on the white boundary
32	Floor beyond the boundary	Prevent robot from being mislead into seeing anything meaningful when looking over the boundary

Table 2.1: Calibration images

Additional images for competition		
33 - 38	Beacons before back drop of Referee Pants, may also want to cast additional shadow on beacon	Automatic white balance in the camera causes beacons to appear differently with different background colour, the shadow of referees as they move about the field also affects beacon colour
39 - 40	Heavily shadowed goals	Prevent illusions in shadowed goals, also maintain goal recognition when there is a scrum, since referees have a tendency of leaning over the goal
41	Referee pants on the field	Prevent illusions in the referee pants

Table 2.2: Additional calibration images for competition

automate the process of manual classification, such as flood filling adjacent pixels that are of configured Manhattan distance in colour space. The user labels pixels in the image with colours as deemed best by the user. The guideline is to label everything we feel the robot should definitely see. Ambiguous areas are left for the learning algorithm to decide, and areas that we are adamant should not belong in any of the interested colours are labeled as the placeholder colour to prevent any labeled colours from generalizing into them. The quality of the calibration ultimately lies in whether the calibrator has correctly distinguished object colours and noise. However, since there are few clean cut cases, and it is generally a case of deciding whether the pixel is more likely to be object colour than noise through the course of the game, the correctness of these decisions are largely experience dependent. And it has been neccesary for us to assign team members to specialize in this area.

Nearest Neighbor

The technique is a straight nearest neighbor algorithm. The first step is to mark all classified colours from the training data on to their respective coordinates in colour space. Because a single coordinate can be marked multiple times and in different colours, we first tally the number of times each coordinate is labeled by each colour. The coordinate then takes the label of its highest tallied colour. However, this also introduces the problem that a large sample of one colour can completely oust another smaller sampled colour on overlapped coordinates by shear sample size. Note that this is a flaw in the implementation of the algorithm and must be worked around in classification by ensuring similar sample size for each colour. The next step is to generalize these marked coordinates into the unmarked regions in colour space. Each unmarked coordinate will find the

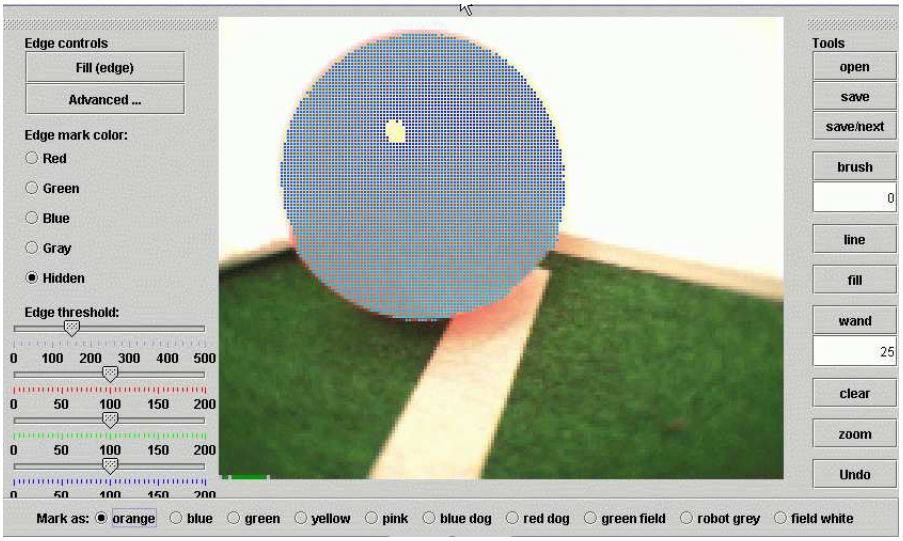


Figure 2.1: Calibration tool, with a classified orange ball

colour it is closest to, and if its Manhattan distance lies within a set threshold, the coordinate then gains the colour label of its closest neighbor. The distance threshold varies from colour to colour, depending on their significance and properties. The result is many coloured blobs in colour space. Blobs of like colour should be joined, and outliers are likely to be calibration error.

C4.5

Two text files with extensions *stem*.names and *stem*.data needs to be created from the training data. The names contain specification of the attribute and class, by listing the possible classes and the values taken by their attributes. We use the competition colours as classes and the YUV coordinates they occupy in colour space as attributes. Thus our names file takes the form:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Y: continuous

U: continuous

V: continuous

The data file contains the training data. It is a list of all the labeled pixels in all the labeled images, and maps for each labeled pixel their coordinate in colour space to their labeled colour. The first three values are attributes and the last is the class these attribute belong to:

.

.

.

12, 34, 56, 7

23, 45, 67, 8

34, 56, 78, 9

98, 76, 54, 3

123, 234, 56, 7

.

.

.

C4.5 then takes these two files and generate two decision trees, a full decision tree and a simplified decision tree. We have experimented with both and found no noticeable difference in performance. We use the simplified decision tree. The decision tree is then converted into a decision tree in C++ code, and from it we generated a colour look up table.

There are various built in options in C4.5 that we have not experimented with. We have not deviated from the default settings in all our runs of C4.5. It is possible that the colours can be better separated under different C4.5 settings.

Override Learning

Both Nearest Neighbor and C4.5 on their own does not produce classification that is entirely satisfactory for our purpose, though through no fault of its own. As fore mentioned, we leave pixels that may belong to more than one colour to the discretion of the learning algorithm. However, for competition reasons we may be more interested at a particular colour than others, or that we may want a finer balance between them. For example, beacon green is much more important than field green, and we would much rather having some beacon green on the field rather than the other way around. A more severe case of overlapping colours is ball and red robot. As the robot approaches the ball, and more of the ball comes under its chain, the ball is almost entirely red through the robot's camera. If we classify these as orange, then red dogs on the field will start to appear as balls, and the robot will give chase. If we classify these as red, the robot will suddenly lose track of the ball and find a red dog in its place when very close to the ball. Thus a very fine balance is necessary under the current colour scheme, and that balance is achieved through manual manipulation of the learned colour table.

We developed a tool that takes an image and shows it as colour segmented by the robot, see Fig 2.2. The user is free to make direct corrections to the colour table, and see the effect of their changes. In allowing direct manipulation of colour tables, the tool is potentially dangerous and care must be taken. Although it offers the user enormous power, the outcome of the tool correlates directly with the skill and experience of the user.

This is really avoiding a fundamental problem in the current colour scheme, in which we assumes that each point in colour space will only belong to a single object. However, this is not true due to varied lighting over the field, shadowing from robots and referees, and digitization errors and edge distortions in the robot camera. As a result, in attempting to label all occurrences of a colour within a particular object, we are also labeling other noises around the field to be that object. And often we find that some areas of different objects will share identical colours, such as a shadowed ball and red robot, shadowed yellow goal and ball, field green and beacon green, and edge distorted beacon blue and beacon green.

2.2.3 Maybe Colours

In the beginning of the year, a brief attempt was made to tackle the problem by splitting colours into 'core colours' and 'maybe colours'. This relies on the assumption that for each object, it is possible to obtain colours that only represent that particular object on the field, which we call core colours. The rest of the colours that make up the object can also belong to other objects or noise, and those we call the maybe colours. And we define some relationship between

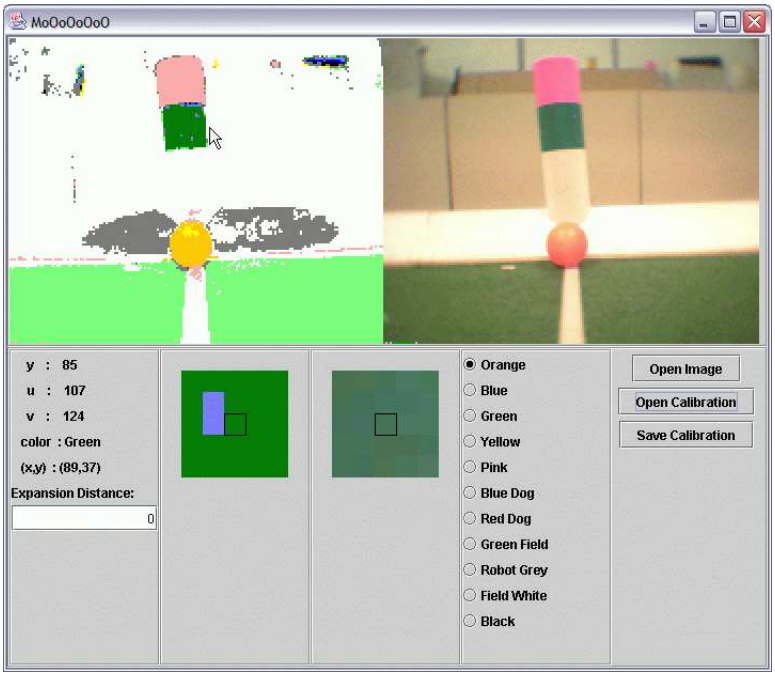


Figure 2.2: Manual Classification Overriding Tool

the two so that we will only take the maybe colours into consideration if it is part of the objects and ignore those that are noise, or are a part of another object. Essentially, the maybe colours are the regions in colour space that span more than one object, including noise, and the core colours are what we rely on in deciding which of the represented colours in the maybe colour region should be taken as the labeled colour.

The concept was never entirely implemented, however we did implement a crude and simplified version, in which eight additional maybe colours were introduced, one for each corresponding colour on the field. However each maybe colour can only be either one of two colours - the core colour that they represent, or background colour. In other words, it does not cater for overlapping of colour representation between objects, but only overlapping between objects and noise.

Core and maybe colours should have been separated statistically. By analyzing the classification of training data, we should be able to conclude which colours are confident enough to be core colours and where there is overlap then we assign them as maybe colours. However, due to the time constraint in this experiment, as it is necessary to commence development on other areas of the robot, we skip over the analysis tool by classifying both core and maybe colours ourselves, and the nearest neighbor algorithm is then ran over the classified data for generalization. Rather, the nearest neighbor algorithm should have been modified, by assigning regions that have been expanded into by several

colours as maybe colours.

The blob formation routines on the robots were modified to merge blobs of maybe and core colours. Blobs of both maybe and core colours are formed normally from runs of like coloured pixels, and we let the colour of maybe colour blobs be determined by the core colour blobs. Since our maybe colours can only be either their corresponding core colour or background, we decide by simply seeing whether there are any adjacent core coloured blobs. The assumption is core colour will not appear on anything else other than the intended object itself, so any corresponding maybe coloured blobs adjacent to the core coloured blob must be of the same colour, and the two are merged. In the event that no adjacent core coloured blobs are found, the maybe colour blob is then discarded as noise.

The result of blob formation is passed to object recognition routines, which was not changed. In the beginning, maybe colours caused a drop in the robots frame rate because of extra load on the blob formation algorithm. However, this is rectified by a change in the algorithm and it could be executed at full frame rate. Using this scheme, the robots were able to recognise more objects, including distant beacons that were often missed and distant robots. On the other hand, although object recognition seems to have improved, the distance calculated for those objects were often wrong. We thought this to be the cause of new blob sizes, so we recalibrated distance measurements using maybe colours, but found no improvements. Distance calculations were based on blob properties, however the C-Plane showed correctly colour-segmented images, so this should suggest that an error exists in the blob formation algorithm. But that was not thought of at the time, and the maybe colour scheme was terminated, because there were many behavioral problems in other aspects of the robot.

It wasn't until later that a bug was discovered in the blob formation in which under certain run configurations blobs were not formed correctly. The problem was especially apparent in images where blobs are formed by many splitting and joining runs, such as a very close shadowed ball where the orange blob contains many runs of noise in robot red. This error would be magnified in the maybe colour scheme, where there are many runs of maybe and core colours intertwined with each other.

Development in this area had ceased very early in the year, it would be interesting to see its performance now that the blob formation bug has been corrected, and to see whether the idea offers any real potential that is worth investing effort into.

2.2.4 Testing

The first test is to see a live C-Plane feed of the robot as it plays on the field. Any problems in this stage would indicate that there is a large problem with the calibration, potentially due to bad camera settings, or errors in the calibration process. At any case, the calibration should be redone from the beginning.

The second phase is to run the calibration in a normal, while capturing a log of all the images taken from the camera. This is a necessary step because human eyes are not fast enough to follow every frame of the C-Plane, and small errors such as one or two frames of unbounded objects and/or phantom objects are easy to escape our eyes. Phantom objects impact greatly on the performance of robots, especially objects with high priority in decision making such as ball and goal. This step is important also because it captures images that have been affected by motion blur as the robot moves around in the game, which is important to testing because motion blur changes certain colours and create problems, such as blurring portions of red dog in front of yellow goal into orange.

The robot is able to send a stream of captured image over the wireless network. Not every image can be sent due to resource restrictions, depending on resource availability it typically transmit one every three or four frames, which is sufficient for our purpose. The captured log is then stepped through an offline vision system. The offline vision system is an offline port of the robot's vision system, see Fig 2.3. Errors found at this stage can either lead to recalibration, or manually overriding the trained calibration, depending on severity and nature of the error.

While our goal is to obtain perfect vision, we must also be aware that it cannot be achieved using the current colour scheme, and so instead we try to push vision in a direction such that the least amount of errors are produced. There are errors that can be fixed, in that it is the result of misclassification, or a flaw in the object recognition method, however there are also errors that in fixing will introduce errors of potentially worse effect in other areas. In such cases, we try to reach a compromise between the two. Early in the year, we had a large problem with the misrecognition of ball and red robot. We had omitted to consider the shadows that would fall on a ball at very close range, so as the robot approaches the ball and prepares for the grab, the ball would disappear because it had been colour segmented to become a red dog, leaving our robot looking lost rather silly. Seeing that, we modified the classification of orange to include shadowed balls, which were really quite red. After the fix the robots were able to catch the balls, but occasionally they would start chasing a red robot, as slabs of red have now become orange. This is one of the ongoing problems that is introduced in many other cases as a result of our inability to cope with colour space sharing, and it demonstrates the dangers in 'fixing' errors as there can be many other unpredictable circumstances in which it fails. We eventually minimized this problem through a combination of classification and modifications to the object recognition code. Even so, there are still cases in

which an error occurs, we average around three errors in a log of two thousand frames.

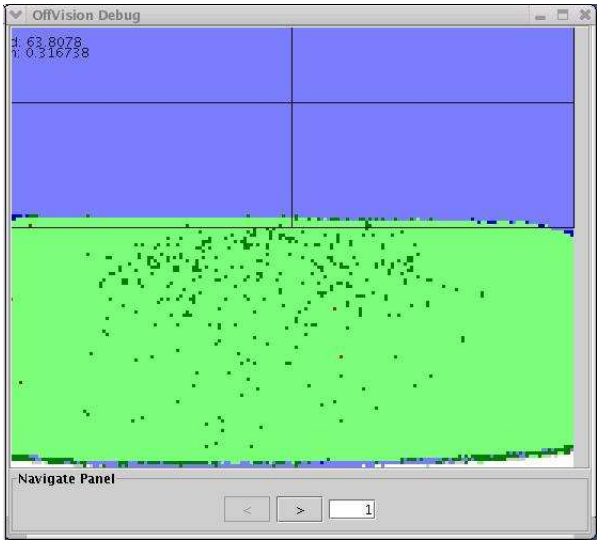


Figure 2.3: Verifying C-Plane Log

2.2.5 Future Improvements

We need to develop a scheme for colour segmentation that is less susceptible to lighting changes. In the early stages of development, we played around with a colour scheme that allows points in the colour space to take on more than one colour. This idea can be further developed by using better method in determining when to assign colours to a particular point and when to ignore it as noise in the sampling. Also, if each point can be of several colours, then how will the robot decide which colour to use. A crude experiment was conducted by specifying colours as maybes and cores. Blobs of maybes and core colours are formed, and the maybe coloured blobs can be merged with their respective bore colour blobs if they are adjacent to one another. More refinement could be made, such as taking into account the proximity of adjacent colour blobs.

Improve the blob-forming algorithm. A thirty percent improvement over previous years in the overall processing time of the vision module was achieved this year. Colour segmentation, run length encoding and blob formation was previously conducted over two runs. The entire process is merged so that it could complete all three components with a single iteration over the image data. In addition, many bugs in the algorithm of the code as well as the codes memory management issues were fixed. It is necessary for the vision module to run as fast as possible so that more sophisticated processing such as edge detection can be performed in real time. One of the simplest foreseeable improvements

is by changing the blob formation algorithm into a variety of the union join algorithm. This idea has been brought up in the later stages of development, close to competition, and we felt it would be unwise to modify a section of vision code without enough time for testing.

2.3 Object Recognition

Following from previous years, the recognition of objects is based on colour, geometric properties and shape. The difference in these properties of objects requires different object recognition methods for each of the types of objects on the field. However, whilst the model remains similar to previous years, there are substantial changes to the object recognition rules and their implementation.

There are primarily three criteria that guides object recognition. The first is physical appearance of the object, which includes colour and the general shape of the object. Shape matching is based entirely on blob bounding box, and only filters out blobs that are outrageously malformed. This leniency is due to the unpredictability of object appearances due to occlusions, motion blurring and colour calibration. We have not been able to improve shape detection without extra resource consumption. With important objects, such as the ball, more sophisticated methods are employed.

The second involve the physical placement of objects. Every object on the competition field has certain zones that they should appear in. For example the ball and robots should not leave the field, and beacons should fall within a certain height range.

The third is the placement of the object relative to other objects. It is not correct for yellow beacons to be seen with the blue goal, even if its shape and height is correct for a beacon. Similarly it is not possible to see both goals at the same time. Such logical deductions can be made on all objects, and filter out the lesser confident of the conflicting objects.

Object recognition is closely tied with colour segmentation. The goal of colour segmentation is to represent image data in a form that the object recognition routines can work with, it is very important for the calibrator to understand what the object recognition routines work best with, and what it considers as a good colour segmented image. Although this year effort has been applied directly at separating the close coupling between the two by writing our object recognition as calibration independent as possible, it is still neccesary for the calibrator to have a good understanding of the object recognition routines to produce competitive colour calibrations.

2.3.1 Image Rotation

Prior to any object recognition is performed, we need to rotate the image back into an upright position. So that basic intuitions such as 'beacons are the highest object in the competition' still hold, and we may use the relative positioning of objects, as well as their known geometric properties to assist us in our task. The effective roll is calculated from the following formula:

Effective Roll = asin (sin (HeadPan) x sin (Head Tilt))

By rotating the image against the Effective Roll about the center of the image, we would have what the image would look like if it was taken with an upright camera. It is very costly to rotate the entire image, but since we are only interested in the coloured blobs, it is neccesary to only rotate the blobs. In this respect, colour calibration could also impact on performance, if the classification produces very noisy images that have many noises blobbed unneccesarily.

For each blob we store both the unrotated and rotated information. However we have found much of the code to misuse these two coordinate system when we began our work on the vision module. Often, the height of objects were compared using the unrotated coordinates of the blobs, thus resulting in spurious object recognition results when the robot has its camera on the side or any other positions that deviates largely from an upright camera position. We had to first go through the vision code and verify every instance of blob processing code, before we were able to perform any meaningful development.

2.3.2 Beacon Recognition

Beacons are the first objects to be processed, and they are in turn used to validate other objects. The chief reason for this is that all beacons consist of a reasonable sized pink section that shows up nicely under the robot's camera in most lighting conditions. In addition pink is well distinguished from all other colours on the field, and it can be reliably classified. For each pink blob, we try to match it with a beacon green, beacon yellow or beacon blue blob, see Fig 2.4

1. The first indication that two blobs could potentially form a beacon is if they are within close proximity of each other. The most straight forward way of checking this is by calculating the distance between the bounding boxes of the two blobs, however bounding boxes are not a good indication of proximity because it does not represent the actual area the blob occu-

pies. Irregular blobs could potentially occupy a large bounding box if it is suitably shaped, but much of the area between the two blobs would be space and the actual blobs could be much further apart than as can be seen through their bounding boxes. Bounding box centroid distance also suffers from the same draw back, although to a lesser extent. What we really want is how close are the actual blobs to each other, in other words we want the amount of space that separates the two blobs. An accurate measure of this would be too expensive to compute between each pair of blob combination, we approximate by drawing an imaginary line from the two centroids and calculate the number of pixels on the line that is space. We disallow all blob pairs with noise to centroid distance ratios greater than a certain threshold.

2. The two coloured sections on a beacon are identical in size and shape, this changes slightly through the eyes of the camera, but the similarity in size and shape should still hold. Thus for two blobs to form a beacon, we expect their screen representations to be equal sized. That is assuming the entirety of the beacon is visible on screen, which is often not the case, as it is common fot the bottom portion of beacons to be occluded by robots and the top portion of beacons to be sliced off by the frame. The second occlusion is especially true when we are tracking the ball, which makes up for most of the our playing time on the field. So instead of direct comparison of sizes, we try to find a ratio that would filter out most noises, whilst catering for still for beacons that have their top partially clipped off by the frame, and beacons with their bottom occluded by roaming robots.

3. Continuing our verification of the beacon shape, we check whether the beacon formed from the combination of two blobs are of appropriate proportions. The previous check ensures that the beacons are similar sized, but does not take into account the shape of the blobsd themselves, so a valid beacon blob could potentially merge with a horizontal line of noise to become a beacon. Thus it is also neccesary for us to check the shape of the resulting beacon. However it is not easy to determine the correct shape because occlusions and frame clipping causes the beacon to take many ratios. This forces us to be lenient so as to not throw out valid combination of blobs, however it also provide opportunity for noise to slip through. A major difference between noise and a valid beacon blob however is that beacon blobs are much more solid than noise of the same colour. In addition, noises form irregular shapes that further decreases their density within their bounding box, whilst a valid beacon blob is solid and have much higher density. Further more, the beacon formed from two beacon blobs would have high concentration of beacon colours due to the compact shape that is formed from two identical sized blobs, so if the beacon produced from the forming of two blobs exhibit low density then it would indicate the shapes of the two blobs do not match, and hence the joining does not result in a valid beacon.

4. Beacons are formed by coloured sections that stand on top of one another. So for two blobs to be considered as candidates for beacon they must be roughly vertical relative to one another. In order to find this vertical relationship, we need to counter the rotation in the robot's camera by

using the unrotated coordinates of the blobs. However the blobs will not be perfectly vertical through the eyes of the robot, due to perspective differences, radial distortion in the lens, edge distortion in the camera, motion blur from camera movement and colour segmentation errors on the edges of blobs. In particular motion blur can cause the most severe distortion to the vertical positioning of the coloured sections of a beacon. At worst, they can be up to 30 degrees from each other, and that forms our threshold.

5. Traditionally, we also check for the amount of field green surrounding the beacon, to eliminate noise on the field. This check has been removed for two reasons. The first is that we have not witnessed any noise on the field that have been recognised as beacons. The greatest source of errors in beacon recognition comes not from the field but from the area above the white boundaries, usually from the colour of audiences or other coloured features in the environment beyond the competition field. And we have devised better checks for the height of a beacon that is applied at a later stage, rendering this obsolete. The second and greater reason for removing this check is that edge detection relies on identifying the boundary between white and field green, and so requires field green to be saturated. However this often causes the shadow casted onto the white boundary from beacons to appear as field green, and thus lures the check into thinking the beacon is instead on the field and so discards the valid beacon. And so this check was removed.

It is possible for more than one blob to satisfy the requirements to join with a given beacon pink blob. In such cases we need to have some way of deciding which blob is more appropriate. We consider a beacon to be ideal if it is vertical, retains the proportions of an average beacon and the two coloured sections are identical shaped and sized. So by the three criteria listed above, we rank the matching blobs by the degree to which they can satisfy the criteria and choose the first in line to be the one that will join with the pink blob.

We calculate the distance to a beacon with the linear formula:

Distance = *Beacon Constant / Centroids Distance*

Where beacon Constant is a number that has been calibrated by measuring the centroid distance of the two blobs that form the beacon when the robot is at various distances from the beacon. The result is then regressed and the constant calculated by applying line of best fit algorithm in Matlab. There are several other properties that could be mapped to distance, including beacon size and beacon height. We found however, that centroids distance is found to be the least susceptible to colour calibration changes. And that is the determining factor due to the large amount of time required for distance calibration and the

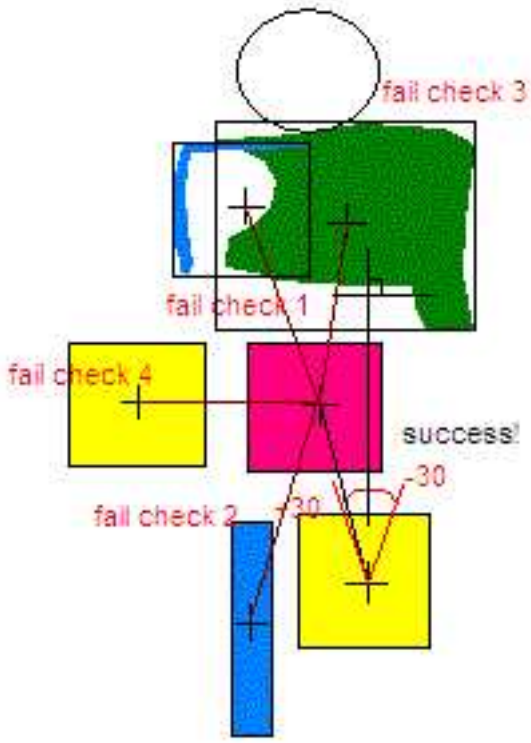


Figure 2.4: Beacon Blob Matching Rules

frequency of colour calibration changes. We have found that Matlab calculates different beacon constants for each of the different coloured beacons, they do not make enough of a difference to warrant a separate constant for each coloured beacon, so we simply take their average. However it does suggest that our colour classification scheme does not treat each colour equally.

However the centroids of the blobs on screen are not neccesarily the centroids of the coloured sections of a beacon, due to either edge clipping or robot occlusion. And extension is neccesary if we are to obtain reliable calculations. We know that the two coloured sections of a beacon is identical in size, so if they are not equal sized then it is occluded or cut off by edge of frame. And we may conclude that the smaller sized of the two is the beacon that has been clipped or occluded, so before we apply distance calculations, we first apply beacon extension by extending the shorter blob to the height of the taller blob. The blobs may not be vertical to each other, as we allowed a 30 degree leniency either way to cater for motion blurred images, so a plain vertical extension in height would not be accurate, rather we extend the actual height of the blobs, in the direction of the line that joins the centroids of the two blobs. By doing this, we are able to better calculate the real centroids of the coloured sections, which into produces distance calculations that is more accurate.

By obtaining the distance to beacons, we are able to determine the height of the perceived beacon by basic geometry, see Figure 2.5. We know that the beacons all sit at constant height around the field, so their calculated heights should all be equal and coincide with the real height of the beacons. And this feature could then be turned into a powerful filter to eliminate noise and false beacons. This is true provided we have perfect distance calculation and camera elevation. Yet neither is the case. Beacon distance is calibrated, and susceptible to inaccuracy in both calibration procedure and the determination of centroids distance. Both camera pan and tilt readings suffer 5 degrees of inaccuracy, and during competition, the level of the base of the robot's neck does not remain still, and this variation is a complex combination of the various leg joints, which cant be measured reliably, and these affect our assumption of a stable neck base in performing the trigonometric calculations. Another error is introduced in the game that changes the position of the legs, the error is a result of interference to leg movements from other robots in a scrum or from the leg scraping against the boundaries of the field. Both further increases the difference in the actual and assumed position of the base of the neck. This height check in principle is very good, as it is simple, clean and geometric. However the errors present means we cannot use the check to its full potential and we need to allow for a much wider range of heights to be potentially acceptable beacons. We have found the range to be very good at removing noise on the field, and the range has been calibrated to remove noise above the the boundary, however it has a danger of removing valid but distant beacons. This mistake is caused by two factors, the first is that distance calculation for very distant beacon is inaccurate, as noise and false colour segmentation near the edges of the blobs make a huge difference to centroid distance. The second is that at large distances, an error in the elevation of the camera results in a large error in the height calculation. Nevertheless, despite its weaknesses, the check is very useful to have around, and has potential for improvement if we are able to somehow calculate more accurately the position of the neck.

In the final step of beacon recognition, we verify the validity of a beacon by comparing it to other beacons.

1. The field layout dictates that it is not possible to see either of the yellow beacons and either of the blue beacons at the same time.

2. It is not possible to have overlapping beacons, unless the front beacon is a green beacon.

If conflicts exist, then the conflicting beacon with lower confidence is discarded. Confidence for beacons is calculated to be a function of its proportions

and density, similar to how we rank the matching blobs to determine one that is best for joining into a beacon.

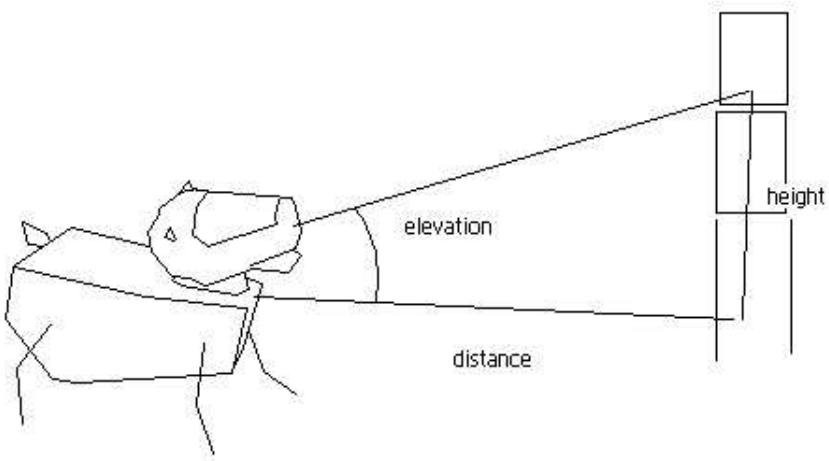


Figure 2.5: height = distance x tan(elevation)

2.3.3 Goal Recognition

Goals are often occluded and separated into multiple blobs, so initially there is no way of knowing whether a goal coloured blob belongs with the goal, or a beacon. Hence goal recognition takes place after beacon recognition, it is more sensible to let the blobs that belong to beacons be picked out, so we can work with the remaining goal coloured blobs knowing that they are either noise or part of the goal. The second justification for this is that we are able to deduce our location from the beacons and hence the location of the goal, but it is significantly more difficult to reliably deduce our location from the goal, and hence it is more valuable to see a beacon than a goal, thus beacon recognition takes precedence.

The first step we take, is to decide for each of the blobs whether they can be potentially a portion of the goal.

1. Goals are always shorter than beacons since beacons are the tallest objects in the field, so if any beacons are detected, we can use it as an indication of height around the image, and remove all blobs that are taller than the beacon.
2. We also remove the blobs that have very low elevation. This check is

rather important because it filters out noise on the surface of the field. The chief noise that could pose as threat to goal recognition is the fringe of the white boundaries, these most often take on the colour of yellow and blue, and have been a source of error in causing the robot to aim at mysterious directions. However, elevation alone is not enough, and we also filter out thin strips of colour and blobs of very low density to filter out diagonal fringes of white lines.

3. Finally we remove blobs that are too small (¡ 30 pixels). Again this is to eliminate noise, mostly to do with blue and yellow coloured fringes of white lines, as it is rather common for fringe of white edges to be classified as yellow or blue as a result of light reflection from the goals, which can cause very undesirable behaviors. If the fringe ends up as part of the goal, extending the goal in a false direction. if the noise as a result of white fringe is small and does not run long, then the previous checks would not have picked it up, for its compact size would give it high density and it would not be skinny enough to be filtered out. Rather it is neccesary for us to eliminate all small blobs with an area under 30 pixels. The obvious disadvantage of this is that we are also throwing out small blobs that could be a part of the goal. However, that is not as big a problem on game play as it seems. When we are close to the goal, these small blobs do not make much of a difference to the shape of the goal, only when we are far away do we lose information due to this check. However when we are far away our concern is in moving the ball up the field rather than aiming for goal and this has not caused any issues. Another effect this has is in failing to join up two sections of a goal if it was split into halves by a robot cleanly. Tha would result in two patches of goal with many patches of small goal coloured blob in between. The small patches is likely to be removed due to their size and so only one of the goal patches would be considered for goal where the other patch will be discarded for being too far away if the small patches are not present to act as links. However this also creates a very interesting side effect, in that by only recognising one of the open sections of the goal, the robot would end up aiming for a gap that is undefended by the goalie when it tries to aim for the center of the goal, and this is a desirable side effect. The draw back is that this misperception of goal placement could adversely affect our localisation.

Having obtained all valid goal blobs, we proceed to join them into a single goal, starting from the largest blob. At each join we verify the correctness of the result, and skip over the blob if the result is unsatisfactory.

1. As a final barrier against noise, we remove goals that are under a certain dimension. It defends well against noise, but again the obvious draw back is that it will ignore distant goals. However a phantom goal formed from noise on or off the field affects game play much more than a distant goal that is not recognised. If the distant goal is not recognised, its location

can still be deduced from the localisation model. However much of the ball aiming decisions we make are based on vision goals, so any noise on the field that have been formed into a goal will trigger a shot, and this is very dangerous for if the noise appeared in the direction of our own half then a huge advantage would have been given to our opponents, potentially a free point. In addition to dimension, we also test density, again to eliminate noise because blobs formed from fringe of white edge are likely to be of low density inside their bounding box.

2. Remove goals to the right of right beacons, and they cannot be to the left of left beacons. As it would imply the goal was off the field.

3. Goals cannot be seen with beacons at the opposite end of the field. For example it is not possible to see the blue goal and any of the yellow beacons at the same time. As an interesting note, in Padua, before the boundaries were erected, we were puzzled over why our walk learner failed to detect any of the goals, and it turned out it was seeing the goal along with beacons in another field that was behind the goal, thus filtering out the goal.

4. We cannot see both goals together, the goal with lesser confidence are removed.

5. Remove goals that are disproportioned. This can be tricky because the camera edges can carve all kinds of sizes out of a valid goal, however in the extreme case that the goal resembles noise, then we discard the goal.

6. If the goal is small then check the number of field green near the goal, there should be an abundance of field green since the goal is situated directly on top of the field. This check is aimed at removing noise that resemble floating goals in the air. It was originally developed because in our laboratory, we can see through a window just above the white boundaries, and during the day it is classified as blue. This check does not apply to large goals. Although we expect to see much field green when we look upon a distant goal, that is not the case with close ranged goals.

If a valid goal can be extracted, then a goal object is created with a bounding box that encapsulates all the blobs it is formed from. Distance to the goal is calculated based on the height of the goal, using a linear equation that is identical in form to the beacon distance equation. The corresponding Goal Constant is obtained in the same fashion as the Beacon Constant, by obtaining the goal height at various distances and to approximate the most appropriate value using line of best fit algorithm. Although calculated in a similar fashion, our goal distance is much less reliable than our beacon distance, and it is an area that requires much improvement. The main reason for this is because the property that we use to determine the distance - goal height - cannot be obtained reliably. The goal is subject to much occlusions and clipping by the camera edges that we cannot be certain how much further the goal extends beyond the top of the frame, or where it starts from if its bottom had been

occluded by a robot. The same problems plague goal width and neither would make a suitable candidate for a property that can give good calibrated distance. Furthermore, the robots cannot obtain good heading information from the goals as humans could. By looking at the boundary at where the goal joins the field, we can deduce our own heading, however the robot treats the goal uniformly regardless of from which angle is it viewed. Some possible angle calculation can be achieved if we use the proportion between goal width and goal height, however as the goal is often occluded we cannot determine the field width of the goal reliably. Thus we can only extract information of limited accuracy from the goals. A more accurate version that assumes no goal occlusion is used in the 2003-localization challenge, in which goal is assumed to be always in full view. [1]

2.3.4 Ball Recognition

Ball recognition gained more importance this year due to the heavy emphasis on Paw Kicks at the strategy level. Because execution of Paw Kicks requires accurate recognition of balls up close when a large portion of the ball would reside outside of the camera frame, a new method was introduced that could determine the center of the ball by applying simple circle geometry and edge detection. Once the center is accurately determined, the ball's relative location can be calculated through 3d transformations.

Far away ball recognition remains similar to previous years, although there were substantial changes to ball sanity checks. Its distance is still determined by the width of its bounding box, with simple extensions.

Far Away Ball Recognition

Ball recognition can be tedious. The ball can be anywhere on the field, it can be of any size, its shape is subject to unpredictable occlusions, and its colour spreads over the widest spectrum due to shadowing and lighting differences across the field. Thus we want a recognition method that is flexible to all the variations mentioned above, for a lost ball would result in delayed reaction, missed opportunities and a potential oppotunity for opponents. On the other hand, we do not want the robot to recognise any phantom balls, for equally that would lead the robot astray and result in all of the disadvantages mentioned above, potentially worse because the phantom ball could lead the robot further away from the actual ball, or into our own team mates, hindering their performance. Ball recognition development was painfully slow because for each change, it needed to be rigourously tested and much of the balance took much time to fine tune correctly. Of all the object recognition routines in vision, ball

[1] Raymond Sheh, *Visual Feature Detection for Robotic Soccer*

recognition was of the highest importance and had the lowest tolerance for error, as any error results in visible hesitation or undesirable behavior in the robot.

We begin by testing the validity of each orange coloured blob, starting fromt he largest. The very first thing we do is extend the bounding box of the blob into a square. We perform this with the assumption that the ball is circular, thus if the bounding box is not a square then it must be partially occluded or partially clipped by the frame. If the ball is clipped, we extend the ball beyond the edge of the frame. Otherwise we extend the ball in the opposite direction of the ball boundary, this is assuming that the visible portion of the ball contains a ball boundary. The boundary is determined by counting the number of ball pixels along the bounding box. The bounding box should be tangent to the boundary of the ball, and so the number of ball pixels on that particular edge of the bounding box should be minimum. Once found, we extend the shorter side of the bounding box in the opposite direction of the edge that is tangent to the boundary of the ball, until it is of equal length to the longer side of the bounding box. This works in all cases except when more than one edge of the ball is occluded or clipped, as we will see later.

1. We check the original dimension of the blob, and we remove blobs that are too thin or too flat, meaning less than or equal to 2 pixels in either dimension. This is intended to remove random noises in the image, especially noise on the fringe of red robots, and in areas where red robot intersects with the yellow goal. It is also intended to remove orange fringe on white edge that has formed from ball reflection or shadows, although this has not been a source of error. This would also remove very far away balls, if they are represented by nothing more than a 2x2 square. This is the primary culprit in preventing our robots from seeing far awar balls. As of now their visual limit in ball recognition is little under two thirds of the field, this distance varies depending on lighting condition, but it does not go beyond three quarters of the field. This has not shown to be a visible disadvantage in competition play, however it would be cause for most concern during penalty shoot out for in such occasions as it is critical to be able to spot a ball from across the entire length of the field. However since this check is the only thing making us short sighted, we can remove this check for the penalty shooter since there will be no robots on the field during a penalty shoot out.

2. Balls stay on the field, so we can remove blobs that are above the horizon. This is would filter out blobs that are too high to be a ball by projecting a ray from the camera to the blob, if the ray does not touch the field, then the blob is ignored. However, this relies on the robot's joint readings and leg stances, and suffers in the same way as the height calculation of beacons.

3. A more reliable height check is to compare the ball height with visible landmarks if there are any. We compare the height of ball to goal and we

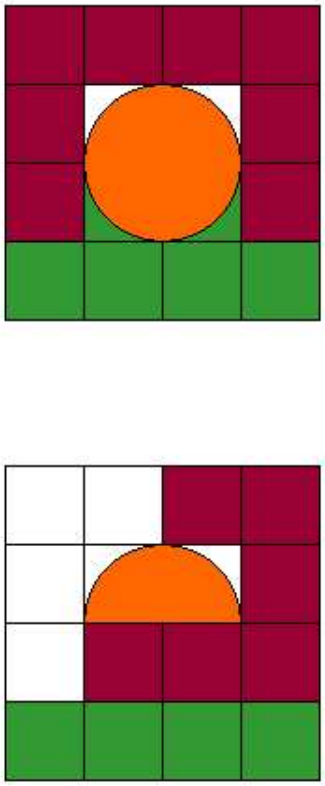


Figure 2.6: Simplified front and side view of red robot in possession of ball

remove any that fly above the goal for we assume balls to stay always on the ground.

4. Similarly we remove everything above beacons.

5. There is a very large tendency for portions of the red robot to turn into balls. This is an inherent problem in that the two colours occupy some substantial overlapping areas in colour space, and there is no way to deal with this problem under our current colour scheme. A shadowed ball takes on the same colour as a red robot, and a motion blurred red robot produces orange, especially when near a yellow goal. There is no real pattern to this noise, the only similarity being that they usually lie on the fringe of red robot coloured blobs. In the calibration process, as much care as possible is taken to separate the two so that the majority of a ball is ball coloured and the majority of red robot is red, this separation is usually done manually. However, it is still necessary to have checks performed specifically against red robot and balls. If an orange blob is adequately big, then we can be sure it is not part of a red robot, assuming we have

reasonable colour calibration, and this check does not apply. And if a blob is small (less than 150 pixels in area) then we need to determine whether it is noise within the robot or a robot holding on to the ball. A simple solution is to take a pixel count of field green, orange and red robot in a square region around the blob. The time when a valid ball is surrounded by most red pixels is when a red robot is holding it. Since it is not possible to define the robot posture whilst holding the ball as both ball and robot shape is unpredictable, we simplify the problem to the general regions the robot and ball occupies, see Figure 2.6. In simplifying this, we can see that the maximum number of red pixels in the square should not exceed the sum of orange and field green pixels. If the number of red pixels is low, then the blob remains, and we filter the blob out in all other cases. With appropriate colour calibration, this simple approach works very well. The key here is in working out the thresholds of conditions that we wish to retain or avoid. This check was introduced in the middle of the year after a considerable effort has gone into preventing phantom balls from appearing on red robots. We were content with the results, however we then found that the blue goalie seemed to concede much more often than the red goalie, and we later discovered that the ball had been filtered out when it was under the possession of a red robot. It is difficult to define accurately when this takes place by relying only on the positionings of orange and red, however since the ball should always stay on the ground, we introduced into the check field green, and that resulted in the stabilized version we now use.

6. The other problem between red dog and ball occurs with the red goalie. As the red goalie moves, its motion blurred body merge with the yellow goal behind it to form orange. If we are blue, it results in our forwards charging the goalie and getting removed from play. If we are red, it results in our defenders approaching the goalie and getting taken to the half way mark. It also affects our game play as it leads robots to falsely believe that they may be in possession of the ball. Either way it causes very undesirable behaviour in our robots. We filter out this noise in a similar manner to the previous check by looking at the number of red, yellow and field green pixels around the ball. And this check does not apply to balls that are of substantial size. But this is also more delicate because at the same time, we do not want to miss balls that are lying just on the goal line. Also we do not want to lose track of balls that are under the goalie's possession. The thresholds for each of the colours were worked out using a similar method to the previous check by simplifying the possible situations to the generalized regions around the ball that each colour can occupy. This method works, but not as reliable as the previous method because this situation is more tedious. Occasionally we would catch one of our robots pan across the goalie then glance back at the goalie or reset their find ball routine, which would indicate that they had seen a phantom ball in the goalie in one of the images. However the problem is much more controlled, and it is not severe enough to bring the robot walking back towards the goalie.

7. Small random noises also exists within the red robot, however this is a much more straight forward and considerably easier problem than han-

dling random orange blobs that appear on the edge of the red robot, as handled by the previous two checks. We filter out all small orange blobs within the bounding box of red robots. This reliably handles noise within the robot.

8. When a robot has its face pressed closely against a yellow goal (camera within 2cm to the goal) or when the yellow goal is under heavy shadowing, the robot has a tendency to see orange in the goal, and would lead the robot walking into the yellow goal. The orange blobs that are seen are usually small and at close distances to the goal, the camera is unable to take in any of the field. So we avoid this by filtering out small orange blobs within yellow goals that is also not adjacent to any field greens. The danger we need to watch for is to avoid filtering out balls that are sitting very close to the goal, or mostly inside the goal. A goal is not scored until the ball has entirely passed the line into the goal. Thus it is possible for most of the ball to be within the goal with the game still running. If we are far away when this happens, then it is not an issues because other landmarks and the green field would indicate to us we are not pressing our head against the yellow goal. If we are close, then the orange blob would be large, and we only execute this filter when the orange blob is unreasonably small to be seen in the same frame with a goal that covers the entire screen. If the camera is pressed entirely against the wall of the goal, then this check also fails because in that case no light is able to enter the camera, and darkness overtakes all. Yet this case rarely happens to our forwards if any at all. It does occur in the case of our goalie, and a separate action was developed to take care of that. Refer to Skills, Out of Goal Spin.

9. Finally, in a final attempt to remove phantom balls, we filter blobs that form ball with low density.

Ball distance is calculated from the linear formula:

Distance = Ball Constant / Ball Side Length

Ball Constant is a number that has been calibrated by measuring the width of the ball on screen at various distances. The length of the side used is that of the longest side. The shorter side is extended to be of equal length with the longer side, in the direction of heavier ball pixel density in the blob. This approach gives accurate measures when at least one of the sides of the blob's bounding box is indeed the longest side, ball extension fails to find the correct size of the ball otherwise. As mentioned previously, the result of the extension is incorrect when neither the length or the width dimension of the orginal blob bounding box represents the true bounding box of the ball. This is true in distant balls if the ball is occluded by two robots in different areas, in such cases it is not so critical because it will still enable us to move in the direction

of the ball, which is the only neccesary function of ball recognition over long distances. When we are at close distances to the ball, the same inaccuracies arise from ball extension because it is very likely for the ball to be clipped by more than one edge of the frame, and in the case that it is only clipped by one edge, then ball extension will still fail unless over half of the ball is present on screen. In other words accurate location of the ball can only be obtained under specific conditions and a substantial amount of time we will be given inaccurate ball calculations, which is unacceptable for any kind of sophisticated ball handling, see Fig 2.7. Therefore a different method needs to be introduced for close balls.

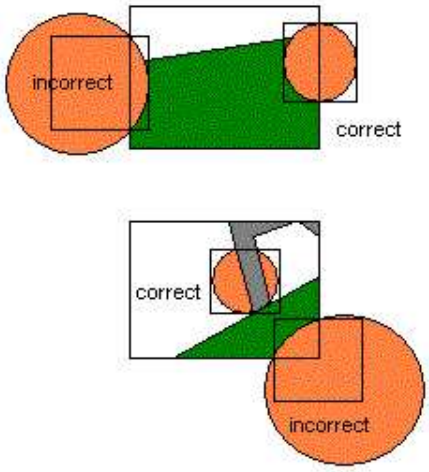


Figure 2.7: Success and failure cases of Ball Extension

Close Ball Recognition(Fireball)

The goal of Fireball is to determine the position of close balls reliably. The old ball recognition method is sufficient if our goal is to grab the ball, for albeit the old method incorrectly extends the ball when it is at the edge of the frame to obtain very inaccurate distance readings, its heading calculation is in the correct direction, and by moving towards the false center the robot is also moving closer towards the real center of the ball, and bringing the ball into center of its view where the old method does not falter. However, ball grabbing is no longer the only interaction we wish to have with the ball. Ball grabbing has several deficits in that it requires the robot to slow down, secure the ball, and setup a kick before the kick can be executed. All of these steps can be potentially interrupted against our favor by opponent robot interferences. Instead we would like our robot to confirm its possession of the ball without grabbing, and to execute actions reliably on the ball, such as to dribble or kick the ball with its paw when the ball comes within kicking range. This would require much more accurate measurement of distances than previously achieved.

It is possible to calculate an accurate ball center using circle geometry, if we assume the ball to be round (Fig 2.8. A perpendicular bisector to a chord on the ball will pass through the center of the ball, and the center can be regarded as the intersection of perpendicular bisectors of two different chords. Two different chords can be constructed from three points on the circumference, so to calculate the center of the ball, all that is required is for us to determine three points in the image that lies on the circumference of the ball.

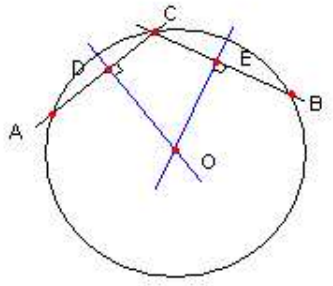


Figure 2.8: Center of circle O, given points A, B and C

The chief difficulty is in detecting the boundary of the ball. We do not want inaccuracy for that is the very thing this method was developed to counter. However we do not have the resources to commit to a full edge detection algorithm, as it would impose an unacceptable performance penalty on our robots. It is necessary for our robots to run at full frame rate for them to behave correctly.

One of the first things we can do is to reduce the edge detection domain from the entire image to an area where the ball is. We are only interested in the boundary of the ball, so we should be able to apply edge detection on the bounding box of the ball only. This is true if there were no noise in the ball, which is not the case, especially when the ball is close and the ball becomes partially red. It is also possible for the ball to bleed into surrounding pixels. This effect takes place most often near the white boundaries, when the ball casts orange shadows onto the white. These noises mean we cannot trust the orange coloured blob to represent the entire ball that can be seen on the image, and the boundary of the orange blob does not necessarily contain the boundary of the ball, so there is no way for us to infer accurately the boundaries of the ball from the boundary of the orange blob, for the amount of noise is unpredictable.

We get around this problem by searching for ball boundary around the border of the frame rather than the border of any blob bounding boxes. We aim to accurately recognise balls that are partially outside of the frame, so the boundaries of the ball must at some point intersect with the boundary of the frame. If it does not, then it must lie either entirely within the image in which case it is not necessary to trigger close ball recognition, or that it lies entirely outside of the image which we can not see.

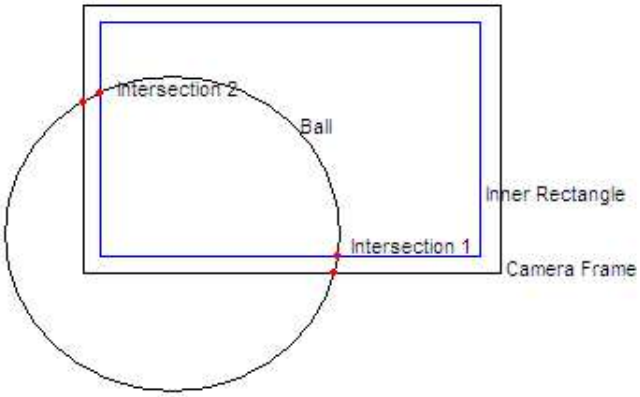


Figure 2.9: Avoid image distortion near edge of frame by using an inner edge detection rectangle

However, distortion to colour is significant around the edge of the image, and much more noise than usual is introduced around the edge of the frame, which makes it non-ideal for our purpose. Thus we shrink the border upon which we perform edge detection by 5 pixels from the frame border, see Fig 2.9. A reduction of five pixels in size on each edge was chosen because it is sufficiently distanced from the border to reduce noise, and it does not give away too much room to let balls slip through. Although it is possible to miss a ball if it intersects the frame border but not our edge detection border. However in such cases, it is unlikely for enough curvature to be apparent for reliable recognition, so we drop back to the old method of simple ball extension, and rely on its heading calculations to bring more of the ball into view.

If the ball does not fit entirely within the frame, then the circumference of the ball must intersect the frame in one or more points. One if the frame forms a tangent to the intersecting point, in which the center of the ball would be the center of the area occupied by the ball in the frame, and it is unnecessary to use fireball, see Figure 2.10. In the second case, it means we can acquire at least two points on the circumference by finding points of the ball that intersect with the edge of the frame, see Figure 2.11.

There were two methods that we have experimented in the edge detection performed around the border. The first edge detection approach attempted was to use 3x3 masks that represented the boundary of a ball at various degree of curvature. We label the cells in the mask to be either one or zero depending on whether they are ball or non-ball respectively. This was first in our experiments because a similar experiment was also conducted at the time in detecting field edges for localisation.

We represent the top left boundaries of the ball with a number of 3x3 masks.

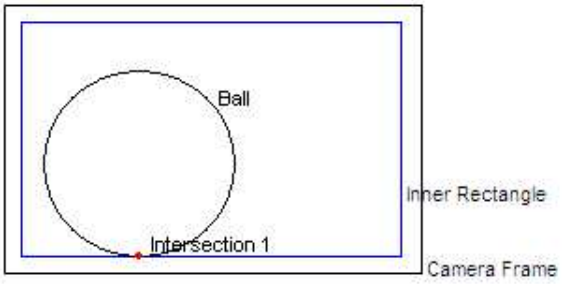


Figure 2.10: Case one

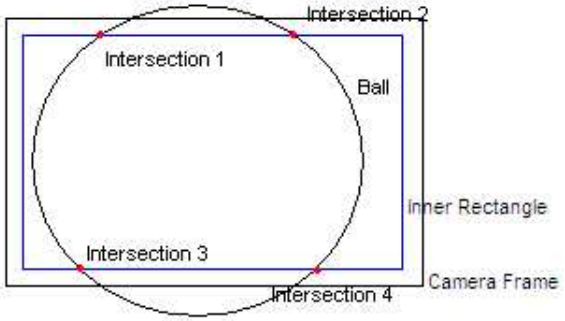


Figure 2.11: Case two

These are decided through analyzing the pixel patterns that form around the ball. We defined a set of sixteen masks for the top left quadrant of a ball's boundary that were then rotated to cover the other three quadrants. We then apply the masks along the the border to detect matches, a valid match occurs when opposite ball quadrants are detected successively, which suggests the entry and exit of a ball. However we have found that it is inadequate to rely on single matches as reliable indications, for in practice, ball boundary take on unpredictable form due to limitations in the camera and colour classification. It is not possible to create a set of 3x3 masks that represent only ball boundaries and no noise. We run the risk of including noise if we enlarge our sets to include rare cases, on the other hand we cannot afford to forfeit any valid ball cases. So we need some way of verifying whether a match is real or noise. This was done by introducing masks that represented the inside of a ball. Once we have discovered an entry point, then we should be inside the ball, and thus masks that match the inside of a ball should succeed until an exit point is matched. This approach fared much better, however it was slow because in effect we are checking nine times the number of pixels around the perimeter of the frame border.

Instead we do away with masks altogether, since they were not able to represent all possible pixel comnbination around the edge of the ball anyway. Rather, we can think of the edge detection border as being either inside the ball or outside the ball, see Figure 2.12. Regions are characterized by consecutive runs of colour, and the boundaries of these regions are the boundaries of the ball. As we traverse the boundary of the border, we keep track of the places where non-ball colours and ball colours switch, if we come upon into a large consecutive run of ball colours then we go back to the last place where non-ball and ball colours switched and label that as the entry point into the ball region. Similarly when we pass into large consecutive run of non-ball colours then we label the last ball and non-ball colour switch position as the exit point from the ball region. This means all switching points that are not followed by consecutive runs of the switched colour are discarded as noise. The definition in the number of pixels in a run to be considered large is dynamic and is proportional to the area of the orange blob bounding box. This method performs better than masks in detecting edges as it would not miss misrepresented ball boundaries, and it only needs to check one nineth of the pixels required by the first method.

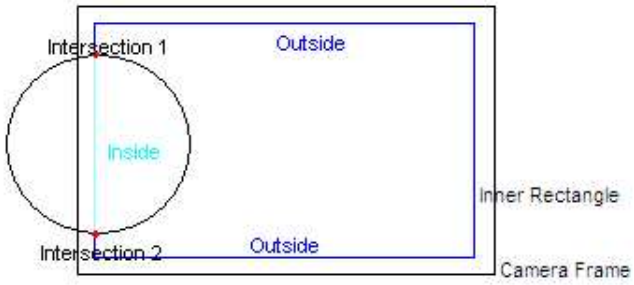


Figure 2.12: Splitting the inner rectangle into Inside of Ball and Outside of ball

At first we felt it was unneccesary to traverse the entire length of the border for we should be able to restrict further the sides of the border that would intersect the ball given the position of the orange blob in the frame. However as mentioned previously, we found we were not able to do so. There is no guarantee that the ball does not extend beyond the orange in other directions in red, see Figure 2.13. It is not possible to rely on the current colour classification scheme to segment colours perfectly, especially when the ball is close, when it is likely to be subject to many shadows.

However, we do try to decide on a suitable starting point and direction. We first test whether the four corners of the edge detection border are within the ball. While we can not say the ball will not intersect an edge of the border based on the position of the orange blob, if the orange blob itself intersects any edges of the frame then we can be sure there are intersections to be found along those edges. Furthermore, we can deduce from where should the rays be fired if any of the corners are occupied. After the corners are identified to be either inside or outside of the ball, we select the appropriate corners as starting point,

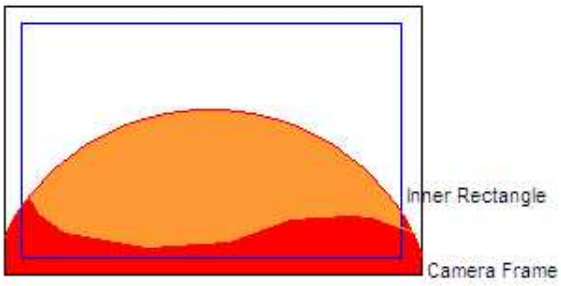


Figure 2.13: Boundary of ball coloured blob not equal to boundary of ball

and the direction from which a ray from the starting point should travel to intersect with the ball. The rule is to choose two corners that are not occupied by the ball, and adjacent to occupied corners. The two may overlap if only one corner is unoccupied. If no corners are occupied, then choose the two corners that subtend the ball coloured blob, see Figure 2.14. From the chosen corners, a ray is fired to search for the boundary of the ball. The ray is fired in the direction of the adjacent occupied corner, see Figure 2.15. In the case that no occupied corners exist, we simply select any corner and fire two rays in opposite directions. If the two rays meet then the ball does not intersect the border.

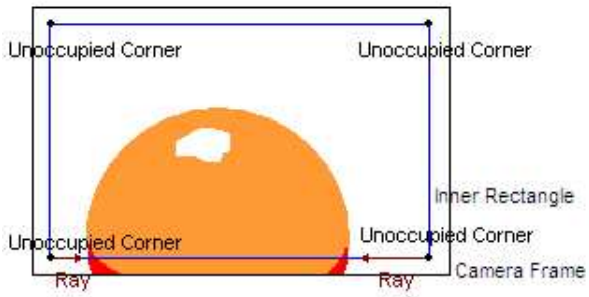


Figure 2.14: the corners that subtend the ball coloured blob is chosen

The ray will continue to advance tracing a path around the border until it comes in contact with the ball. Collision with the ball is defined to be when the ray encounters a consecutive run of ball coloured pixels. We require a run size proportional to the size of the ball to eliminate random noises along the border from falsely detected as the boundary of the ball. This method has been very good at eliminating noise. However around noisy ball boundaries this method is known to throw out the real boundary of the ball if the run of consecutive ball pixels is broken by noise, and pickup the boundary of the ball to be several pixels deeper into the ball where the colour is more solid. This would cause the calculated center of the ball to differ slightly from the real center, however this has not shown to cause any problems.

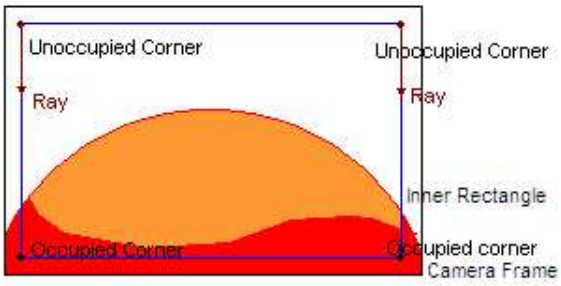


Figure 2.15: the ray is fired in the direction of the occupied corner

The rays around the border will identify two of the three points that we need to build two chords. The third point may or may not intersect with the edge detection border. At any rate, identifying the third point is easier because we already know where the inside of the ball is. We know that the area between our already identified points must be inside the ball, so if we fire a ray from any point in that area in any direction other than the two identified points then we will cross the boundary of the ball eventually at a different place.

Suppose we name the two identified points intersecting the circumference of the ball and the edge detection border Intersection 1 and Intersection 2. Then a line drawn from Intersection 1 to Intersection 2 must be inside the ball, as it would then the chord of the ball, and any point on the chord would also be inside the ball. We choose the midpoint of the chord and fire off a ray in the direction of the perpendicular bisector. The main reason for choosing this point and direction is to have our third point equidistant from the first and second. Because the image is pixilated, quantitation errors are introduced when determining the gradient between two points. The closer together the points, the greater is the error introduced, so to obtain an accurate approximation, we want our three points to be as far apart from each other as possible. Thus we choose the midpoint between Intersection 1 and Intersection 2 to be the starting point of the ray, and we fire the ray perpendicular to the chord, see Figure 2.16. There are two directions for the ray to choose from, and it is determined by the position of the unoccupied corners. We avoid the occupied corners because it is likely the ray will reach the edge of the screen before leaving the ball, instead the ray is fired in the direction nearest to most unoccupied corners. We simply reverse the direction if the initial direction fails to leave the ball.

Edge detection for the third point is identical to the previous two points, we will call this point Intersection 3. We then create two chords by joining Intersection 1 and Intersection 2 to Intersection 3. If only a very small portion of the ball's curvature intersects the edge detection border, the three points will be very close to being collinear. Quantization errors in the camera's pixelated image coupled with small overshooting or undershooting in the edge detection routine could produce three collinear points in rare cases, in which case their

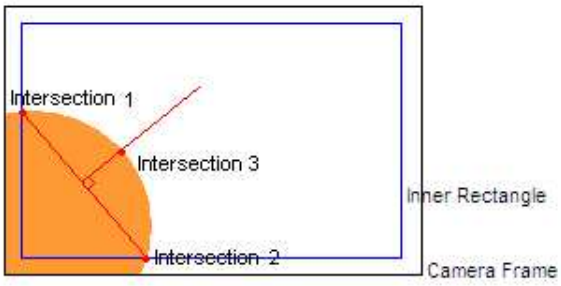


Figure 2.16: Finding the third point on the circumference of the ball

perpendicular bisectors are parrallel and we drop out to ball detection based on blob size. Otherwise we calculate the intersection between the perpendicular bisector of the two chords, which would give to us the real center of the ball in the image plane. This center is very accurate, and several pixels worth of error during edge detection does not impact heavily on the center position.

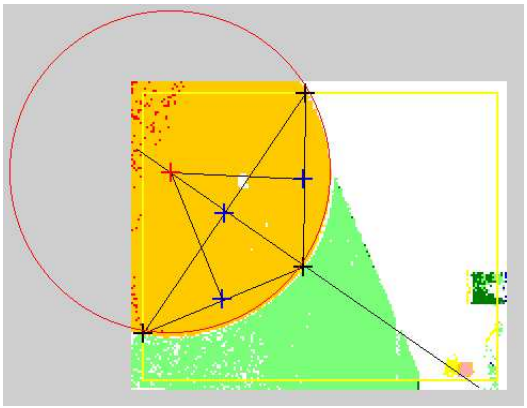


Figure 2.17: Fireball screen shot 1

To calculate the position of the ball, we imagine that the ball on the image plane as the result of rays from the ball striking the camera, see Fig 2.20. So if we follow that ray back through the image center of the ball, we will then end up at the center of the ball at where the ray intersects with the plane upon which the real ball sits on. We will call this the ball plane. This idea is identical to the method used in determining distance to field lines in the localization challenge. The difference is that we raise the plane of intersection from the field to the height of the center of the ball. For details in the three dimensional transformations involved, refer to Raymond Sheh's paper on Robotic Soccer [2]. Note that by using a constant ball plane, we are assuming that the ball will only ever move in a two dimensional domain, that it will never be raised in the

[2]Raymond Sheh, *Visual Feature Detection for Robotic Soccer*

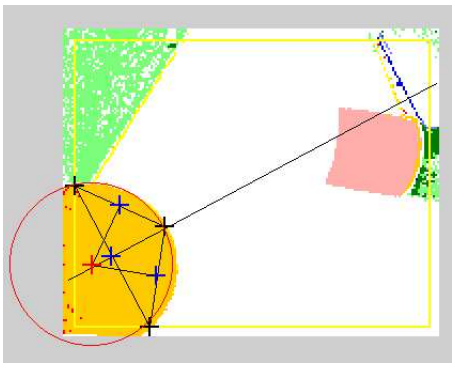


Figure 2.18: Fireball screen shot 2

air. However this assumption is not true around the raised corner ramps, and distance measurements is inaccurate while the ball is raised on the ramp. This had caused a problem in ball tracking, and we had sinced switched to visual rather than geometric ball tracking.

We may obtain ball distance as square_root(square(x_distance) + square(y_distance)) and the heading of the ball to be atan(x_distance / y_distance). This method is more accurate than the older method of using ball radius to determine ball distance, as it uses geometry rather than a calibrated measure based on ball radius, and it is superior in all instances if the robot is stationary. The draw back is that it relies heavily on the position of the camera. Transforming the readings of the three head joints and assuming constant height and angle for the base of the robot's neck calculate the position of the camera. There is a +/- 5 degrees error in the robots head joint readings, although it introduces some degree of inaccuracy, it is not significant over short distances. A more important factor is the assumption that the robot's base of the neck is constant in height and at a constant angle to the ground. Neither is true when the robot is moving, but it is most noticeable if a robot slips severely, or if one of its legs is stepping on the elevated field boundary, causing its body to tilt severely and thus creating large discrepancy between the assumed height and angle of the neck and the actual height and angle of the robot's neck, which affects severely the robot's distance calculation efforts.

Figure 2.21 demonstrates the advantage of fireball's ball center determination and distance calculation. The robot is placed standing on the field, with its head facing directly forward (0 degrees pan). A ball is then placed 10 cm from the robot, at a heading of 45 degrees. The robot then pans its head in 5 degree increments and twenty readings were taken for each angle. Three methods are evaluated in the experiment. The first is the ball recognition method from 2002, which is identical to far away ball recognition. It performs simple bounding box extension by extending the shorter side of the bounding box to equal the length of the longer side, assuming the bounding box of a circle should be square in shape. It then uses the center of the bounding box as center of ball, and from

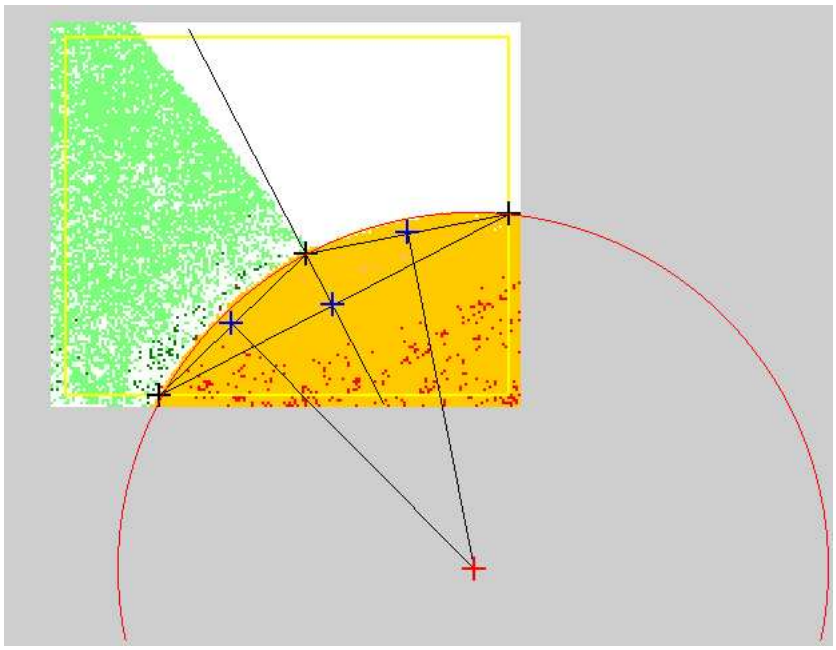


Figure 2.19: Fireball screen shot 3

that calculates ball radius, which then determines ball distance through calibrated values. The second is to use fireball's method of determining ball center geometrically, then calculate ball radius and use the radius to determine ball distance, like that in the first method. The third method is fireball.

When the ball is at large angle offset from the robot, much of the ball will belong outside of the image, and it will not be possible to detect ball center through simple bounding box extensions. This causes the 2002 method to perceive the ball much further than it is. The second method, performs much better by calculating ball radius using the real ball center on the image place and finding its difference to one of the points on the circumference of the ball. However, it still fluctuates because of variations in pinpointing the exact pixel that marks the edge of the ball, its position could differ by several pixels due to noise or lighting changes in the environment. That variation in radius size then affects the distance calculations. The third method, fireball offers result that is most accurate and stable, because the center of the circle calculation is affected minimally by small variations in the position of the points used, and the distance calculation relies solely on the center of circle coordinates, thus offering the stability required for certain demanding ball handling skills such as paw kick.

Table 2.3 and 2.4 shows a broader comparison between Fireball and the older method. We place the ball at various distances and headings from the robot and compare the differences in distances and headings between measured and

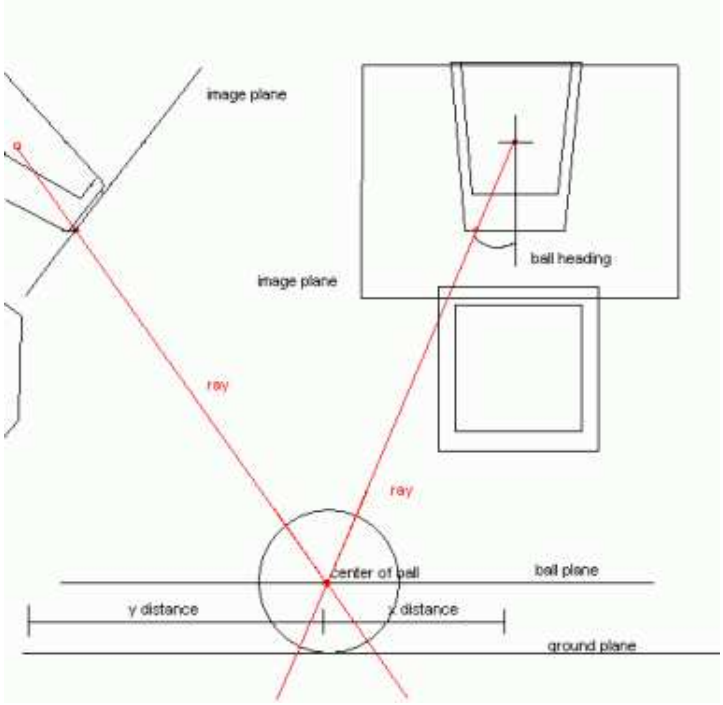


Figure 2.20: Projecting ball onto ball plane

calculated values.

Ball recognition method trigger

The switch from far away ball recognition to close ball recognition is triggered by a combination of the robot's effective head tilt and the size of the orange blob. In addition we check to see if the blob touches any of the screen edges. If a large orange blob is on screen, then if it is a ball, it will be close ranged and thus warrants the use of close ball recognition. However the size of the ball alone is not enough because there is no guarantee how much of the ball remains orange at close range due to noise and various error introducing factors including edge distortion and colour segmentation errors. Thus it is neccesary to rely on other means of detecting whether we will be seeing a far away or close ranged ball. And the head tilt would make a good indication. This is so because the field is flat and the effective tilt of the head directly dictates the distance of objects that can be possibly seen. We say effective tilt because tilt coupled with pan would result in the robot looking side ways possibly into the horizon. We find the tilt such that the furthest ball it can see lies just within the range of reliable close

Actual Distance (cm)	**Actual Heading (degrees)**	**2002 Distance (average)**	**2002 Distance (std)**	**Fireball Distance (average)**	**Fireball Distance (std)**
10	-45	15.89	5.35	10.98	0.81
10	0	11.21	4.53	9.98	0.37
10	60	17.73	5.41	11.68	1.09
20	-45	18.28	3.56	18.21	0.56
20	0	17.99	3.09	18.19	0.37
20	60	19.52	8.14	18.09	0.79

Table 2.3: Result of distance comparison between various ball recognition methods

Actual Distance (cm)	**Actual Heading (degrees)**	**2002 Heading (average)**	**2002 Heading (std)**	**Fireball Heading (average)**	**Fireball Heading (std)**
10	-45	-41.28	10.83	-47.17	3.00
10	0	-4.47	7.83	-1.03	2.31
10	60	50.32	7.62	61.02	3.67
20	-45	-44.25	2.64	-45.89	1.24
20	0	1.21	3.25	0.06	0.42
20	60	52.65	2.23	55.36	1.41

Table 2.4: Result of heading comparison between various ball recognition methods

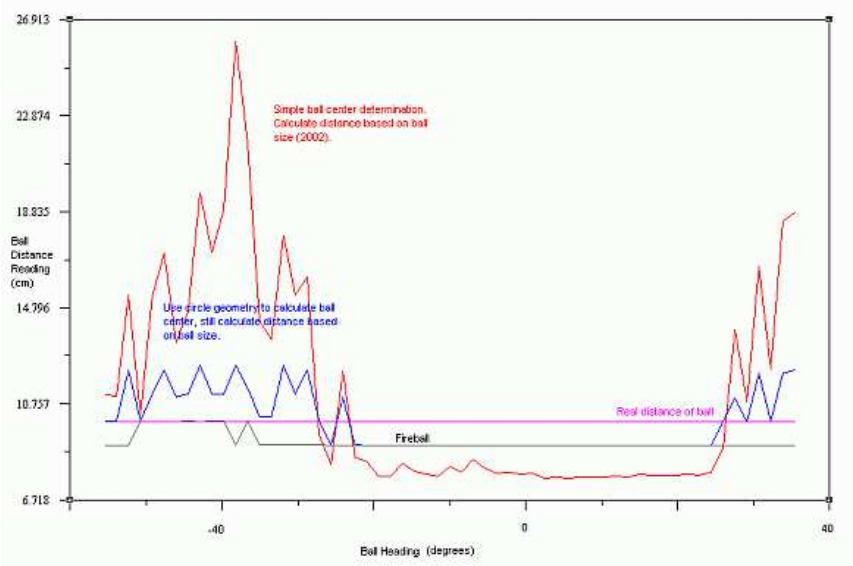


Figure 2.21: Ball distance calculated by the various methods

range ball detection, and at that point we switch the ball detection methods. That would solve all our problems if the robot was stationary and joint readings were error free, alas it is not so there are times when we still need to rely on the blob size of the ball to trigger the switch.

There is also a special case in which the very close ball detection method kicks in. When we are so close to the ball that there are no isible curvatures on the ball. In otherwords, the ball occupies the entirety of the screen and we have no information to work with. This case happens when the robot presses its camera on the ball. Usually in such events the ball would fail to appear orange as there is insufficient light and instead the screen would be almost entirely red. In such cases, we simply assume the center of the screen to be the center of the ball, and we project a ray through the center of the screen to find an intersection with the ball plane, and approximate that to be the ball's position. This method is triggered by a combination of head tilt and area of red blob. The head tilt is defined to be the minimum at which it is possible for the robot to press its head against the ball, so as to avoid falsely recognising the ball in similar situations that arise from presseing its head against a red robot.

As with all object recognition, close ball recognition has its own set of sanity checks. We know there is an error if the calculated bounding box of the ball does not contain the entire ball blob. We allow minor leniencies on this check because as seen from the screen shots, the detected ball tends to stray a pixel or two from the ball blob due to edge detection variations. But the calculated bounding box of the ball should contain the entire ball that is visible, in fact the circumference of the calculated ball should overlap with the circumference

of the visible ball if it was entirely accurate. By not bounding the entire visible ball, it shows there are errors in the calculated ball.

We also test whether the calculated ball can reside entirely within the camera frame. If it is able to then there is either an error, or it was unneccesary of us to use close ball recognition method, as ball extension would have done equally well if not better by avoiding the errors introduced by edge detection. In either cases we drop back to far away ball recognition methods.

Finally we calculate the elevation of to the center of the calculated ball and elevation to the center of the orange blob on screen. The elevation to the center of the ball should always be equal or less than the elevation to the center of the orange blob, except in the case when the ball blob is cut off by the top edge of the camera. If any of the fore mentioned checks indicate an error with close ball recognition, then we discard the result and resort to the far away ball recognition method.

2.3.5 Robot Recognition

The most unpredictable shapes in an image are robots, which makes it difficult to decide whether a robot coloured blob is noise or indeed a section of a robot. This section deals with the filtering of robot blobs and the filtering of formed robots. For robot formation from the filtered blobs, multiple robot separation and robot distance calculations refer to Eileen Mak's paper [3]. We begin blob filtering by rotating every robot coloured blob to an upright orientation, similar to what we did for beacon and goal recognition.

1. If the robot's camera tilt is less than -45 degrees, then we cease all robot recognition. For it is not possible to see any robots at that angle, except for the occasional robot legs in which case it is not possible to identify the colour of their owners reliably. We have also found to this to have the benefit of preventing the shadowed sections of a very close ball from turning into red robots. Also, there is a tendency for shadows projected by various objects including the camera itself to be classified as blue robot. This happens because robot blue is very close to black through the robot's camera. And this check reliably filters out blue robot blobs that result from shadows.

2. Similar in principal to the ball checks, we remove robot blobs that are higher than any visible landmarks. If there are no landmarks in sight, then check the elevation of the blobs itself. It is more accurate to check against landmarks, and only in their absence do we check blob elevation with a tight constraint that sometimes falsely filters out valid robots. This

[3] Eileen Mak, *Undergraduate Thesis*

is necessary because a far away robot coloured blob such as a dangling blue jumper from spectators can have equal elevation to the head of a very close robot. So in filtering out the dangling arm of a spectator, we also filter out a valid robot head in the absence of landmarks.

3. In the case of the ball, we have many checks guarding against orange noises in red robots from turning into balls. Similarly, we must guard against red noises in ball from turning into red robots, because it will cause a red robot to back off from the ball because as it will see the ball as being in the possession of its team mate. At the same time, we need to be wary so as to not remove red robots that appear to be around the ball because it is holding onto the ball, because then it would prevent another red robot from backing off when it should. This is essentially identical to the problem we have with ball but with red robot as subject. However this problem is considerably easier to solve because red robots are always much larger than the ball, so the difference between noise and valid red robot around the ball is easier to distinguish.

4. Similarly, remove all robot blue blobs inside and around the bounding box of the ball, to prevent ball shadows from forming blue robots.

5. The dark areas on a robots legs and in various other sections of its body are very often classified as robot blue. It is an area that we spend alot of effort to balance during the calibration phase because the robot's camera has difficulty int separating robot blue and black well. We do not want these black sections to be recgonised as robot blue because it would render our robot recognition system useless if we begin seeing blue robots lingering on the legs of the red robots. So we filter out all blue robots that lie inside the bounding box of the red robot, much like how we filter out the red robots that lie within the bounding box of the ball. We also filter out blue robots in the region immediately adjacent to the bounding box of the red robot. This is because the legs of a robots are often outstretched, beyond the bounding box that is formed by their red patches. We realise that this extension of filtering space can possibly filter out valid blue robots, so care is taken to restrict the size of the region proportional to the size of the red robot, in addition the filtered blue robot blobs are only considered if their proportion to the red bounding box falls under the maximum threshold that can be obtained between the ratio of a robot's legs and its body.

6. Filter out red robots that lie entirely within the bounding box of a blue robot and vice versa. This is performed after performing both red and blue robot formation. We are now working with robots as basic entity rather than blobs. If robots were inside the bounding box of another, then it would imply the robot is farther away. However in that case then the larger robot should block it visually and we should not be able to see it. So by being able to identify a robot within the bounding box of another means it is likely that we are dealing with noise. There are cases where it is possible to see valid robot bounding boxes within one another, such as when we look over the back of a robot, however this happens rarer than misidentifying noise, and so the check remains.

7. Robots can never be under a ball. If a ball is found to have higher elevation than a robot then the robot is either a shadow under the ball or some section of the ball misclassified as robot. There is one case where a robot can appear to be under a ball, if we are overlooking the back of a robot at a distant ball. However, in this case, by not recognising the robot, it impacts minimally on game play. Although this is subject to change if we start incorporating obstacle avoidance into our core strategy, and it becomes critical not to miss any obstacles on the field.

8. Robots remain on the field and so should not appear above any goals or beacons.

9. In addition, we compare the distance of robots to visible beacons to ensure that the robot does not lie beyond the field, in which case it would mean the robot is a result of noise. We give robot distances a generous 20 cm leniency in this check due to difficulties in obtaining accurate robot distance calculations. This has some success in removing the blue robots that form as a result of the referee's black pants during competition.

10. If no landmarks are sighted, then we apply some tightened checks on robot height and elevation to prevent noises over the boundaries from being recognized as robots.

Of utmost importance in our robot sanity checks is in preventing noise from forming valid robots. We had first witnessed the effects of this problem in the Australian Opens, in which our robots began to back off from audiences around the field and from the referee's pants. Of almost equal importance is the reliable detection of valid robots. This is important because much of our strategy relies on close range interaction between team mates and this cannot be execute correctly without good robot detection. The two goals are often conflicting in that by improving one, we can be unknowingly weakening the other, and like ball detection robot detection needs very fine balance. However, not as much time as we would have liked to was spent in this area, and there is much that can be done to make improvements in this area.

2.3.6 Sanity Checks Development

Vital to object recognition are deductive reasoning that take the form of sanity checks. However, they are often difficult to develop and test due to the limited access to the robots during run time. The live C-Plane feed is no longer sufficient for this task, because it is difficult for the human eye to pick out errors such as a falsely bounded or falsely recognized object that appears for only a frame. In addition, even if the error was picked out, it is often difficult if not impossible to reproduce the same environment again due to factors such as the camera's automatic white balancing and laboratory lighting. As a result, even after a fix has been applied, we cannot run the robot over a situation identical to

the original one that triggered the problem and we can only approximate the situation.

To solve this, we saved each of the frames sent through the C-Plane live feed to disk, and the robot's vision module is ported to work offline. There is no difference between the offline and online vision code, except that a dummy robot body called Offline Vision manages its execution. Localization code has also been integrated into Offline Vision, however although it runs successfully based on the processed visual information, it is inconsistent to the localization data on the robot because it is lacking previous localization details and odometry updates. In addition to offering a platform for developing vision, Offline Vision can also be used in conjunction with GDB and Valgrind to obtain performance profiles and to check for memory leaks that are otherwise difficult to find.

Offline Vision reads in C-Planes, and converts them back in to raw Y U V format. This is unnecessary if we are only interested in the object recognition code, since that works with colour-segmented data. However we would also like to check for errors in colour segmentation and blob formation, so we reverse the C-Planes back to raw colours using the robot's colour classification table. This information is then processed as a raw image is processed on the robot.

During processing, the executed code is free to display as much debugging data as necessary. The result including the debugging data is then written out to a file. A java program then reads the results file with the input C-Plane, and displays the C-Plane with the bounding boxes and object data as recognized by Offline Vision. This allows us to quickly determine errors in object recognition, and we are free to experiment various object recognition methods on a set of data to compare their results. The display program should have been integrated into Offline Vision, this has not been done because I was much more familiar with swing, and there was not enough time to learn anything else.

The Offline Vision tool plays a vital role in ensuring the validity of our vision system. By running our code through Valgrind is also the main means through which we pick up memory management issues. Although offline development has drastically improved development time and quality, it must be noted that ultimately the robot needs to be tested in a real game. By having the system work perfectly with the captured set of logs, we run the risk of over fitting our vision module to particular set of conditions, thus exposing us in an unpredictable environment such as during competition play. In addition, the robot is unable to send every frame to the base station, we only receive one out of two frames on average, so it is possible there are errors that slipped away, and logs should be taken over an extended period of time to avoid that risk.

We have seen a drastic increase in speed in the development of sections that can make use of Offline Vision. Not only sanity checks development, but ball challenge and obstacle avoidance challenge both included Offline Vision as a vital tool in their development cycle. Perhaps in porting other sections of the

code offline, we could witness similar improvements to development speed.

2.3.7 Future Improvements

The positions of the robot's legs needs to be taken into account in determining the real position of the robot's camera so that all the distance measurements relying on 3D transformations can be freed from the tyranny of the bobbing camera.

A more systematic approach has been taken in developing the vision system this year, in that we have more means of evaluating effect of changes and better methods in catching errors. However, to approach vision in an even more principled way, perhaps it is possible to teach the robot to generate a set of rules from the positions and sizes of blobs. An experiment conducted by Alex Tang on this topic can be found in his thesis.

One of the landmarks that is not used to its full potential, are lines. Raymond Sheh has worked on line detection for the 2003 localization challenge, detail can be found in his thesis [4]. However the line detection algorithm impacted heavily on our frame rate and it was too slow to be used in competition play. We need fast and reliable edge detection, so that the robots are able to maintain localization even when their cameras are focused downwards on the ball. This is especially important for our goalie and our forwards, as they are able to benefit enormously from localizing off the goal box.

Robot recognition will need to be improved for development of advanced strategies. In addition we need to recognise the orientations of robots as well, as this information could be of great value in determining the next action for our robot.

Offline Vision can be extended to run not only vision, but also the entire robot code online. The current code already runs the behaviors, localization and vision modules. However the behavior and localization representations are not consistent with the robot's internal representation given the same visual data, due to lack of odometry and incorrect frame rate. By porting the entire code offline, we could play out entire matches offline and analyze in detail the pros and cons of each action. Whilst it cannot replace real games, it would be enormously beneficial to test whether an idea is worth investing time in.

[4] Raymond Sheh, *Visual Feature Detection for Robotic Soccer*

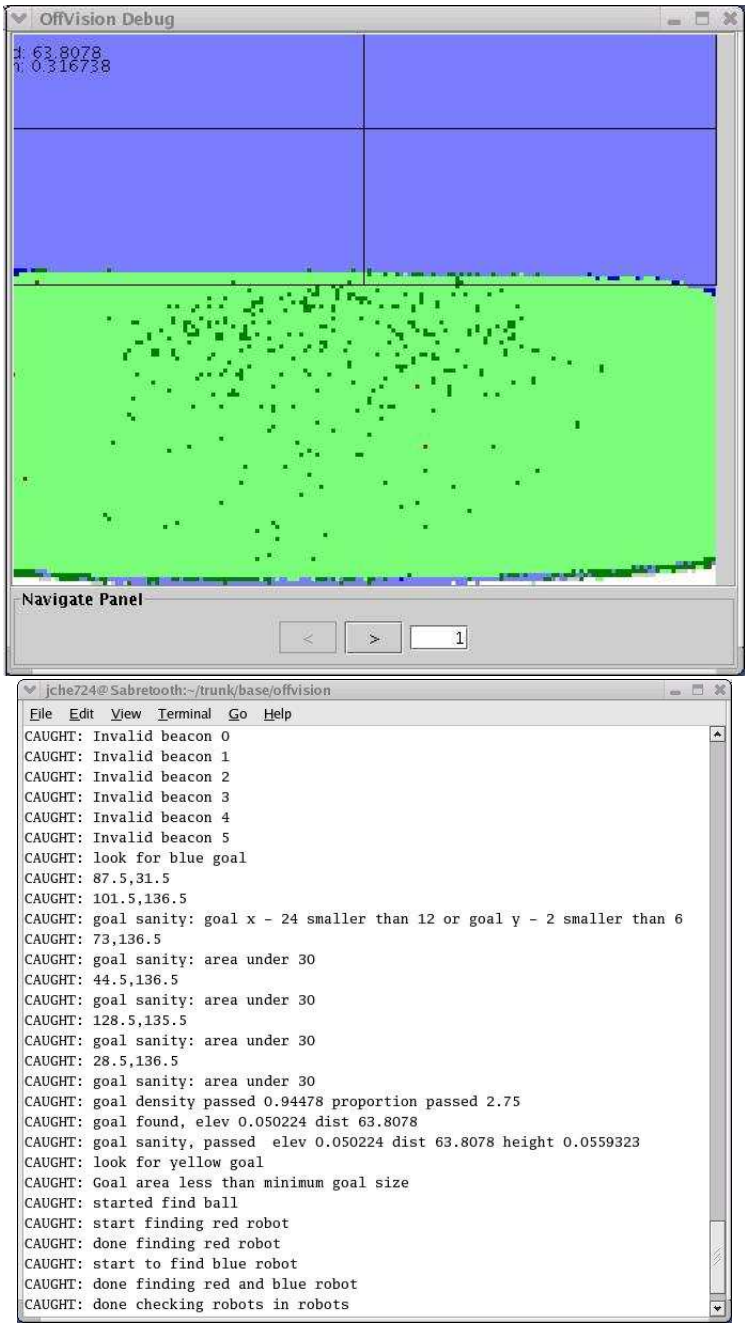


Figure 2.22: Offline Vision display and corresponding debugging output

Chapter 3

Localisation

3.1 Introduction

In a perfect world, we would liken the behaviour of playing soccer to the act of consideration: given the current position of all definitive objects, what action would deliver the most desirable next set of positions. It is clear to anyone who understands the game, that if an entity has the ball, with a clear shot at a close attacking goal, then the obvious action is to shoot. However, as in most real-life situations this view is dramatically simplified. When considering how to program a robot to play soccer, a key factor in developing intelligent behaviour is the robot's ability to determine the current position of the definitive objects in the game. In fact, this seemingly simple task has been, and continues to be a limiting factor in the ability of individual robotic entities, and teams, to act intelligently in the game of soccer. It is important to understand that an agent's level of intelligence will be limited by the information it is basing it's decisions.upon.

The term Localisation is used to represent the system that robots use when determining the position of the definitive objects in the field. The term "definitive" is used to portray the fact that their may be different quantities deemed important to a robot playing soccer and this is highlighted by the fact that a different set of quantities may be estimated by the localisation system in different teams. The localisation system uses the information gained by the vision system, the locomotion unit, and the information received from teammate robots through wireless communication, to keep a running estimation of the values of these definitive quanitities that the robot may call upon during it's decision making process. It may seem unnecessary to use a system in this way, since one would assume that a robot may act on what it can see, and processing this information can not possibly be used to gain information. However, there are

several reasons for the localisation system :-

- Information received from systems is not necessarily accurate, and may be inherently inaccurate. For instance, the vision system uses a CMOS camera with a resolution of only 176 x 144 pixels, hence given a perfect vision system, distant objects may only be composed of a few pixels, and since this is largely used to determine distance, noise in the camera guarantees that the distance estimate will be inaccurate. The localisation system may be used to smooth out these errors, using recently received information to avoid believing outlying errors.

- Information gathered from several systems is hard to consider simutaneously, in such cases information from a single source is usually considered, while the others are ignored. The localisation system can be used to take information from several sensors and merge them into a single model of the world, a process referred to as data fusion.

- The vision system, as well as locomotion and wireless communication, take no consideration of temporal issues, dealing only in the readings their respective sensors are currently obtaining. The localisation system is used to take readings from all sources, including recent readings, and determine an estimation of the current state. This is similar to the multiple sensor point, since the readings over time could be considered as separate systems, and is also considered part of data fusion.

Having considered the reasons we have a localisation system we may better define exactly what it is. The localisation system takes a (possibly infinite) series of readings from various sensors, in a temporal sense, and maintains a best estimation of the state of the world. In the case of rUNSWift 2003, the state maintained by the Localisation system is considered as composing of :-

- Self Localisation - this is maintained as an estimation of a vector in x, y, θ space, where x and y are coordinates across and down the field respectively, and θ is an angle representing the direction the robot is facing relative to the x-axis. This is equivalent to, and considering the rest of the world model better thought of as, maintaining the position of the static world (beacons, goals) relative to the robot.

- Ball Localisation - this is maintained as an estimation of the vector $x, y, \dot{x}, \dot{y}$ where x, y is the position of the ball in global coordinates similar to above, and $\dot{x}, \dot{y}$ is the change in position per frame. This is equivalent to maintaining an estimate of the position and velocity of the ball relative to the robot's position [1].

[1] Probability factors such as covariance are conceptually the same despite being numerically different

- Teammate Localisation - this is maintained as a set of three estimated positions of friendly robots, identical to that of our own position. Again this is equivalent to maintaining their positions relative to the robots position.

- Opponent Localisation - this is maintained as a set of four estimated positions of opponent robots, x, y vectors, similar to x, y of our own position. The equivalency also carries here as well.

rUNSWift 2002 team developed a fully functioning localisation system which was analysed prior to the development of the current system. Hence the new system is strongly based on the general methods of the old one. While the details difference between the individual components will be discussed in the relevant sections, there are several general areas that were greatly improved. The new system has greatly increased the complexity of the general system used, this has produced a more mathematically correct model, and has particularly improved the correctness of the model's interpretation of observations from the sensors. The current system also performs a lot more data fusion, where the old system requires the definition of "Vision model", "World model" and "Wireless model", the new system concentrates on providing a single estimation of the world, taking into account the inaccuracies of respective information sources.

3.2 Self Localisation

3.2.1 Introduction

Self localisation is the estimation of the robots current position on the field. As previously noted, this is equivalent to estimating the position of all static objects on the field, given their physical arrangements. The purpose of the self localisation system is to provide the behaviour system with the best estimation of the robot's current position, so that it may make the best decision regarding it's next action. Self localisation is extremely important in all facets of the game, since all actions depend on current position, regardless of whether the robot does or does not control the ball.

In 2002, and 2003 localisation, the self localisation system, and the other localisation systems, represented the probability distribution of a state as Gaussian.

3.2.2 Gaussian Distributions

A Gaussian probability distribution, also known as a normal probability distribution, with one dimensional variate x can be defined as

$$\mu = \text{Exp}(x) \tag{3.1}$$
$$= \text{expected value of} x \tag{3.2}$$
$$\sigma^2 = \text{Var}(x) \tag{3.3}$$
$$=< x, x > \tag{3.4}$$
$$= \text{Exp}((x-\mu)^2) \tag{3.5}$$
$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.6}$$

Hence it is defined by it's mean μ and variance σ^2, and denoted $x \sim N(\mu, \sigma^2)$. Plotting such a distribution leads to a bell curve centered at the mean μ, flattened according to it's variance σ^2, such that it's total area above the x-axis limits to 1.

$$\int_{\infty}^{\infty} P(x)dx = 1 \tag{3.7}$$

We may take confidence intervals as intervals about the mean μ, [μ-$\alpha\sigma$,μ+$\alpha\sigma$]. For instance setting $\alpha = 1.96$ defines the 95% confidence interval (we are 95% confident the true value lies in this interval). We may also consider the interval of a single variance, which also uniquely identifies a Gaussian distribution, though will not have a constant confidence, since a variance is a variable number of standard deviations (one standard deviation of them to be precise).

Given the definition of a 1-dimensional probability solution, we may naturally form a definition of an n-dimensional Gaussian distribution: -

$$\underline{z} = z_1 \times z_2 \times \ldots \times z_n \tag{3.8}$$

$$z_i \sim N(\nu_i, \sigma_i) \tag{3.9}$$

$$P_i(\zeta) = \frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{(\zeta-\nu_i)^2}{2\sigma_i^2}} \tag{3.10}$$

$$P(\underline{z}) = P\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix} \tag{3.11}$$

$$= P_1(z_1)P_2(z_2)\ldots P_n(z_n) \tag{3.12}$$

This distribution is well defined since: -

$$\int_\infty^\infty \ldots \int_\infty^\infty \Pi_{i=1}^n P_i(z_i) dz_1 \ldots dz_n = \Pi_{i=1}^n \int_\infty^\infty P_i dz_i = 1 \tag{3.13}$$

We may again consider intervals of a set number of standard deviations, except that we must consider an n-space "volume", whose boundary will pass through $\nu \pm \alpha\sigma_i$ on each axis around the mean. When a point is not on one of the axes, it is difficult to define the number of standard deviations it is away from the mean, but we know it must have the same probability as a point that lies on one of the axes the same number of standard deviations away from the mean. This is easiest to see if we rewrite our probability equation: -

$$P(\underline{z}) = \frac{1}{\sigma_1 \ldots \sigma_n \sqrt{2\pi}^n} (e^{-\frac{(z_1-\nu_1)^2}{2\sigma_1^2}} \ldots e^{-\frac{(z_n-\nu_n)^2}{2\sigma_n^2}}) \tag{3.14}$$

$$= \frac{1}{\sigma_1 \ldots \sigma_n \sqrt{2\pi}^n} e^{-(\sum_{i=1}^n \frac{(z_i-\nu_i)^2}{2\sigma_i^2})} \tag{3.15}$$

Now, if we assume $z_i = \sigma_i \pm \alpha_i\sigma_i$ we get: -

$$P(\underline{z}) = \frac{1}{\sigma_1 \ldots \sigma_n \sqrt{2\pi}^n} e^{-(\sum_{i=1}^n \frac{\alpha_i^2}{2})} \tag{3.16}$$

Hence the probability is a function of the sum of squares of the differences between each coordinate and the mean of that coordinate. Therefore, a point lies on the boundary of the k-standard deviation volume if and only if it is of the form

$$\begin{pmatrix} \nu_1 \pm \alpha_1\sigma_1 \\ \nu_2 \pm \alpha_2\sigma_2 \\ \vdots \\ \nu_n \pm \alpha_n\sigma_n \end{pmatrix} \sum_{i=1}^{n} \alpha_i^2 \;=\; k^2 \tag{3.17}$$

It is quite clear that if all standard deviations σ_i are 1, this set will form a hypersphere, centered at the mean, radius k. The action of each standard deviation in the above set is to scale each of the coordinates with respect to the mean, hence, in general, the k-standard deviation volume will be any translation and scaling of the hypersphere S^{n-1}, an ellipse. For a particular distribution it will be the ellipse centered at the mean with the standard basis as the axes and $k\sigma_i$ the length of each axis respectively. The same can be said for variance volume, although the ellipses will not be similar, they will have the same axes with a one variance volume having axes lengths the square of the one standard deviation ellipse.

So far we can define an n-dimensional Gaussian distribution in $\Re^n$ which can have arbitrary mean (translated away froim origin) and arbitrary positive standard deviation / variance (scaling) but must be uncorrelated in each standard basis' axis. We would like to be able to define it for any orthogonal set of axes. We may do so by taking an arbitrary rotation transform R, and associated matrix $\underline{\underline{R}}$, which rotates the standard bases x_i to the bases of the distribution z_i, i.e. $\underline{\underline{R}}\underline{x} = \underline{z}$. We also must restate our probability equation in a more linear transformation friendly way, let $\underline{\underline{D}}$ be the diagonal matrix with variances equal to the distributions' variances: -

$$\underline{\underline{D}} = \begin{pmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_n^2 \end{pmatrix} \tag{3.18}$$

$$\underline{\underline{D}}^{-1} = \begin{pmatrix} \sigma_1^{-2} & & 0 \\ & \ddots & \\ 0 & & \sigma_n^{-2} \end{pmatrix} \tag{3.19}$$

and let $\underline{\nu}$ be our mean vector in distribution coordinates: -

$$\underline{\nu} = \begin{pmatrix} \nu_1 \\ \vdots \\ \nu_n \end{pmatrix} \tag{3.20}$$

It is easy to see that $|\underline{\underline{D}}| = \det(\underline{\underline{D}}) = \sigma_1^2 \dots \sigma_n^2$ and: -

$$(\underline{z} - \underline{\nu})^T \underline{\underline{D}}^{-1} (\underline{z} - \underline{\nu}) = \sum_{i=1}^{n} \frac{(z_i - \nu_i)^2}{\sigma_i^2} \tag{3.21}$$

Hence the probability equation becomes: -

$$P(\underline{z}) = \frac{1}{\sqrt{|\underline{\underline{D}}|\sqrt{2\pi}^n}} e^{-\frac{(\underline{z}-\underline{\nu})^T \underline{\underline{D}}^{-1} (\underline{z}-\underline{\nu})}{2}} \tag{3.22}$$

$$P(\underline{x}) = \frac{1}{\sqrt{|\underline{\underline{D}}|\sqrt{2\pi}^n}} e^{-\frac{(\underline{\underline{R}}\underline{x}-\underline{\nu})^T \underline{\underline{D}}^{-1} (\underline{\underline{R}}\underline{x}-\underline{\nu})}{2}} \tag{3.23}$$

$$= \frac{1}{\sqrt{|\underline{\underline{D}}|\sqrt{2\pi}^n}} e^{-\frac{(\underline{\underline{R}}(\underline{x}-\underline{\underline{R}}^{-1}\underline{\nu}))^T \underline{\underline{D}}^{-1} (\underline{\underline{R}}(\underline{x}-\underline{\underline{R}}^{-1}\underline{\nu}))}{2}} \tag{3.24}$$

$$= \frac{1}{\sqrt{|\underline{\underline{D}}|\sqrt{2\pi}^n}} e^{-\frac{(\underline{x}-\underline{\underline{R}}^{-1}\underline{\nu})^T \underline{\underline{R}}^T \underline{\underline{D}}^{-1} \underline{\underline{R}} (\underline{x}-\underline{\underline{R}}^{-1}\underline{\nu})}{2}} \tag{3.25}$$

Since $\underline{\underline{R}}$ is a rotation matrix, it is orthogonal. Hence $|\underline{\underline{R}}| = |\underline{\underline{R}}^T| = 1$ and $\underline{\underline{R}}^T = \underline{\underline{R}}^{-1}$, and further $\underline{\underline{R}} = \underline{\underline{R^{T^{-1}}}}$. Hence our equation simplifies to: -

$$P(\underline{x}) = \frac{1}{\sqrt{|\underline{\underline{R}}^T||\underline{\underline{D}}||\underline{\underline{R}}|\sqrt{2\pi}^n}} e^{-\frac{(\underline{x}-\underline{\underline{R}}^{-1}\underline{\nu})^T \underline{\underline{R}}^{-1} \underline{\underline{D}}^{-1} \underline{\underline{R^{T^{-1}}}} (\underline{x}-\underline{\underline{R}}^{-1}\underline{\nu})}{2}} \tag{3.26}$$

$$P(\underline{x}) = \frac{1}{\sqrt{|\underline{\underline{R}}^T \underline{\underline{DR}}|\sqrt{2\pi}^n}} e^{-\frac{(\underline{x}-\underline{\underline{R}}^{-1}\underline{\nu})^T (\underline{\underline{R}}^T \underline{\underline{DR}})^{-1} (\underline{x}-\underline{\underline{R}}^{-1}\underline{\nu})}{2}} \tag{3.27}$$

If we let $\mu = \underline{\underline{R}}^{-1}\underline{\nu}$, the mean in the standard bases, and $\underline{\underline{C}} = \underline{\underline{RDR}}^T$, the equation becomes: -

$$P(\underline{x}) = \frac{1}{\sqrt{|\underline{\underline{C}}|}\sqrt{2\pi}^n} e^{-\frac{(\underline{x}-\underline{\mu})^T \underline{\underline{C}}^{-1}(\underline{x}-\underline{\mu})}{2}} \tag{3.28}$$

Since $\underline{\underline{C}}$ is of the form $\underline{\underline{R}}^T\underline{\underline{DR}}$ where $\underline{\underline{D}}$ is diagonal and $\underline{\underline{R}}$ is orthogonal it is a symmetric matrix. The simplicity of the transform makes it obvious that the eigenvectors of $\underline{\underline{C}}$ will include the bases of the distribution space and the eigenvalues will be equal to the corresponding variance σ_i. The standard deviation (confidence) volumes and variance volumes are ellipses rotated freely in n-dimensional space, and are usually found using the eigenvectors and eigenvalues of $\underline{\underline{C}}$, although the one variance volume is equivalent to $\underline{\underline{C}}$ applied to the unit hypersphere S^{n-1}.

We have therefore defined a general n-dimensional Gaussian distribution N($\underline{\mu}, \underline{\underline{C}}$). The matrix $\underline{\underline{C}}$ is called the covariance matrix of the Gaussian distribution, and we note: -

$$\underline{\underline{C}} = \underline{\underline{R}}^T\underline{\underline{DR}} = \underline{\underline{R}}^T Exp[\underline{z}\underline{z}^T]\underline{\underline{R}} \tag{3.29}$$

$$= Exp[\underline{\underline{R}}^T\underline{z}\underline{z}^T\underline{\underline{R}}] \tag{3.30}$$

$$= Exp[\underline{\underline{R}}^{-1}\underline{z}(\underline{\underline{R}}^{-1}\underline{z})^T] \;=\; Exp[\underline{x}\underline{x}^T] \tag{3.31}$$

Hence $\underline{\underline{C}}$ is of the form: -

$$\begin{pmatrix} < x_1, x_1 > & < x_1, x_2 > & \cdots & < x_1, x_n > \\ < x_1, x_2 > & < x_2, x_2 > & \cdots & < v2, x_n > \\ \vdots & \vdots & \ddots & \vdots \\ < x_1, x_n > & < x_2, x_n > & \cdots & < x_n, x_n > \end{pmatrix} \tag{3.32}$$

where the covariance of x_i and x_j is defined

$$< x_i, x_j > = Exp((x_i - \mu_i)(x_j - \mu_j)) \tag{3.33}$$

and it is important that the notion of $\mathrm{Var}(x_i) = < x_i, x_i >$ is preserved. If we take two probability distributions $\underline{x} \sim N(\mu, C_1)$ and $\underline{y} \sim N(\nu, C_2)$ uncorrelated, and of the same dimension, we may take the probability distribution of their sum: -

$$Exp(\underline{x} + \underline{y}) = Exp(\underline{x}) + Exp(\underline{y}) = \mu + \nu \tag{3.34}$$

$$< x_i + y_i, x_j + y_j > = Exp((x_i + y_i - (\mu_i + \nu_i))(x_j + y_j - (\mu_j + \nu_j))) \tag{3.35}$$

$$= Exp((x_i - \mu_i)(x_j - \nu_j) \; + \; (y_i - \nu_i)(y_j - \nu_j) + (x_i - \mu_i)(y_j - \nu_j) + (y_i - \nu_i)(x_j - \nu_j)) \tag{3.36}$$

$$= Exp((x_i - \mu_i)(x_j - \nu_j)) + Exp((y_i - \nu_i)(y_j - \nu_j)) + Exp((x_i - \mu_i)(y_j - \nu_j)) + Exp((y_i - \nu_i)(x_j - \nu_j)) \tag{3.37}$$

$$=< x_i, x_j > + < y_i, y_j > + Exp(x_i - \mu_i)Exp(y_j - \nu_j) + Exp(y_i - \nu_i)Exp(x_j - \nu_j) \tag{3.38}$$

$$=< x_i, x_j > + < y_i, y_j > \tag{3.39}$$

3.2.3 Kalman-Bucy Filtering

The Kalman-Bucy filter algorithm is a recursive solution to the discrete-data linear filtering problem [2]. It was introduced by Rudolph E. Kalman in 1960 and refined by Kalman and R. Bucy. In it's simplest form, it takes imprecise data in a linear system with Gaussian errors, and forms the best estimate of the system's current state. It is commonly used in autonomous systems, particularly assisted navigation, and was the foundation for the localisation system in 2002, and again in 2002.

A Kalman-Bucy filter is a solution for the problem of trying to estimate the state $\underline{x} \in \Re^n$ (though there is no reason for the restriction to the real numbers), of a discrete temporal controlled process, governed by the stochastic equation :-

$$\underline{x}_k = \underline{\underline{A}}\underline{x}_{k-1} + \underline{\underline{B}}\underline{u}_{k-1} + \underline{w}_{k-1} \tag{3.40}$$

The $n \times n$ matrix $\underline{\underline{A}}$ in (3.40) is referred to as the state transition matrix, and relates the state at timestep k-1 with the state at timestep k. The $n \times l$ matrix $\underline{\underline{B}}$ relates the optional input $\underline{u}_{k-1} \in \Re^l$ to the state $\underline{x}_k$. Finally, the stochastic

[2]Kalman, R. E. 1960. "A New Approach to Linear Filtering and Prediction Problems," Transaction of the ASME–Journal of Basic Engineering, pp. 35-45 (March 1960)

variable $\underline{u}_{k-1}$ represents the process noise, assumed to be white (uncorrelated) with Gaussian distribution $\underline{u}_{k-1} \sim N(0, \underline{\underline{Q}})$, where $\underline{\underline{Q}}$ is referred to as the process noise covariance.

The state is estimated through a series of observations $\underline{z}_k \in \Re^m$, assumed to be governed by the stochastic equation :-

$$\underline{z}_k = \underline{\underline{H}}\underline{x}_k + \underline{v}_k \tag{3.41}$$

The $m \times n$ matrix $\underline{\underline{H}}$ in (3.41) relates the state to the measurement, and may change with each timestep (as may other inputs mentioned above). The stochastic variable $\underline{v}_k$ represents the measurement noise, assumed to be white (uncorrelated) with Gaussian distribution $\underline{v}_{k-1} \sim N(0, \underline{\underline{R}})$, where $\underline{\underline{R}}$ is referred to as the measurement noise covariance. It is also worth noting that the process and measurement noise are assumed to be independent of each other.

The Kalman filter algorithm can be broken down into a cycle of two seperate phases (See Fig. 3.1), each strongly linked to one of the equations above. The first of the phases is the prediction phase, the current state is projected forward in time and an initial estimatation is taken, referred to as the a priori estimation. The second phase is the correction phase, the prediction earlier calculated is compared to measurements received and a new estimation is taken, referred to as the a posteriori estimation. For notation, at timestep k, $\underline{x}_k$is the true value of the state, $\underline{\hat{x}}_k^-$the a priori estimate and $\underline{\hat{x}}_k$the a posteriori estimate. We can hence define a priori and a posteriori estimate errors :-

$$\underline{\hat{e}}_k^- = \underline{x}_k - \underline{\hat{x}}_k^- \tag{3.42}$$
$$\underline{\hat{e}}_k = \underline{x}_k - \underline{\hat{x}}_k \tag{3.43}$$

We also define the covariance matrix of the error of the a priori estimation and the covariance matrix of the error of the a posteriori estimation :-

$$\underline{\underline{P}}_k^- = Exp[\underline{\hat{e}}_k^- \underline{\hat{e}}_k^{-T}] \tag{3.44}$$
$$\underline{\underline{P}}_k = Exp[\underline{\hat{e}}_k \underline{\hat{e}}_k^T] \tag{3.45}$$

Earlier, it was mentioned that the Kalman filter algorithm calculates the "best estimate", though it is unclear as to what is the best estimation. The

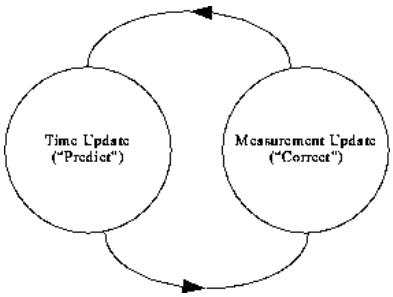


Figure 3.1: The Kalman filter cycle

assumed best estimation is the least squares definition, this means that we want to minimise the sum of the squares of the errors. We may consider any bases for this calculation, though the standard bases are the simplest. Hence, this is equivalent to minimising the trace of the error covariance $Tr(\underline{\underline{P}}_k)$ [3].

The prediction phase equation is

$$\underline{\hat{x}}_k^- = \underline{\underline{A}}\underline{\hat{x}}_{k-1} + \underline{\underline{B}}\underline{u}_{k-1} \tag{3.46}$$

$$\underline{\underline{P}}_k^- = \underline{\underline{A}}\underline{\underline{P}}_{k-1}\underline{\underline{A}}^T + \underline{\underline{Q}} \tag{3.47}$$

Hence we have calculated our a priori state estimate and error covariance.

The correction equations are significantly more complex. To explain the reasons fully would be past the scope of this papeer, but we let the a posteriori state estimate $\underline{\hat{x}}_k^-$ be a linear combination of the a priori estimate $\underline{\hat{x}}_k^-$ and the difference between our measurement and our expected measurement.

$$\underline{\hat{x}}_k = \underline{\hat{x}}_k^- + \underline{\underline{K}}_k(\underline{z}_k - \underline{\underline{H}}\underline{\hat{x}}_k^-) \tag{3.48}$$

The difference $(\underline{z}_k - \underline{\underline{H}}\underline{\hat{x}}_k^-)$ is referred to as the innovation vector, and is the difference between our a priori estimate and the measurement in measurement space. The matrix $\underline{\underline{K}}_k$ is referred to as the Kalman gain, and is a multi-dimensional weight specifying which coordinates of the innovation vector affect each coordinate of the estimation and by how much. We may substitute the Kalman gain into the equations for error covariance and find a minimum, hence find the form

[3]The trace of a matrix is the sum of the diagonal elements. It is invariant under similarity transforms and hence does not change if we take the distribution's natural bases or the standard basis

$$\underline{\underline{K}}_k = \underline{\underline{P}}_k^- \underline{\underline{H}}^T (\underline{\underline{H}} \underline{\underline{P}}_k^- \underline{\underline{H}}^T + \underline{\underline{R}})^{-1} \tag{3.49}$$

We also see that this is intuitively correct since

$$\lim_{\underline{\underline{R}} \to 0} \underline{\underline{K}}_k = \underline{\underline{H}}^{-1} \tag{3.50}$$

$$\lim_{\underline{\underline{P}}_k^- \to 0} \underline{\underline{K}}_k = 0 \tag{3.51}$$

Hence, when we are certain about our measurement (zero covariance) we take the inverse transformation of the measurement as our estimate, and when we are certain about our a piori estimate we leave it unaffected.

We also need to calculate our a posteriori error covariance estimate, which can be done by considering the a priori estimate error covariance, the Kalman gain and the measurement transformation.

$$\underline{\underline{P}}_k = (\underline{\underline{I}} - \underline{\underline{K}}_k \underline{\underline{H}}) \underline{\underline{P}}_k^- \tag{3.52}$$

After each timestep, the process is repeated with the previous a posteriori estimate used to predict the new a priori estimate. At any point the system may provide it's best estimation of the state value by returning the current a posteriori estimate.

The Kalman filtering algorithm is used in the self localisation system by defining the current state as a triplet in x, y, θ space as previously covered. While $x, y \in \Re$ the θ value, while also being real, is only unique in the interval [0,360) (using degrees). The algorithm works perfectly well in this case, we simply restrict θ to the interval and make a few alterations such that 0 and 360 are truely treated the same (i.e. we must be careful as 0 is very close to 359). It must be noted that there is no restriction on the variance of θ, as variance is a measure of accuracy, not of the value itself.

3.2.4 2002 Self Localisation

As with rUNSWift 2002, this year's self localisation system was based on the Kalman filtering algorithm. In 2002, many assumptions were made in order to

simplify both the filtering algorithm, as well as the associated gaussian distributions. Although a three dimensional Gaussian distribution was used, it was assumed that the three dimensions were uncorrelated, and that the variance of the x and y estimate error was the same. Hence, overall, we have the assumption that: -

$$\underline{\underline{P}}_k = \begin{pmatrix} \sigma_{xy} & 0 & 0 \\ 0 & \sigma_{xy} & 0 \\ 0 & 0 & \sigma_\theta \end{pmatrix} \tag{3.53}$$

In general it would be impossible to ensure the maintainance of diagonal covariance when using the Kalman filter algorithm. The measurement of the Kalman Filter algorithm was always taken to be of the same form as the state itself, hence it was assumed that $\underline{\underline{H}} = \underline{\underline{I}}$. By doing this we may have that

$$\underline{\underline{K}}_k = \underline{\underline{P}}_k^- (\underline{\underline{P}}_k^- + \underline{\underline{R}})^{-1} \tag{3.54}$$

$$= \begin{pmatrix} \sigma_{xy}^- & 0 & 0 \\ 0 & \sigma_{xy}^- & 0 \\ 0 & 0 & \sigma_\theta^- \end{pmatrix} \tag{3.55}$$

$$= \begin{pmatrix} (a+\sigma_{xy}^-)^{-1} & 0 & 0 \\ 0 & (a+\sigma_{xy}^-)^{-1} & 0 \\ 0 & 0 & (b+\sigma_\theta^-)^{-1} \end{pmatrix} \tag{3.56}$$

$$= \begin{pmatrix} \frac{\sigma_{xy}^-}{a+\sigma_{xy}^-} & 0 & 0 \\ 0 & \frac{\sigma_{xy}^-}{a+\sigma_{xy}^-} & 0 \\ 0 & 0 & \frac{\sigma_\theta^-}{b+\sigma_\theta^-} \end{pmatrix} \tag{3.57}$$

$$\underline{\underline{P}}_k = (\underline{\underline{I}} - \underline{\underline{K}}_k)\underline{\underline{P}}_k^- \tag{3.58}$$

$$= \begin{pmatrix} \sigma_{xy}^-(1-\frac{\sigma_{xy}^-}{a+\sigma_{xy}^-}) & 0 & 0 \\ 0 & \sigma_{xy}^-(1-\frac{\sigma_{xy}^-}{a+\sigma_{xy}^-}) & 0 \\ 0 & 0 & \sigma_{xy}^-(1-\frac{\sigma_\theta^-}{b+\sigma_\theta^-}) \end{pmatrix} \tag{3.59}$$

Hence the assumptions could be maintained.

The largest problem with this method is the fact that, although the measurement fed into the filter algorithm is calculated directly from the actual observations, the dimension of the measurement and the observation is not necessarily, or even usually, the same. This means that data is either being created or discarded after the observation is initially considered. Consider seeing a single beacon, a common occurance during the game. Observations of angle to the

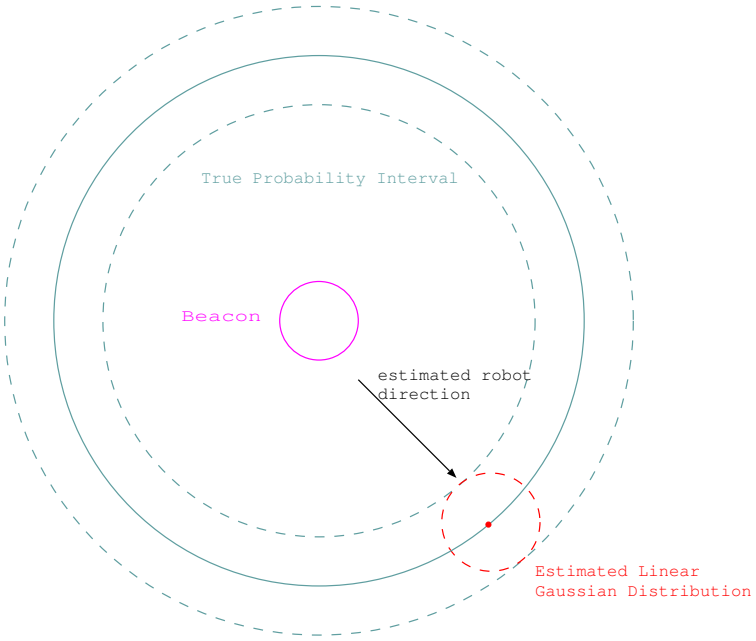


Figure 3.2: Localising off a single beacon

beacon, and distance to the beacon are observed, hence this is a two dimensional observation. If we project into xy space, heading to the beacon provides us with no information, as that heading is possible from anywhere, so our observation becomes a single dimension. The information we feed into the Kalman filter contains a measurement of x and y and is thus two dimensional (See Fig. 3.2).

As we can see, despite our previous observation, the measurement interpretted by the localisation system is always an estimation of the state, and hence projected into xy space two dimensional. In other words, we are telling the Kalman filter algorithm, that we are equally (since there is a single xy variance) as sure about our position around the circular locus, as we are of how far we are away from the beacon. This is despite the fact that we actually have no information about where we are around the circle. The result of this is, that if the robot can see a single beacon only, the Kalman filter is told an exact position and hence, given a series of readings, will assume to know exactly where the robot is. If we catch a glimpse of a different beacon, giving us a different position around the circular locus, it will be largely ignored, as the Kalman filter will believe the series of measurements previously given. This was countered by limiting the size of variance to greater than a threshold. Although this will allow a new observation to correct our previous error, it will also mean that the correct information that is fed to the filter will be quickly ignored.

The artifcact also carries over to observations which have dimension greater than the dimension of the current state. If an observation of two beacons is made, it consists of heading and distance to each, i.e. four dimensions. The

2002 localisation system would use three of these dimensions to calculate a measurement based state, throwing out the fourth (usually the distance to the furthest beacon). Hence they were literally wasting any dimensions of observation greater than three.

The 2003 localisation system can overcome these problems by first removing the assumption that there is a single variance for x, y position and an uncorrelated variance for θ, but rather use a general three dimensional gaussain distribution. Without the assumptions, we no longer require our measurements to be in state space, and hence can have $\underline{\underline{H}}$ other than identity. This means that our measurements given to the Kalman filter can be of dimension other than three, if the observation is of dimension n, $\underline{\underline{H}}$ would be $n \times 3$.

We now have our n-dimensional observations, a facility for using n-dimensional gaussian probability measurements, we need a method of matching our possibly non-linear observations to linear measurements. The facility for this is given through a modified Kalman filter algorithm known as the Extended Kalman Filter.

3.2.5 Extended Kalman Filter

As previously discussed, we would like a simple method of determining a linear gaussian distribution given a possibly non-linear observation. We would like our linear distribution to represent our observation in the best possible way. If we consider a projection into xy space as before, and imagine an observation of a single beacon, the information we may use is simply the distince to that beacon, hence the radius of the circular locus (as seen in Fig.3.2). We should therefore infer no information perpendicular to the line from the robot to the beacon (see Fig. 3.3).

This is not a perfect interpretation of the given observation, but logic tells us that, since we are interpreting a non-linear observation, to a linear distribution, perfection is impossible. In fact, it is this logic that tells us how we can calculate the linear distribution given a general observation, we must answer the question: "what is the best linear approximation to a non-linear function?" Simple calculus tells us that given a function $y = f(x)$, and a point a, the best linear approximation to y at $x = a$ is $f(a) + f'(a)(x - a)$, i.e. the line that passes through $(a, f(a))$ and has the same slope as f at $x = a$.

It must be noted that we are not really looking for a function of a line similar to the example above, but rather a linear function, similar to using the derivative of $f(x)$ above. In fact, we do this by basically treating the function from m variables to n variables as $m \times n$ different functions from a single value to a single value. We can do this by treating the multi-value function as n separate functions, and taking partial derivatives with respect to the m separate inputs.

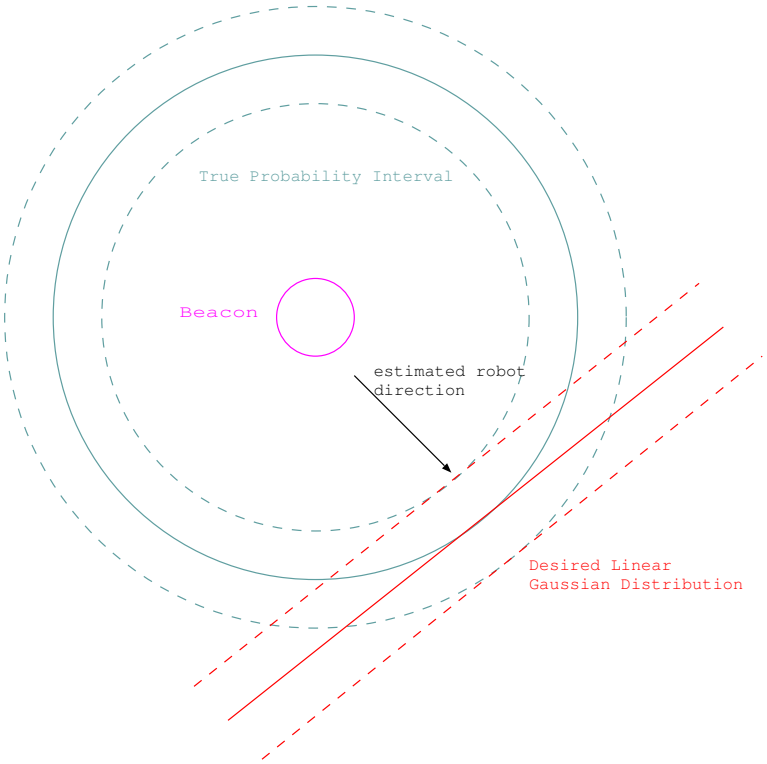


Figure 3.3: Desired localising off a single beacon

This leads us to using the multi-dimensional equivalent of the derivative, the Jacobian matrix. If we assume again that our function $\underline{F}$ maps m values to n values, we may let

$$\underline{F} = \begin{pmatrix} f_1(x_1, .., x_m) \\ \vdots \\ f_n(x_1, .., x_m) \end{pmatrix} \tag{3.60}$$

then, the Jacobian matrix is: -

$$\underline{\underline{J}} = \frac{\partial \underline{F}}{\partial (x_1, .., x_m} \tag{3.61}$$

$$= \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{pmatrix} \tag{3.62}$$

We may now apply the Kalman filtering algorithm, instead of our linear state to observation equation (2) we may assume

$$\underline{z}_k = \underline{h}_(\underline{x}_k) + \underline{v}_k \tag{3.63}$$

It is important to note that, as previously stated, the matrix $\underline{\underline{H}}$ may change with each timestep. Our equations 3.48, 3.49 and 3.52 still work after setting

$$\underline{\underline{H}}_k = \frac{\partial \underline{h}_k}{\partial \underline{x}_k} \tag{3.64}$$

and using $\underline{\underline{H}}_k$ in place of $\underline{\underline{H}}$, except when calulating the innovation vector, where we may use the true function of the a priori state. It is also important to note that the same technique may be used with the state transition function, such that the next state need not necessarily be a linear function of the previous state, but this complexity is not necessary for our model. Another complexity we need not deal with is the situation where the observation is also a function of the stochastic variable $\underline{v}_k$, where the use of Jacobian matrices as a linear estimate may also be used. This is not needed in our system as the observations carry natural variances in common space to observation itself.

3.2.6 Kalman Prediction Update 2003

The Kalman prediction update in the 2003 localisation system is triggered by the actuator system, i.e. when the robot moves. The state prediction is calculated by taking our previous estimation and adding the movement actuator control reports. We let our optional input $\underline{u}_{k-1}$ equal the movement estimated by actuator control

$$\underline{u}_{k-1} = \begin{pmatrix} \Delta f \\ \Delta l \\ \Delta t \end{pmatrix} \tag{3.65}$$

Here our f represents forward, l left, and t our turn to the left from straight ahead in degrees, and they may all be negative. We take each ofthese independently, according to our original estimated position (i.e. assume forward and left occur before turn). Unfortunately, the turn we are given by actuator is about the centre of the body, whilst our θ axis (i.e. $x = 0\, y = 0$ is located at the base of the neck. This leads us to using a non-linear function $\underline{b}_{k-1}$ rather than the matrix $\underline{\underline{B}}$, though we never use the linearity of this matrix and hence it is of no consequence. It also leads to us using our a priori heading estimate, although this is also insignificant.

$$\underline{b}_{k-1}(\underline{u}_{k-1}) = \begin{pmatrix} \Delta f \cos(\theta^-) \; - \; \Delta l \sin(\theta^-) \; + \; \alpha(\cos(\theta^+) - \cos(\theta^-)) \\ \Delta f \sin(\theta^-) \; + \; \Delta l \cos(\theta^-) \; + \; \alpha(\sin(\theta^+) - \sin(\theta^-)) \\ \Delta t \end{pmatrix} \tag{3.66}$$

α above is a constant factor, representing the length from the robot's centre to the base of it's neck, i.e. the difference between the centre of rotation of the heading and turn.

Our prediction phase also requires us to calculate the stochastic variable $\underline{w}_{k-1}$, or more correctly, it's covariance matrix $\underline{\underline{Q}}$. Our actuator system does not provide us with variances on it's readings and hence our calculation of the covariance matrix is based on rough theoretical outlines. Ideally we would like to take empirical data for our values. Due to taking Gaussian distributions, and the nature of the values being extremely non-Gaussian, a perfect model of the noise would be impossible. Further, it's nature is governed by other robot interference and other unknown factors, hence we work on a theoretical model which is based on a reasonable worse case. Basically, we follow several rules: -

- Even if we do not move ($\underline{u}_{k-1}$=0) we would like our variance, or "unsureness" to grow, since we may be getting pushed by another robot for

example

- We would like our variance to grow in a particular direction at a greater rate, the faster actuator is telling us we are moving in that direction. This is due to several factors, it is accounting for the fact that actuator's readings may be inaccurate and we may be caught on objects and in fact not moving.

Our final value for the process noise covariance $\underline{\underline{Q}}$ comes from a constant growth in each natural dimension (i.e. x, y, θ) as well as a multiple of the square of each update in each dimension, hence it becomes: -

$$\underline{\underline{Q}}_{k-1} = \begin{pmatrix} \beta + \gamma(\Delta x)^2 & 0 & 0 \\ 0 & \beta + \gamma(\Delta y)^2 & 0 \\ 0 & 0 & \beta_\theta + \gamma(\Delta\theta)^2 \end{pmatrix} \tag{3.67}$$

We use the square of the estimated change since variance is a measure of the value squared, rather than the value itself. The values β and β_θ can be greatly varied, depending on the desired effect, for the competition the values stayed at $\beta = 3$, $\beta_\theta = 0.5$. The value of γ was set to a value of 4, since this will mean that, ignoring the constant growth, the 95% confidence interval of the covariance update will pass through zero movement (value should actually be 1.96^2).

This concludes our prediction phase, since the a posteriori estimate is a function of the a priori estimate and not the last timestep estimate, the system simply updates the values of the current estimation, this is done in GPS::GPSMotionUpdate(double,double,double) in vision/gps.cpp.

3.2.7 Kalman Correction Update 2003

The Kalman correction update in the 2003 localisation system is triggered by the vision system, i.e. 25 updates per second. We provide the Kalman filter with a measurement based upon the beacons and goals that the robot identifies. Each object implies a two-dimensional observation, distance and heading. Of course there is no real difference between goals and beacons so they are treated the same. The system interprets the observation of n-dimension as n separate equations, each one being either a heading (in degrees anti-clockwise from straight ahead) or distance. The Jacobian for a single equation such as this, will form a row vector, the Jacobian of the entire measurement is composed of rows of such single Jacobians. If we assume the global coordinates of the object are (b_x, b_y), the function from state to distance becomes (see Fig. 3.4): -

$$d_{(b_x,b_y)}(\underline{x}) = \sqrt{(x-b_x)^2 + (y-b_y)^2} \tag{3.68}$$

$$\frac{\partial d_{(b_x,b_y)}}{\partial \underline{x}} = (\frac{x-b_x}{\sqrt{(x-b_x)^2+(y-b_y)^2}}, \frac{y-b_y}{\sqrt{(x-b_x)^2+(y-b_y)^2}}, 0) \tag{3.69}$$

The function from state to heading becomes: -

$$a_{(b_x,b_y)}(\underline{x}) = \tan^{-1\star}(\frac{b_y-y}{b_x-x}) - \theta \tag{3.70}$$

$$\frac{\partial a_{(b_x,b_y)}}{\partial \underline{x}} = (-\kappa\frac{y-b_y}{(x-b_x)^2 + (y-b_y)^2}, \kappa\frac{x-b_x}{(x-b_x)^2 + (y-b_y)^2}, -1) \tag{3.71}$$

The $\tan^{-1\star}$ differs to the inverse tan function in two ways, firstly the trigonometry is done in degrees not radians, and secondly the function can return two values in the interval [0,360). The use of the C function atan2 or equivalent will eliminate the latter problem. The constant κ is placed in the Jacobian due to the trigonometry being calculated in degrees, and is the value $\frac{180}{\Pi}$. We also must be careful when dealing with θ space, since it is multi-valued when treated as $\Re$. For instance, if our observation for heading to an object is 1°, and our calculated estimate (from $\underline{a}$) is 359°, it is better to take the value of $-1°$ since it is the same value, and using euclidean distance, closer to our observation value. Hence, for an angle measurement, the value in the innovation vector calculated below should never be of magnitude greater than 180.

Hence we may now form the measurement function $\underline{h}_k$ and corresponding Jacobian $\underline{\underline{H}}_k = \frac{\partial \underline{F}_k}{\partial \underline{x}_k}$. The innovation vector and Kalman gain can then be calculated: -

$$\underline{v}_k = \underline{z}_x - \underline{h}_k(\hat{\underline{x}}_k^-) \tag{3.72}$$

$$\underline{\underline{K}}_k = \underline{\underline{P}}_k^- \underline{\underline{H}}_k^T (\underline{\underline{H}}_k \underline{\underline{P}}_k^- \underline{\underline{H}}_k^T + \underline{\underline{R}})^{-1} \tag{3.73}$$

where the matrix $\underline{\underline{R}}$, the measurement noise covariance, is a diagonal matrix whose entries are the variances for each dimension delivered by vision. Updates are then performed to the state accordingly: -

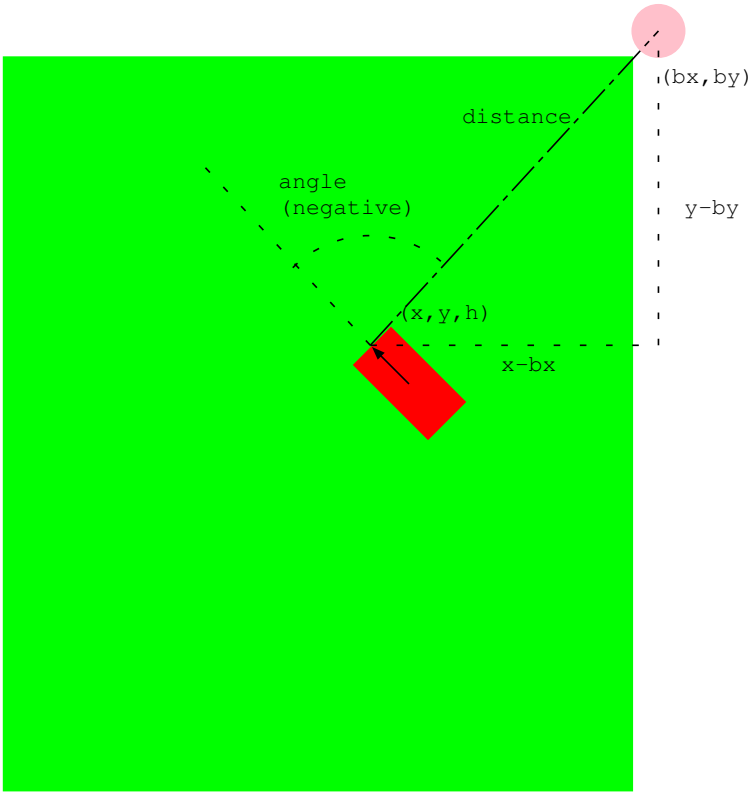


Figure 3.4: Position from a single beacon

$$\underline{\hat{x}}_k = \underline{\hat{x}}_k^- + \underline{\underline{K}}_k \underline{v}_k \tag{3.74}$$

$$\underline{\underline{P}}_k = (\underline{\underline{I}} - \underline{\underline{K}}_k \underline{\underline{H}}_k)\underline{\underline{P}}_k^- \tag{3.75}$$

Hence the system uses these equations, and updates it's current position and covariance with the new values. The method that provides this is GPS::GPSVisionUpdate(), with GPS::addMeasurement() performing the calculations for single objects (innovationvector, Jacobian etc) and GPS::kalmanUpdate() performing the Kalman filter computation.

3.2.8 Active Localisation

Throughout the game there are many situations where the robot will gain more advantage through trying to improve the accuracy of the self localisation system, rather than to concentrating wholely on the ball and the rest of the game. This situation usually arises while moving towards the ball but still being a fair distance away from it. In this case the robot may keep moving towards the ball, turning it's head in a position to see visual objects that will improve it's estimation, and rely on other systems (ball-localisation) to return attention to the ball after visual contact is made. In fact this behaviour is often necessary, since while following the ball the robot generally has a low tilt of the head which restricts the sight of beacons.

Hence the self localisation system must not only be used for the occasional sight of statically placed objects, but also for the purposeful behaviour of looking for such objects. Active localisation is certainly not new in any sense, originally the beacons that were in the robot's field of view were ordered from left to right, and each time active localisation was required, the next object on that list was targetted, pan and tilt calculated and the head would turn to that position. This is a fairly routine algorithm, which does indeed roughly spread the targetted beacons evenly across all six.

The algorithm was greatly improved by asking the question: Which beacon would have the greatest impact on the self localisation system? It is clear that, if the robot does not see any beacons for a period of time, and has been moving in a general direction consistently, then it would gain most benefit by looking for a beacon in that direction as the values sent from actuator control can be checked. Also, if the robot has been repetitively receiving observations of a certain beacon for a period, self localisation would be greatly improved by an observation roughly orthogonal to the known beacons line of sight, since it is the robot's position around the circular locus discussed previously that would be in doubt.

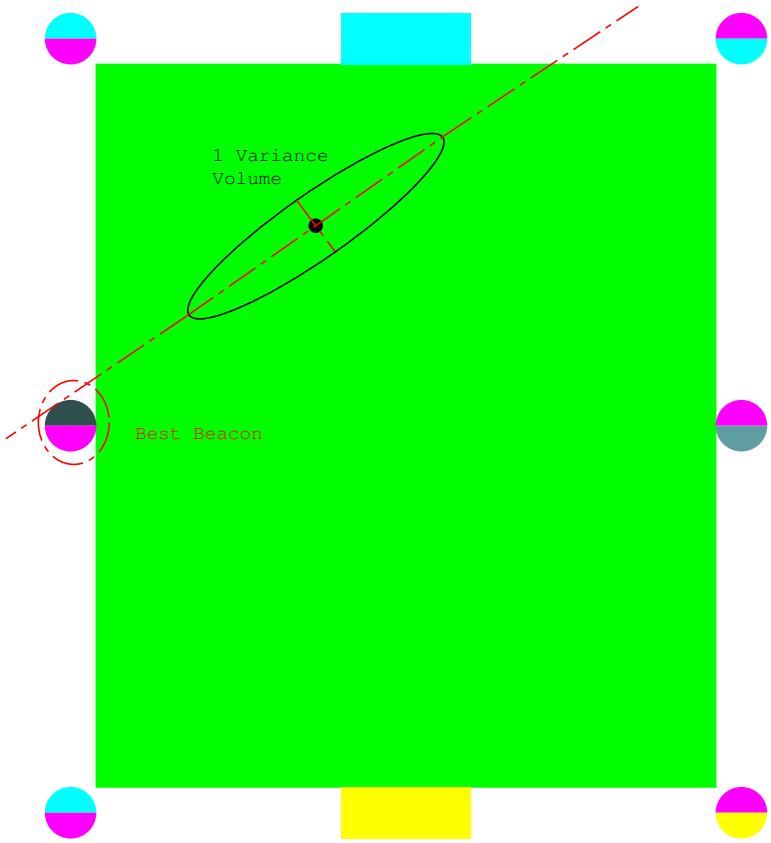


Figure 3.5: One variance volume - a measure of "unsureness" in two dimensions

In general, all beacons will provide equal information in relation to the heading dimension of the robot's own position, so we may project onto the x, y plane for our analysis. The 2×2 upper left submatrix of our positional covariance: -

$$\underline{\underline{\hat{P}}}_{2,2} = \begin{pmatrix} < x, x > & < x, y > \\ < x, y > & < y, y > \end{pmatrix} \tag{3.76}$$

provides us with the covariance of our poition on the x, y plane. If we take a one variance confidence "volume", we may see the area in the x, y that lies within one variance of our estimation (similar to confidence interval). We can see this in Fig. 3.5.

The one variance volume is the effect of the two dimensional covariance matrix $\underline{\underline{\hat{P}}}_{2,2}$, as a linear transform, on the unit circle ($\underline{x} \in S^2$). Therefore the volume is in fact an ellipse, and it's major and minor axis, as shown in Fig.3.5, are defined by the matix's (or linear transform's) eigenvalues and eigenvectors,

which must be defined due to the properties of a covariance matrix. While the variance volume is not similarly equivalent to the confidence interval, which is also an ellipse, the use of the axis of the variance volume is equivalent to the use of the axis of the confidence interval, since they will have identical directions, and the standard deviation's lengths will be the square root of the variance volume's (a strictly increasing function). Hence we may determine the length of the major axis by: -

$$E_{max} = \frac{< x,x > + < y,y > + \sqrt{(< x,x > + < y,y >)^2 + 4 * (< x,y >^2 - < x,x >< y,y >)}}{2} \tag{3.77}$$

The direction of the major axis can be determined by the kernel space: -

$$\underline{e}_{max} \in ker \begin{pmatrix} < x,x > -E_{max} & < x,y > \\ < x,y > & < y,y > -E_{max} \end{pmatrix} \tag{3.78}$$

Hence the angle, unique to 180°, can be determined using the following algorithm, provided floating point inaccuracies are avoided: -

if $(< x,x > -E_{max}) == 0 \,\&\&\, < x,y > == 0)$ **then**

  if $(< y,y > -E_{max}) == 0$ **then**
    return h; {circular, check heading for speed}
  else
    return 0; {first column zero, vector (1,0)}
  end if
else
  return $tan^{-1}(\frac{-< x,y >}{E_{max}})$ {assume kernel OK, use atan2}
end if

Now the visible beacons can be calculated and the camera focuses on the one with approximate heading closest to the output of our algorithm (note mod 180°). The code for this can be found in Behaviours::smartSetBeacon(), that takes on a few factors not mentioned here. The active localisation worked particularly well both in practice under controlled conditions and in game situations.

3.2.9 Self Localisation Evaluation 2003

Precise evaluation of the localisation system is a difficult task to accomplish. The system was undoubtedly greatly improved after the introduction of the

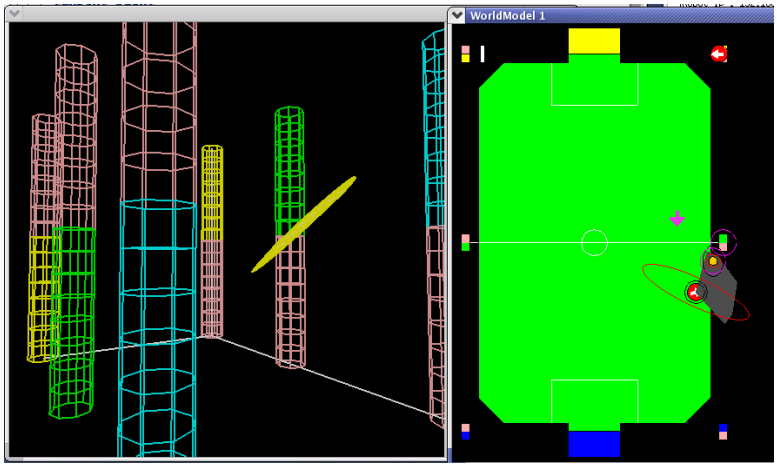


Figure 3.6: Localisation whilst seeing green on pink beacon

full extended Kalman filter algorithm. The system undoubtedly became much smoother and more effective at both maintaining a consistent model and solving the kidnapped robot problem (i.e. performing in situations where our current estimation is extremely off).

We can see the performance of the new system in Fig. 3.6 and Fig. 3.7. In both figures the view on the left shows x, y, θ space, θ up, with the rotated ellipsoid representing the 95% confidence volume of the robots position in this space (i.e. 1.96 standard deviations). The view on the right shows the same system projected into x, y space, the direction of the white arrow indicates estimated position, the red ellipse shows the 95% confidence volume projected into x, y space and the grey region in front shows the 95% confidence volume projected into θ space.

The figures show a robot placed on the field. Fig.3.6 shows the robot localised of a single beacon for a significant period of time. The pink arrow indicates it's rough true position. It is important to note that there is no way for the robot to tell the difference between the true and estimated positions from an observation from the single beacon. We can see however, that the covariance of the estimation has grown large perpindicular to the beacon, indicating the Kalman filter is receiving no information in that dimension. In the 3d display we can see that the ellipsoid actually angles down the spiral locus that represents the possible positions from which the current observation could be made. The true position of the robot would lie on that locus. Fig.3.7 shows the same robot then view another beacon and goal down the field, providing vital information about the inaccuracy of the current estimation. The estimate quickly shifts along the elongated axis of the ellipsoid, and within moments becomes extremely accurate.

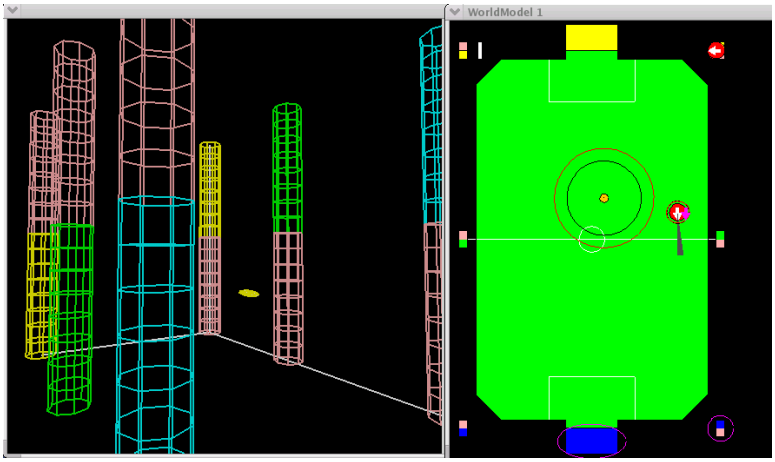


Figure 3.7: Localisation correct after seeing other beacons

3.2.10 Visual Pull

The localisation system for rUNSWift in 2003 also saw the introduction of an intiative to detect collision situations referred to as visual pull. Visual pull is a vector measurement of the Kalman filter corrections for the self localisation estimate. The vector consists of the sum of the corrections $(\underline{\underline{K}}_k(\underline{z}_k - \underline{h}_k(\hat{\underline{x}}_k^-)))$ calculated by the filter over the last second (25 frames) using a sliding window system. These values are the amount by which the visual observations taken shift our estimation, and hence are equal to the error in our prediction values received from the Actuator system. Given a relatively accurate calibration of odemetry leading to theoretically correct values sent by Actuator, the Visual Pull vector represents external forces acting on the robot, assumed to be physical restraints on movement such as having limbs entangled in other robots, friendly or opponent.

The Visual Pull vector is unfortunately extremely volatile and unpredictable. It does show promise in detection of physical impediments but also often produces substantial readings when no physical impediment are present. The self localisation system receives observations in a non-uniform pattern, the inherent error in observations of uncorrelated visual objects means there will be a significant "correction" when switching between current objects observed. Patterns of no visual observations followed by a series of observation also produce a system that does not have corrections spread evenly over time but rather performs large "bulk" corrections in intervals. These problems make a consistent Visual Pull algorithm impossible to implement. Visual Pull was not used in competition at all this year though it did show some promise. Given an improvement in consistency of uncorrelated visual observations, particularly the improvement of line detection leading to a more even spread of observations, the Visual Pull concept should again be considered.

3.2.11 Teammate Localisation

Teammate localisation is the knowledge of the position of all the robot's teammates on the field. This is very useful for team coordination, particularly global positioning. Since 2002 and the introduction of wireless communication teammate localisation has become a simple matter of each teammate broadcasting their current position, as well as confidence information, to all other teammates. Hence, each teammate can maintain a running vector, identical to self-position vector, for each teammate's position and associated variance levels. The information sent is the self localisation vector, as well as the maximum x, y variance, mentioned previously, and heading variance. Fig. 3.8 shows four teammates broadcasting their position and variances to all other teammates. The owner of a particular world model will have a non-circular x, y variance volume and a heading variance shown.

As can be seen, all teammates carry a model of every teammates position and hence may make decisions based upon this. There is also a facility to track teammates similarly to opponents, although this was never used in game conditions as the wireless communication made it redundant as it is relatively inaccurate.

3.3 Ball Localisation

3.3.1 Introduction

The ball localisation system in rUNSWift 2003 involves the estimation of the ball's position on the field, as well as its current velocity. The knowledge of the position of the ball, along with the player's own position, is probably the most fundamental and important knowledge in a game of soccer. The accuracy of the ball's position relative to the player is extremely important, hence information from the vision system is usually used when the player is handling the ball. The ball localisation system's tracking of the position of the ball is used however, for short term memory of the position of the ball, as it is often lost by the vision system, and losing the ball for a single frame should not lead the robot into giving up and going into search mode. The position estimation is also used as a rough estimate of where to start the search for the ball when it is lost. Being first to the ball is of key importance in the game, and a small advantage such as often knowing where to start looking for a lost ball can make a vital impact.

The tracking of ball velocity was a component added to the localisation system in 2003. It is a very difficult problem, since visual information on the ball is imperfect, and even the inaccuracies of the self localisation system have significant effect on the reading of velocity. When working well, the velocity

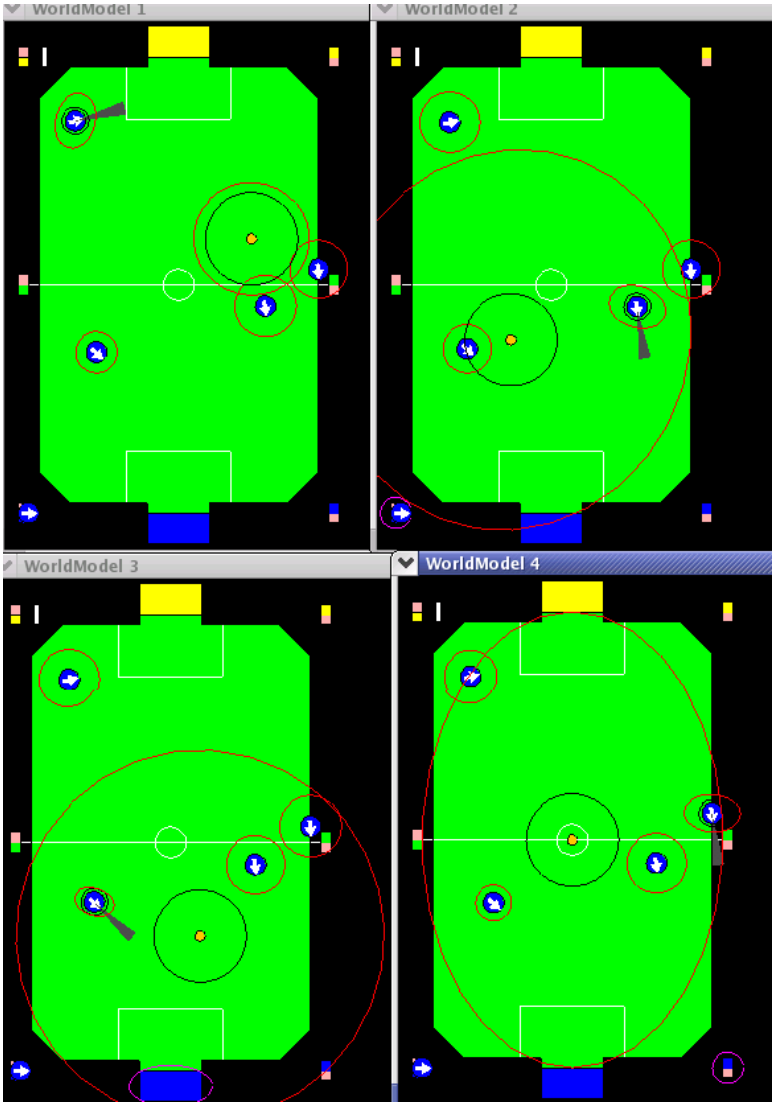


Figure 3.8: Four teammates sharing information on their own position

can be very effective in reading the movement of the ball and gaining control, especially for the keeper with the use of a blocking action.

3.3.2 Ball Position Tracking 2003

The ball's position, similar to the robot's own position, was tracked using an extended Kalman filter. Observations used in the ball position Kalman filter originate from the vision system and are identical in form to that of beacon and goal observations, i.e. a distance estimate and a heading estimate, along with their associated variances. There is some choice in the coordinates used to track the ball's position with both global and local x, y coordinates being viable candidates. The state of the ball's position is defined as a simple x, y vector, representing the position of the ball on the field in identical coordinates as self position, i.e. x across the field, y down the field, origin positioned at the left defensive corner.

The prediction phase of the ball position Kalman filter is very simple, as no update of the estimated position is done, only the variance is enlarged. Hence the update becomes: -

$$\underline{b}_k = \underline{b}_{k-1} + \underline{w}_{k-1} \tag{3.79}$$

$$\underline{w}_{k-1} \sim N(0, \underline{\underline{Q}}_b) \tag{3.80}$$

$$\underline{\underline{Q}}_b = \begin{pmatrix} \beta_b & 0 \\ 0 & \beta_b \end{pmatrix} \tag{3.81}$$

β_b is a constant that effects the overall variance of the ball's position. The value was set to 3 for the competition, this leads to a fairly fast growth of the variance when the robot cannot see the ball, which is ideal since the ball moves quickly and is subject to great position change. It also leads the system to rely more heavily upon recent observations, which is again ideal for the same reason, and also the fact that movement is not factored into the estimation update.

The Observation of a ball is identical in form to that of a beacon, i.e. for distance: -

$$d(\underline{x}, \underline{b}) = \sqrt{(x - x_b)^2 + (y - y_b)^2} \tag{3.82}$$

$$\frac{\partial d}{\partial \underline{x}} = \left(\frac{x - x_b}{\sqrt{(x - x_b)^2 + (y - y_b)^2}}, \frac{y - y_b}{\sqrt{(x - x_b)^2 + (y - y_b)^2}}, 0\right) \tag{3.83}$$

and for heading: -

$$a(\underline{x}, \underline{b}) = \tan^{-1\star}(\frac{y_b - y}{x_b - x}) - \theta \tag{3.84}$$

$$\frac{\partial a}{\partial \underline{x}} = (-\kappa \frac{y - y_b}{(x - x_b)^2 \ + \ (y - y_b)^2}, \kappa \frac{x - x_b}{(x - x_b)^2 \ + \ (y - y_b)^2}, -1) \tag{3.85}$$

The observation of course is a function of both the robot's current position ($\underline{x}$) and the ball's current position ($\underline{b}$). Our previous Kalman filter observation equation involved only variables that we were tracking, where this observation involves other stochastic variables. We can, however, treat these variables as constants and add in extra variance to counter the problem that errors in the self localisation system may lead to errors in the interpretation of the ball observation. We do this by transforming our self localisation covariance $\underline{\underline{P}}_k$ into ball observation space using the Jacobian matrices given above. Hence, if $\underline{h}_k(\underline{x}, \underline{b})$ is our observation equation (note: this will always be two-dimensional) then our measurement noise covariance becomes: -

$$\underline{\underline{R}}_b = \underline{\underline{\hat{R}}}_b + (\frac{\partial \underline{h}_k}{\partial \underline{x}}(\underline{\hat{x}}_k, \underline{\hat{b}}_k^-))\underline{\underline{P}}_k(\frac{\partial \underline{h}_k}{\partial \underline{x}}(\underline{\hat{x}}_k, \underline{\hat{b}}_k^-))^T \tag{3.86}$$

The matrix $\underline{\underline{\hat{R}}}_b$ is the covariance matrix formed from variances sent by the vision system, it is a diagonal matrix whose nth diagonal entry is the variance of the nth dimension of the observation. As can be seen, we evaluate the Jacobian of the observation with respect to the robot's current self localisation a posteriori estimate and the robot's ball-localisation a priori estimate. This gives us the best linear approximation of the affect of the covariance of the self localisation system on the observation, in observation space.

The ball-localisation correction update also requires the calculation of the linear approximation $\underline{\underline{H}}_k = \frac{\partial \underline{h}_k}{\partial \underline{b}}(\underline{\hat{x}}_k, \underline{\hat{b}}_k^-)$. We may again do this by taking the Jacobian with respect to each dimension (distance and heading) and forming the full Jacobian with the calculated rows: -

$$\frac{\partial d}{\partial \underline{b}} = (\frac{x_b - x}{\sqrt{(x - x_b)^2 + (y - y_b)^2}}, \frac{y_b - y}{\sqrt{(x - x_b)^2 + (y - y_b)^2}}) \tag{3.87}$$

$$\frac{\partial a}{\partial \underline{x}} = (-\kappa \frac{y_b - y}{(x - x_b)^2 + (y - y_b)^2}, \kappa \frac{x_b - x}{(x - x_b)^2 + (y - y_b)^2}) \tag{3.88}$$

Again, the constant κ in the above differentiation calculations is factored in

because of the use of degrees, and is equal to the conversion factor from radians, $\frac{180}{\Pi}$.

The Kalman filter equations may now simply be applied to our calculations above, exactly the same way as in the self localisation system. The ball localisation is caculated using the same code as the opponent-tracking system, discussed later, and hence is actually done in information form. The computation is mathematically equivalent to the above, information form was used simply for code reuse, especially to avoid further calculation bugs. The advantages of the information form that are discussed in the opponent tracking system were not used in ball localisation.

3.3.3 Wireless Ball Position

With the introduction of wireless communication in 2002 a new facility for localisation began, the ability to inform teammates, and be informed, of the position of objects on the field. The ball is probably the most obvious candidate for these new abilities, as often the ball is lost, and information on it's location provided by a teammate who can see it would allow the robot to quickly determine it's position.

There are two main ways of performing this sharing of information. The localisation system could either form a data-fused model of the ball's position, or each robot could maintain it's model of the ball's position, broadcasting that information, and acting on the robot's own model and teammates' model in a separate way. For 2003 the second of these systems was chosen, each robot would maintain a separate model of the ball's position, as seen in the previous section, and would broadcast information on that position when required. The information is only broadcast when the robot can see the ball, or the model's variance is small (it has very recently seen the ball). At this point the robot broadcasts the position of the ball in global coordinates, a key reason the ball was tracked globally, as well as a single variance value that indicates the variance of the estimated position. The single variance was calculated as the maximum eigenvalue of the position covariance matrix as discussed in the active localisation system.

Each robot maintains two models of the ball's position, it's own calculated model, as well as one teammates model, which is the one last broadcast that has the smallest variance associated with it. Hence, when receiving a teammates model, it will replace the current wireless model if the incoming model's variance is lower, or it is the robot who generated the current wireless model. Fig. 3.9 shows two teammate robot's placed on the field with two balls (this was done on purpose). Each robot can see one of the ball's (shown in orange on each world model) and is also told the position of the other ball by the teammate (shown in pink on the world model).

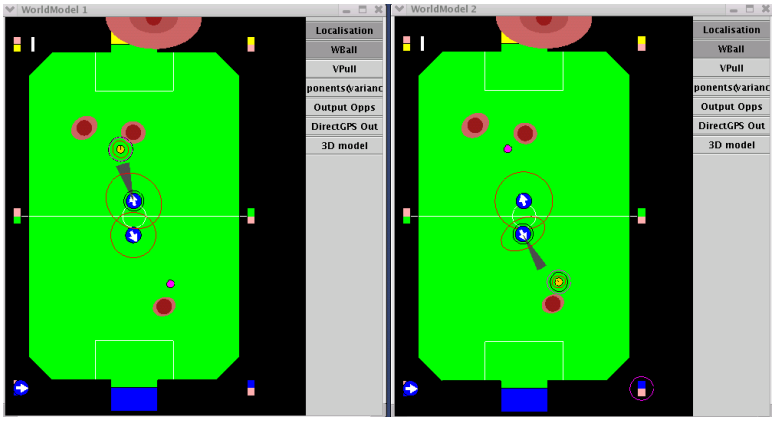


Figure 3.9: Two teammate robot's with two balls on the field

Hence, the localisation system does not worry about merging information from multiple sources, or deciding which of the two models to use in any particular instance, but rather leaves this to the Behaviours system to decide. In general, the Behaviours system always uses the robot's own model if it can currently see the ball, or has very recently seen the ball. This is because there is no error in self localisation systems affecting the robot's own model, since the observations are taken locally. The wireless ball model however, is affected by both errors in the broadcasting robot's self localisation system and the receiver self localisation system. The Behaviours system will only consider the wireless ball model when it has deemed the ball to be lost to it's own model, which is still very important in the game.

The wireless communication was not used to it's full potential when dealing with ball position determination in 2003. Throughout the year, the robot's use of the wireless model encountered great problems and caused many errors in behaviour. A key problem is that often the only robot that can see the ball is the robot closely chasing it. However, while chasing the ball, robots in general have very low lines of sight for a significant period of time, leading to a lack of sight of beacons and ultimately to bad self localisation. Hence, the model of the ball's position sent to teammates is often incorrect. Despite these problem areas that only improvement of the self localisation system would fix, the wireless model general broadcasts correct information and could be used to greater effect.

The use of wireless information on ball position is an area of localisation and behaviour that needs to be looked at in greater depth in the future. The facility for a data fusion model to be created is provided, since the ball position model is currently kept in information form, which is discussed in depth in the opponent tracking section. The sending and receiving of wireless information in the localisation system is done in GPS::getShareInfo() and GPS::processShareInfo().

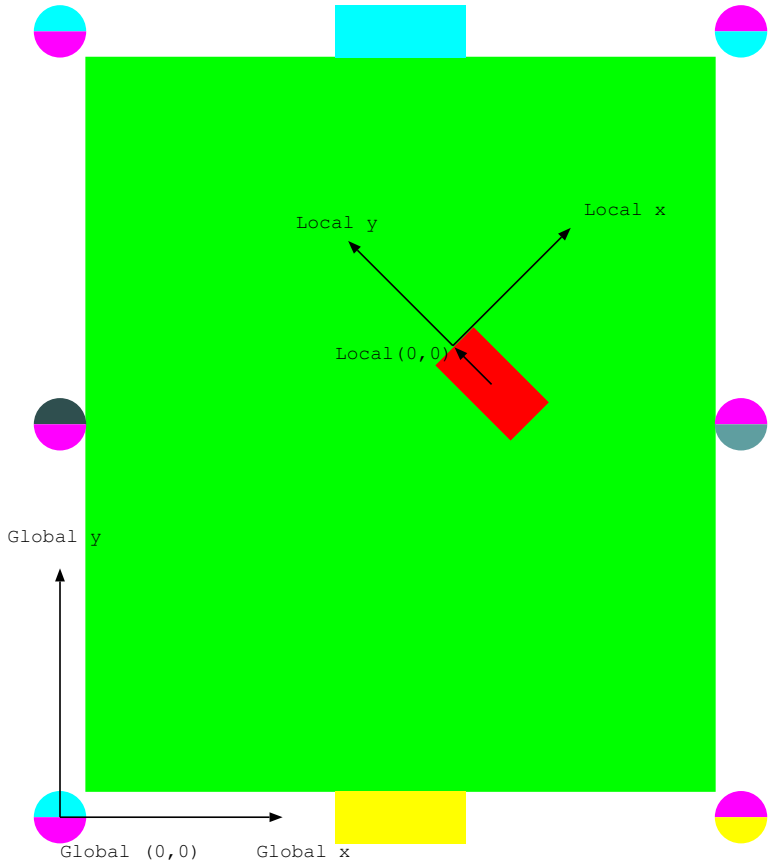


Figure 3.10: Global vs Local Coordinates

3.3.4 Coordinate Conversion

As mentioned above the tracking of objects can be done logically in global coordinates or local coordinates, with respect to the robot. Global coordinates consist of an x, y vector, identical to the self localisation system. The values are based on centimetres and $\underline{0}$ is the centre of the left defensive beacon. In this section global coordinates will be denoted $\underline{x}_G = (x_G, y_G)^T$.

Local coordinates also consist of an x, y vector, with values based on centimetres. $\underline{0}$ is the base of the robot's neck, x directed towards the robot's right, y directed straight ahead (See Fig. 3.10). In this section global coordinates will be denoted $\underline{x}_L = (x_L, y_L)^T$.

Clearly, the use of both, in some form, is required. The Behaviours system of the robot usually requires the use of local coordinates, since the action "move to the ball", for instance, would require the knowledge of the ball's position relative to the robot itself. The sending of information to teammates would require

global coordinates as they otherwise would be uninterpretable. A conversion of coordinates is required to deal with the two systems, and we can do this relative to the $\underline{x}\,(x, y, \theta)^T$ self-position vector: -

$$\begin{pmatrix} x_L \\ x_L \end{pmatrix} = \begin{pmatrix} cos(90-\theta) & sin(90-\theta) \\ -sin(90-\theta) & cos(90-\theta) \end{pmatrix} \begin{pmatrix} x_G - x \\ y_G - y \end{pmatrix} \tag{3.89}$$

$$\begin{pmatrix} x_G \\ x_G \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} cos(\theta-90) & sin(\theta-90) \\ -sin(\theta-90) & cos(\theta-90) \end{pmatrix} \begin{pmatrix} x_L \\ y_L \end{pmatrix} \tag{3.90}$$

These tranforms are used in both dealing with the ball models, opponent tracking and teammate positions. The exact form of the above is not used, the implementation is partly in KalmanInfo2D and Vector classes.

3.4 Ball Velocity Tracking

3.4.1 Introduction

The attempt to track, not only the current position of the ball, but also the current velocity of the ball, was a new feature to the rUNSWift localisation system in 2003. Knowledge of the velocity of the ball would be extremely useful for a robot, since it can act with an estimate of the ball's position in the future, rather than as i the ball was still. For instance, the velocity of the ball would tell the keeper if the ball was moving towards the goal it is defending, and even if it had time to side step and grab or whether a desperate block is required. The introduction of an accurate ball velocity tracking system would, in fact, introduce limitless possibilities of use, as the dynamic nature of the game could be interpreted by the robot.

3.4.2 Initial Method

There are, as usual, several intuitive methods of tracking ball velocity using the Kalman filter algorithm. Perhaps the most natural and theoretically correct method [4] of doing this is to incorporate the velocity with the position of the ball in a single $x, y, \dot{x}, \dot{y}$ vector, maintaining the natural observation interpretation of distance and heading, functions of x and y. The key to this method of velocity tracking is the state transition matrix, which is set to: -

[4] See Bar-Shalom, Yaakov - "Tracking and Data Association" for a 1-dimensional example of this

$$\underline{\underline{A}} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & \mu \end{pmatrix} \tag{3.91}$$

The constant μ is the deceleration factor ($\in [0, 1]$), representing the natural slowing of the ball's motion over time. The above is equivalent to: -

$$\underline{x}_k = \underline{x}_{k-1} + \underline{\dot{x}}_{k-1} \tag{3.92}$$
$$\underline{y}_k = \underline{y}_{k-1} + \underline{\dot{y}}_{k-1} \tag{3.93}$$
$$\underline{\dot{x}}_k = \mu \underline{\dot{x}}_{k-1} \tag{3.94}$$
$$\underline{\dot{y}}_k = \mu \underline{\dot{y}}_{k-1} \tag{3.95}$$

Hence, the $\dot{x}$ and $\dot{y}$ components of the tracked vector represent the difference between the x and y values respectively between timestep k-1 and k. In other words the vector $\underline{\dot{x}} = (\dot{x}, \dot{y})^T$ represents the velocity of the ball in centimetres (as x and y are in centimetres) per frame (i.e. the length of a single timestep). Although the observations received are not functions of the $\dot{x}$ and $\dot{y}$ components, an estimation of their value is propogated through the Kalman filter phases. The Kalman filter automatically attributes a portion of the estimated error in x and y to an error in $\dot{x}$ and $\dot{y}$ updating them accordingly. The rest of the estimated error is attributed to error in the last a posteriori estimate. This proportion is determined (automatically) through the comparison of the covariance matrix. Hence a key to the algorithm is balancing the factor of growth of the position estimate covariance with the velocity estimate covariance.

This particular method, for our purposes, showed promise, but unfortunately experienced many difficulties. The largest of these difficulties was balancing between a system of stagnant velocity, where an actual moving ball was not significantly picked up by the velocity components, and a system of volatile velocity, where an actual still ball was producing velocity estimates of significant magnitude. These problems not only render the velocity estimate almost unusable, but have extremely detrimental effects on the ball position estimate that is important for many sections of behaviour. Results show this particular method does show great potential if mastered, the key problems are: -

- Finding the balance, mentioned above, between the factor of growth of the position estimate and the factor of growth in the velocity estimate.
- Dealing in an environment of extreme stochasticity. For instance, the observation of the ball is also dependent on the current self localisation

estimate, hence severe corrections in the self localisation system causes large inmpacts on the velocity tracking system.

- Modelling deceleration of the ball. Clearly deceleration is not a geometric relationship, and is in fact a very complicated process. A better method of modelling deceleration may greatly improve this and other methods.

3.4.3 Second Method

Another natural method of tracking the velocity of the ball is to completely seperate the velocity estimate from the position estimate, and use a calculated observation of velocity from the past sequence of ball distance and heading observations. This method is made possible since the observation of the ball is equivalent (up to linear estimate) to a measurement of global position x, y of the ball, without this fact the method would be possible, but difficult to implement.

The estimation update for the ball velocity Kalman filter is much the same as previously stated. We have the state transition matrix: -

$$\underline{\underline{A}} = \begin{pmatrix} \mu & 0 \\ 0 & \mu \end{pmatrix} \tag{3.96}$$

where the constant again represents a factor for deceleration ($\mu \simeq 0.99$. The estimate update phase consists of premultiplying the current estimate by $\underline{\underline{A}}$, premultplying the current covariance matrix by $\underline{\underline{A}}$ and adding a constant growth of variance, which was set to 5 for each dimension for the competition. Note that it is fairly clear, since the state transition matix has determinate less than 1, that given a large covariance, tendency will have the covariance matrix actually shrink in the estimate update phase. The simplicity of the state transistion matrix actually tells us that this artifact leads to an "equilibrium" value, given by the equation: -

$$e = \mu e + 5 \tag{3.97}$$

and hence for our value of $\mu = 0.99$ the equilibrium will be at 500. Hence, for x and y, if the variance in that particular dimension is less than 500 it will tend to grow, and if it is greater it will tend to shrink. While this seems extremely worrying, it is actually a good thing for our system, while we cannot put a quantity to it, the ball's speed in the game does not exceed some value, and the equilibrium variance actually model's this situation, i.e. we may not be

sure about the true speed of the ball, but we know it is not going over a certain speed.

The estimation phase of the velocity tracking system is to update our estimates according to the above transformation. The system also checks to see if the ball has hit the edge of the field, in which case we simply set the velocity vector to zero. It may make more logical sense to set the component of velocity perpendicluar to the wall only to zero, however we also found the ball hitting the edge generally slowed the ball's velocity parallel to the wall, to the point of stopping.

The measurement for a velocity correction update is processed from true observations. We saw earlier in self localisation that interpretation of observations to measurements that are easier to use is, in general, detrimental to the Kalman filter algorithm. In this case, our measurement and observation will be of the same dimension, and the covariance of the measurement will be a direct linear representation of the observations covariance. Hence, while not ideal, many of the problems previously discussed are not relevant here.

A velocity measurement is formed from two ball object observations separated by time. For the moment we may assume that these observations are taken over consecutive frames. In the ball position section we saw how to convert a distance/heading ball position observation into a global ball position observation, including the factoring in of the self localisation covariance. Hence, we may assume the two observations have global position observations $N(\underline{x}_{B1}, \underline{\underline{R}}_{B1})$ and $N(\underline{x}_{B2}, \underline{\underline{R}}_{B2})$. Given the one frame difference, we may form a measurement of velocity in centimetres per frame by simply taking a vector of the difference of the two global observations. For covariance, since difference is the same as addition for variance, we add the covariance matrices. Hence our measurement of Ball velocity becomes $N(\underline{x}_{B2} - \underline{x}_{B1}, \underline{\underline{R}}_{B1} + \underline{\underline{R}}_{B2})$ (See Fig. 3.11).

We may also lift the assumption of observations being taken on consecutive frames. Given two observations, we do not know what happened to the ball between the two moments in time, our ball velocity measurement equation follows the assumption that the ball travels in a straight line at constant speed. We limit the number of frames our observations may temporally differ in to 4, given we do not see the ball between these two observations we pretend that for a gap greater than this we have no idea what path it travelled in. We also enlarge the covariance of the measurement for a greter temporal difference since we lose a lot of confidence, we actually take the covariance above to the power of the number of frames difference. If our covariance matrix had a determinant less than one this would actually increase our confidence, but in practice this never comes close to happening, and we want to capture a lot of loss of confidence. Hence, given a temporal difference of n frames between the two observations, our measurement becomes $N(\frac{\underline{x}_{B2} - \underline{x}_{B1}}{n}, (\underline{\underline{R}}_{B1} + \underline{\underline{R}}_{B2})^n)$

Our correction update phase then consists of performing the Kalman filtering

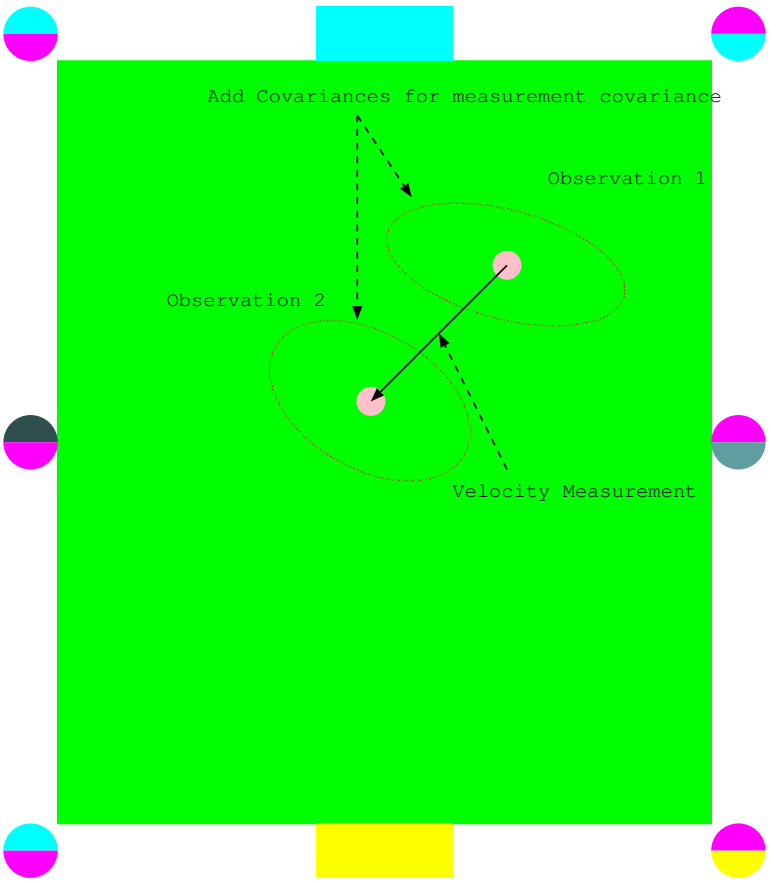


Figure 3.11: Forming a measure of ball velocity from two global position observations

algorithm equations on this measurement. The measurement lies in the same space as our state, hence the non-exteneded Kalman filtering algorithm may be used with $\underline{\underline{H}} = \underline{\underline{I}}_2$, the identity. Our update at this point is therefore extremely simple and is simply substitution into equations previously discussed.

This method produces a much more stable model of the ball's velocity than the first model discussed. As previously discussed, the uses of a ball velocity method are endless, however, our main uses were for action to be taken when the ball travels towards a robot at high speed. The method still produces a lot of outlying estimates and, even when the ball is still, produces significant velocities in random directions. For our purposes, it is much more important to make sure that large velocities are not estimated when the ball is still, than to make sure large velocities are correctly estimated. For these reasons the use of improbability filtering was used to reject observations that were deemed incorrect outliers.

3.4.4 Improbability filtering

Improbability filtering is a method for removing false-positive measurements from degrading a tracking model. False-positive measurements are incorrect observations or measurements, reported with reasonable confidence. They are non-Gaussian white noise and hence severely degrade the performance of the Kalman filter algorithm since it depends on the underlying Gaussian systems.

Although not technically improbability filtering, the first example of this method is when the vision system of the robot changes it's method for determining ball distance. rUNSWift use several methods of determining the distance to the ball, since their performance may vary in different circumstances. The vision system has considerations it performs when deciding the method to use for ball distance, and this inevitably leads to clear cut differences between estimates where the vision system actually changes. While this does not cause a great effect on the ball position tracking system, the switch of the ball distance estimation system causes an inherent difference between consecutive observed positions and hence the velocity measurement taken between such frames is ignored.

The second function of the improbability filter is to avoid using the velocity measurement produced when (or after) a wrongful ball observation is taken by the vision system. It is unavoidable that, at some time, the vision system may produce an observation that is not, in fact, the ball, this may be caused by orange objects outside the field for example. The ball velocity tracking system would produce extremely poor estimates if the measurements it was using were calculated from observations that were, in fact, separate objects. We attempt to detect this situation by placing a limit on the magnitude of the measured velocity. If the measurement reports a velocity of over 10 (i.e. 10 cm/frame =

2.5 m/sec), it ignores the measurment, assuming it is false-positive. A velocity of 10 equates to the ball moving well over half the field in a second, while this is possible, it rarely happens in a real game.

The third function of the improbability filter is to ensure a level of consistency in the measurements of the velocity, as we are mainly concerned with tracking consistent velocity and not as interested in accurately detecting sudden changes in velocity. This is perhaps the purest form of improbability filtering for Gaussian estimations. We compute a measure of probability of the given measurement, the number of standard deviations it is from the current estimate: -

$$numSD = (\underline{x}_M - \underline{x}_{BV})^T \underline{\underline{P}}_{BV}^{-1} (\underline{x}_M - \underline{x}_{BV}) \tag{3.98}$$

where $\underline{x}_M$ is the estimate, $\underline{x}_{BV}$ the current ball velocity estimate and $\underline{\underline{P}}_{BV}$ the current ball velocity estimate covariance. If the measurement is more than a certain number of standard deviations, 1 for the competition - which corresponds to 90% confidence, then we ignore the measurement, as well as doubling the covariance. The doubling of the covariance of our current estimate is to ensure that, given a series of outlying measurements, they will not continue to be thrown out. This means that given a measurement deemed false-positive, the system not only throws the measurement out, but also lowers the confidence in the current estimate, as clearly that "outlier" has brought doubt to the current estimate.

3.4.5 Ball Velocity Evaluation

The tracking of the velocity of the ball was greatly improved by the use of a more stable system and the use of improbability filtering. The basic aims of the velocity tracker, to consistently track a fast moving ball, while preventing large estimates for a still ball, were well met. We can see in Fig. 3.12 a ball with a strong velocity moving towards, and past, the robot. The black line coming out of the ball's position is the robot's ball velocity tracker's estimate of the velocity of the ball in distance per 10 frames, we can see that it estimates a strong velocity very accurately. Fig. 3.13 shows the same situation with the robot observing a ball remaining still on the field. The lack of the black velocity line indicates the velocity estimated by the robot is smaller than a single pixel. While the estimation of velocity of a still ball is not always this accurate, the system in general estimates a velocity small enough to be approximated as zero by the behaviour system. As previously discussed, this is extremely important if the behaviours system acts appropriately for a ball with an appropriate velocity.

The ball velocity estimate was used in velocity prediction for forwards, as

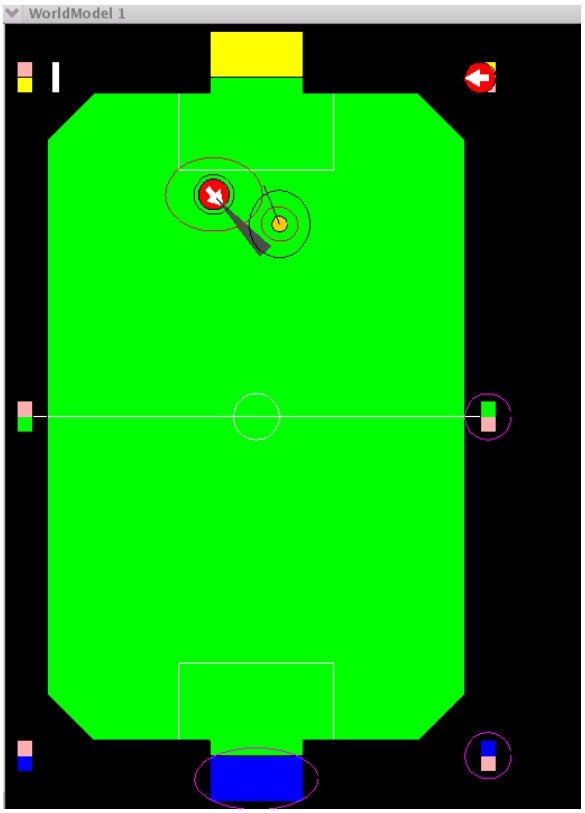


Figure 3.12: World model given observations of a ball moving towards the robot

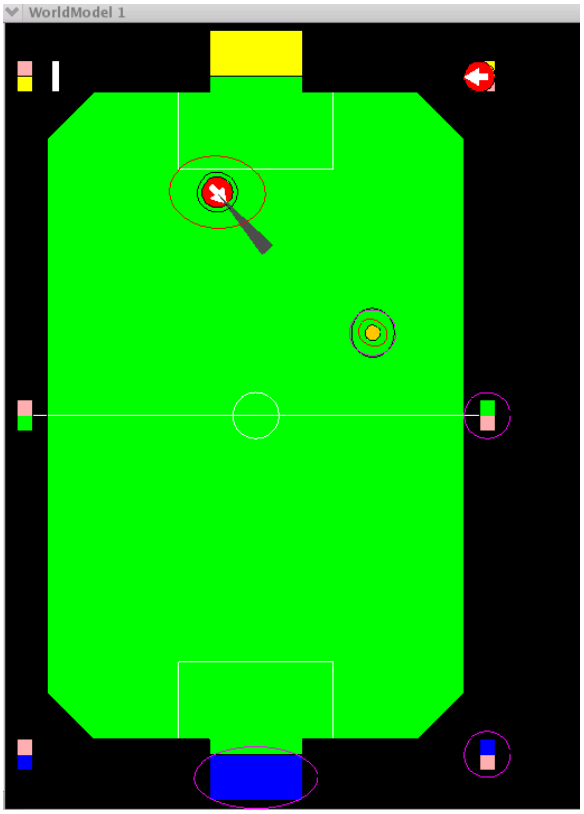


Figure 3.13: World model given observations of a still ball

well as velocity prediction and blocking in the goalie (see relevant sections). These behaviours had some problems, and took precautions of their own to avoid false-positive velocity predictions. However, they showed a lot of promise throughout the year.

The ball velocity tracking system, with work, could be improved. Despite the precautionary steps of the current system consistency is still it's major problem. Given an improvement the estimation of the ball velocity would not only be valuable in the circumstances it was employed in this year, but also other areas including global strategy and, most importantly, passing. Undoubtedly work in this area of the game will continue to be important.

3.5 Opponent Tracking

3.5.1 Introduction

Keeping track of opponents in general adversary game playing is an important factor in developing intelligent behaviour. In robotic soccer this currently consists of the problem of knowing where the opponent players are positioned on the field. This problem is inherently difficult since the visual detection of robots is relatively poor, and the observation frequency is low, since we need to track four entities in general, though we usually receive few observations. Our aim is to track the approximate position of the opposing team, hence we are not interested in fine accuracy as we would be with our own position, or especially in the case of the ball. For these reasons we developed a distributed sensor network for opponent tracking. This consists of multiple sensor nodes (ie our team's robots), which are connected with a temporal constraint, since our robots do not communicate every frame (the cycle period of the sensor), but rather wait for a period of several frames. The processing of the network is also done in a distributed fashion, with each node keeping it's own model of the opponents' positions. The model we developed is based on Information Form Kalman Filtering.

The problem of opponent tracking is not only complicated by the need for a data fusion model, but also due to the nature of the objects. The system is attempting to track four identical objects given sporadic observations. This, of course, leads to the problem of matching particular observations with a particular entity in the current estimation. This problem is an extremely difficult one, especially mixed with a decentralised sensor network where multiple sensors are performing observation matching on the same estimate for separate observations.

3.5.2 Data Fusion and Decentralised Sensing Networks

A decentralised sensing network, as mentioned above, consists of a network of sensor nodes, in our case robots, each with it's own processing facility, which together need no central entity for information convergence. Data fusion is the method of convergence of that data requiring only the individual sensor nodes [5]. Durrant-Whyte and Mike Stevens characterise a decentralised data fusion system by three constraints: -

- There is no single central fusion centre; no one node should be central to the successful operation of the network
- There is no common communication facility; nodes cannot broadcast results and communication must be strictly node-to-node basis
- Sensor nodes do not have any global knowledge of sensor network topology; nodes should only know about connections in their own neighbourhood

For our purposes, we relax the second constraint, since data is in fact broadcast by a single node to all others, and hence the third constraint carries no real meaning. Despite this the problem remains the same, since the core constraint is the lack of a centralised unit, which is necessary in our situation. The advantages of such a network include scalability (adding of extra nodes is transparent), survivability (the system can survive the failure of any on-line loss of sensing nodes), and modularity (nodes are constructed and programmed in a modular fashion, as in our case, they are identical).

As with the previous systems employed by the localisation system, opponents are tracked using Gaussian distributions and the Kalman filtering algorithm. We use a decentralised form of the algorithm known as the Information Form Extended Kalman Filter (IFEKF).

3.5.3 Information Form Extended Kalman Filter

The information form Kalman filter is mathematically equivalent to the general EKF, and is a simple recasting of the equations seen previously. The assumptions made of the system are identical to those seen previously and hence our state transition and estimation equations are also identical to those seen previously, i.e.

[5] For a full introduction into the problem of data fusion, especially in the use of information form Kalman filters, seeData Fusion in Decentralised Sensing Networks - Hugh Durrant-Whyte and Mike Stevens

$$\underline{x}_k = \underline{\underline{A}}\underline{x}_{k-1} + \underline{\underline{B}}\underline{u}_{k-1} + \underline{w}_{k-1} \tag{3.99}$$
$$\underline{z}_k = \underline{h}_(\underline{x}_k) + \underline{v}_k \tag{3.100}$$

We should note however, an extended form of the state transition equation could also be used.

The IFEKF is based around the recasting of the state $\underline{x}_k$ and error covariance matrix $\underline{\underline{P}}_k$into information form: -

$$\underline{y}_k = \underline{\underline{P}}_k^{-1}\underline{x}_k \tag{3.101}$$
$$\underline{\underline{Y}}_k = \underline{\underline{P}}_k^{-1} \tag{3.102}$$

These variables are well defined due to the restrictions on Gaussian distributions, particularly covariance matrices being symmetric positive definite, and are in fact equivalent to the original form since reverse transformations are identical (i.e. information form of information form is the original form). The estimation update phase currently consists of converting back to normal form and applying the previous estimate update procedure, this could perhaps be performed without the conversion, which would be of great benefit. The correction update equations are: -

$$\underline{\underline{H}}_k = \frac{\partial \underline{h}_k}{\partial \underline{x}} \tag{3.103}$$
$$\underline{y}_k = \underline{y}_k^- + \underline{\underline{H}}_k^T\underline{\underline{R}}^{-1}(\underline{z}_k + \underline{\underline{H}}_k\hat{\underline{x}}_k^- - \underline{h}_k(\hat{\underline{x}}_k^-)) \tag{3.104}$$
$$\underline{\underline{Y}}_k = \underline{\underline{Y}}_k^- + \underline{\underline{H}}_k^T\underline{\underline{R}}^{-1}\underline{\underline{H}}_k \tag{3.105}$$

We notice that, dealing with a non-extended example, $\underline{\underline{H}}_k\hat{\underline{x}}_k^- = \underline{h}_k(\hat{\underline{x}}_k^-)$ and hence the calculation of normal form would not be needed. This is the main reason that a more efficient method of the estimation phase was not developed, since the calculation of normal form every cycle was required anyway.

First, we establish that these updates are indeed equivalent to the standard EKF updates. In the following we use inverse matrices and other tools that require the assumption on the matrices, these assumptions are met through the covariance matrices being positive definite and symmetric, however it is left up to the reader to verify this. We begin with the covarince update, the first statement is the equivalence to prove and all subsequent statements are

reversible equivalences, each step may require some thought but are only stated for briefness: -

$$(\underline{\underline{Y}}_k^- + \underline{\underline{H}}^T\underline{\underline{R}}^{-1}\underline{\underline{H}})^{-1} = (\underline{\underline{I}} - \underline{\underline{P}}_k^-\underline{\underline{H}}^T(\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}})^{-1}\underline{\underline{H}})\underline{\underline{P}}_k^- \tag{3.106}$$

$$(\underline{\underline{P}}_k^-)^{-1}(\underline{\underline{Y}}_k^- + \underline{\underline{H}}^T\underline{\underline{R}}^{-1}\underline{\underline{H}})(\underline{\underline{P}}_k^-)^{-1} = (\underline{\underline{P}}_k^-)^{-1}(\underline{\underline{I}} - \underline{\underline{P}}_k^-\underline{\underline{H}}^T(\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}})^{-1}\underline{\underline{H}}) \tag{3.107}$$

$$(\underline{\underline{P}}_k^- + \underline{\underline{P}}_k^-\underline{\underline{H}}^T\underline{\underline{R}}^{-1}\underline{\underline{HP}}_k^-)^{-1} = (\underline{\underline{P}}_k^-)^{-1} - \underline{\underline{H}}^T(\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}})^{-1}\underline{\underline{H}} \tag{3.108}$$

$$\begin{aligned}\underline{\underline{I}} &= \underline{\underline{I}} + \underline{\underline{P}}_k^-\underline{\underline{H}}^T\underline{\underline{R}}^{-1}\underline{\underline{H}} \\ &\quad - \underline{\underline{P}}_k^-\underline{\underline{H}}^T\underline{\underline{R}}^{-1}\underline{\underline{HP}}_k^-\underline{\underline{H}}^T(\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}})^{-1}\underline{\underline{H}} \\ &\quad - \underline{\underline{P}}_k^-\underline{\underline{H}}^T(\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}})^{-1}\underline{\underline{H}} \qquad (3.109)\\ \underline{\underline{0}} &= \underline{\underline{R}}^{-1} - \underline{\underline{R}}^{-1}\underline{\underline{HP}}_k^-\underline{\underline{H}}^T(\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}})^{-1} \\ &\quad - (\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}})^{-1} \qquad (3.110)\\ \underline{\underline{0}} &= \underline{\underline{R}}^{-1} - \underline{\underline{R}}^{-1}(\underline{\underline{I}} - \underline{\underline{R}}(\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}})^{-1}) \\ &\quad - (\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}})^{-1} \qquad (3.111)\\ \underline{\underline{0}} &= \underline{\underline{R}}^{-1} - \underline{\underline{R}}^{-1} + (\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}})^{-1} \\ &\quad - (\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}})^{-1} \qquad (3.112)\\ \underline{\underline{0}} &= \underline{\underline{0}} \qquad (3.113)\end{aligned}$$

Hence the algorithms are equivalent for calculating a posteriori covariance. We now similarly check for a posteriori state estimate and use the above identity: -

$$\hat{\underline{x}}_k^- + \underline{\underline{P}}_k^-\underline{\underline{H}}^T(\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}})^{-1}(\underline{z}_k - \underline{h}(\hat{\underline{x}}_k^-)) = \underline{\underline{P}}_k(\underline{\underline{Y}}_k^-\hat{\underline{x}}_k^- + \underline{\underline{H}}^T\underline{\underline{R}}^{-1}(\underline{z}_k + \underline{\underline{H}}\hat{\underline{x}}_k^- - \underline{h}(\hat{\underline{x}}_k^-))) \tag{3.114}$$

$$\begin{aligned}&= \underline{\underline{P}}_k((\underline{\underline{Y}}_k^- + \underline{\underline{H}}^T\underline{\underline{R}}^{-1}\underline{\underline{H}})\hat{\underline{x}}_k^-) \\ &\quad + \underline{\underline{P}}_k^-\underline{\underline{H}}^T\underline{\underline{R}}^{-1}(\underline{z}_k + \underline{\underline{H}}\hat{\underline{x}}_k^- - \underline{h}(\hat{\underline{x}}_k^-))) \qquad (3.115)\\ &= \underline{\underline{P}}_k\underline{\underline{Y}}_k\hat{\underline{x}}_k^- + \underline{\underline{P}}_k\underline{\underline{H}}^T\underline{\underline{R}}^{-1}(\underline{z}_k\underline{h}(\hat{\underline{x}}_k^-))) \qquad (3.116)\end{aligned}$$

$$\underline{\underline{P}}_k^-\underline{\underline{H}}^T(\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}})^{-1} = (\underline{\underline{Y}}_k^- + \underline{\underline{H}}^T\underline{\underline{R}}^{-1}\underline{\underline{H}})^{-1}\underline{\underline{H}}^T\underline{\underline{R}}^{-1} \tag{3.117}$$

$$(\underline{\underline{Y}}_k^- + \underline{\underline{H}}^T\underline{\underline{R}}^{-1}\underline{\underline{H}})\underline{\underline{P}}_k^-\underline{\underline{H}}^T = \underline{\underline{H}}^T\underline{\underline{R}}^{-1}(\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{R}}) \tag{3.118}$$

$$(\underline{\underline{I}} + \underline{\underline{H}}^T\underline{\underline{R}}^{-1}\underline{\underline{HP}}_k^-)\underline{\underline{H}}^T = \underline{\underline{H}}^T\underline{\underline{R}}^{-1}\underline{\underline{HP}}_k^-\underline{\underline{H}}^T + \underline{\underline{H}}^T \tag{3.119}$$

Hence we have equivalence of the calculation of a posteriori estimate in the two models.

The essential advantage of using information form are the correction updates, which we want to broadcast over the sensor network, and are of the form: -

$$\underline{y}_k = \underline{y}_k^- + \underline{i}_k \tag{3.120}$$

$$\underline{\underline{Y}}k = \underline{\underline{Y}}_k^- + \underline{\underline{I}}_k \tag{3.121}$$

where $\underline{i}_k$ and $\underline{\underline{I}}_k$ are the updates due to observations at time k. This provides a distinct advantage over regular Kalman filtering, in that estimations are formed from linear combinations of observation information. Hence if sensor n last sent distributed update data at time k-α, then it simply sends $\sum_{j=k-\alpha}^{k} \underline{i}_j$ and similar for $\underline{\underline{I}}$, as an update to the other sensors. Therefore, ignoring the complexity of syncronization of updates, all sensors should hold the data (assuming N sensors)

$$\underline{y}_k = \underline{y}_{k-\alpha} + \sum_{n=1}^{N}(\sum_{j=k-\alpha}^{k} \underline{i}_{n,j}) \tag{3.122}$$

and similar for $\underline{\underline{I}}$. This solves a key problem in decentralised data fusion, all nodes hold equivalent data in the long term. The Kalman filter is mathematically equivalent to the regular Kalman filter, therefore the estimation held by all the nodes is equivalent to the estimation made if a single node was to receive all observations by all nodes combined.

3.5.4 Opponent Tracking using IFEKF

Updates for tracking four opponents consist of the above equations applied to four separate models representing x, y global position. The state transition matrix is taken as the identity and the observation equation identical to that of the ball position, i.e.

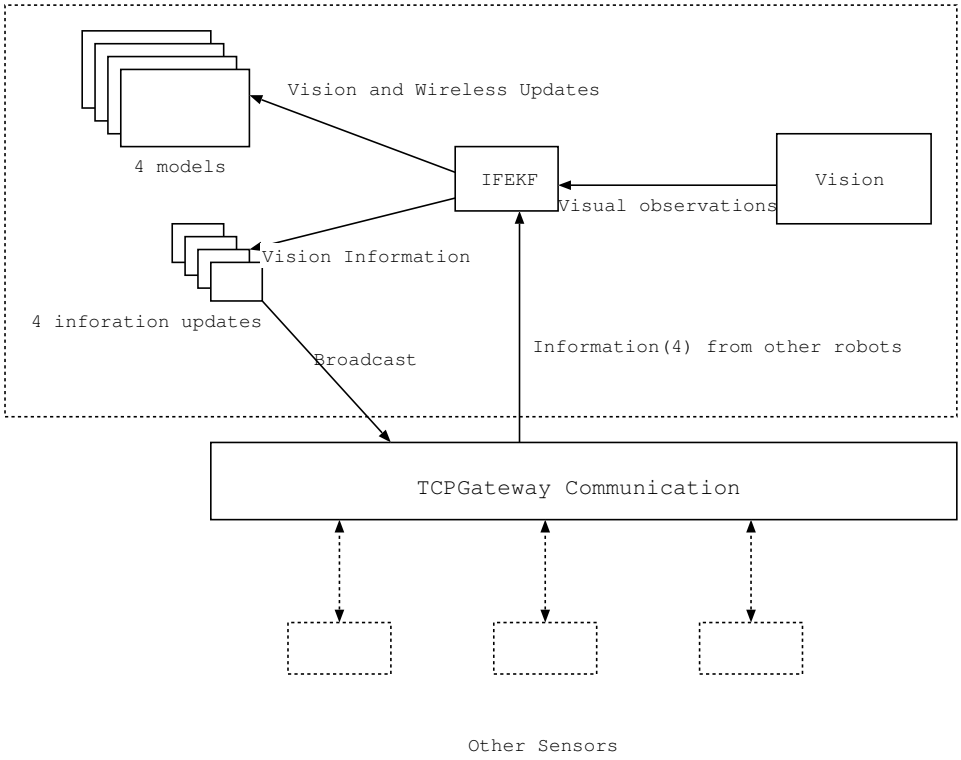


Figure 3.14: Decentralised opponent tracking network

$$d(\underline{x},\underline{b}) = \sqrt{(x-x_b)^2+(y-y_b)^2} \tag{3.123}$$

$$\frac{\partial d}{\partial \underline{x}} = (\frac{x-x_b}{\sqrt{(x-x_b)^2+(y-y_b)^2}}, \frac{y-y_b}{\sqrt{(x-x_b)^2+(y-y_b)^2}}, 0) \tag{3.124}$$

$$a(\underline{x},\underline{b}) =^{\star} \tan^{-1}(\frac{y_b-y}{x_b-x}) - \theta \tag{3.125}$$

$$\frac{\partial a}{\partial \underline{x}} = (-\kappa\frac{y-y_b}{(x-x_b)^2+(y-y_b)^2}, \kappa\frac{x-x_b}{(x-x_b)^2+(y-y_b)^2}, -1) \tag{3.126}$$

We can then substitute these into the IFEKF equations and perform our correction update. As well as update our current model (we are assuming we know which model to update) we keep a running sum of the values $\underline{i}_k$ and $\underline{\underline{I}}_k$, these sums are, at certain intervals decided by systems external to localisation, broadcast over the TCPGateway network and cleared back to zero. Fig. 3.14 is a diagram showing the general form of the decentralised network as seen by a particular module (robot).

3.5.5 Observation Matching

The algorithm described above solves the problem of having multiple sensor nodes with our temporal constraint, however the task of tracking opponent positions still involves other issues. Tracking a team of four opponents requires the estimation of four positional vectors, this indicates that four separate information Kalman filter models need to be used, or a similar approach applied. This leads to a very elementary problem, given an observation, and that all opponents look identical, which model should be updated? There are two simple solutions to this problem, both of which were tried.

Most Probable Model

The first solution is arguably the logical one, choose the most probable model. This approach can be realised through the equation

$$P(O|G_n) = \frac{1}{2\Pi\sqrt{|\underline{\underline{K}}|}} e^{-\frac{\underline{x}^T \underline{\underline{K}} \underline{x}}{2}} \tag{3.127}$$

where O represents the observation, G_n the nth Gaussian probability distribution, $\underline{x}$ the innovation vector and $\underline{\underline{K}}$ the Gaussian covariance projected to observation space. By choosing the Gaussian with the largest of these probabilities, and updating it's Kalman filter model with the observation, we can match individual observations to individual models.

Although this solution is perhaps the mathematically precise one, it suffers from several problems, the most severe of which is that some Gaussians reach outer limits, growing very large, and end up not winning any observations, while other Gaussians win a lot of observations and hence "bounce" between observations.

A variation of the above is to use the same algorithm, except instead of using the probability of the observation given the Gaussian, use the number of standard variations away from the expected value the observation is, this is given by

$$n = e^{-\frac{\underline{x}^T \underline{\underline{K}} \underline{x}}{2}} \tag{3.128}$$

By rewarding the smallest value calculated above, Gaussians that do not attract observations, grow in variance, and hence have a much greater chance of "winning" an observation. However, this method still fails to offer adaption to an opponent's positions quickly enough, suffering especially in the kidnapped robot situation. This method is implemented in the function applyKalmanUpdateSimple() in KalmanInfo2D.cc.

Share the Observation

The second simple solution to the problem of observation matching is to "share" the observation around. We can apply a fraction of an observation to a Gaussian in the IFKF by multiplying the variance of the observation by the inverse of the fraction. This is also equivalent to multiplying the information values $\underline{i}_k$ and $\underline{\underline{I}}_k$ by the desired fraction. Hence, we can award a weight for each Gaussian (n): -

$$w_n = \frac{P(O|G_n)}{\sum_{i=1}^{N} P(O|G_i)} \tag{3.129}$$

and apply the fraction of the given observation to the IFKF. This solution leads to a set of models that can very quickly adapt to changes in the opposition's positions as Gaussians quickly move from areas where there are no observations to places of high observations. However, the solution also leads to problems of multiple Gaussian distributions becoming extremely similar, at which point they are very unlikely to separate. The bulk of observations strongly attract all of the Gaussian distributions, and once the Gaussian distributions become approximately close they will never separate. This method is implemented in the function applyKalmanUpdateWA() in KalmanInfo2D.cc.

Hybrid matching method

After analysing the two simple solutions discussed, we can see that both have their own problems. Ideally, the solution should have the ability for models to adapt to sudden changes in opposition position's that the second method offers, while maintaining a reasonable spread of Gaussian distributions that the first method has. The logical step was to try to create a hybrid method which combined the two, and would hopefully inherit both ideal behaviours. By using the calculated weights above, we can create a new weighting system taking both methods into account. The new weight v can be calculated by

$$v_n = (1 - \alpha)w_n + \delta_{nj}\alpha \tag{3.130}$$

where w_n is the original weight, j is the "winning" Gaussian calculated using standard deviations, δ is the Kronecker delta and α is a constant between 0 and 1. In fact if we take α as 1 we get the first solution, and as 0 we get the second. We can see this solution takes a portion of the observation and assigns it to the "winner", then breaks the compliment portion into pieces according to the second solution. This hybrid solution works extremely well, providing a highly adaptive set of distributions while maintaining a spread over all observation areas. Unfortunately it still inherits some of the problems that the initial solutions had, and it certainly would not be reasonable where acute accuracy of positions was required, but it succeeds in our initial goals, providing a good approximation of the opponent's positions, representing observations from all teammates. This method is implemented in the function applyKalmanUpdateHybrid() in KalmanInfo2D.cc.

Preventing Multiple Wins

The hybrid solution to the pattern matching problem, as well as the most probable model solution, deal with each individual observation for a single frame in a completely modular way. The method takes no consideration of the fact that it may have several observations from a single frame, and hence each observation must be a separate object. This, of course, causing some error when attributing a portion of the observation to the "winning" observation. If a robot receives several observations in a single frame, it is possible, even probable since the observation must be in some proximity tp each other, that an observation may "win" more than one of these observations.

The next optimisation to the observation matching algorithm is to, rather than decide on a most probable Gaussian distribution for a given observation, assign a different Gaussian observation to each observation such that the set assigned is the most probable set of matches for that set of observations. We may determine this, given the probability each observation is each Gaussian distribution, $P(O_i|G_n$, then we desire the set $n_1, n_2, ...$ such that: -

$$P(O_1|G_{n_1})P(O_2|G_{n_2})... \tag{3.131}$$

is maximised. This was done using a simple A* search where 1-P was to be minimised, and the heuristic calculated the maximum probability combination for unassigned values ignoring repeated Gaussians.

This modification, while adding much sophistication to the hybrid method, had a severe problem with the poor quality of visual detection of robots. The problem of visually detecting robots is extremely difficult because of the colours and especially the shape that must be detected. The vision system frequently misses the detection of a robot or, more detrimentally, reports the presence of multiple robots in a position a single robot is positioned. The second error leads to few problems without this modification, since a single Gaussian would win all the observations for the single robot and hence other Gaussians would not be too greatly effected. With this modification however, the correct Gaussian could only "win" a single one of the multiple observations, and hence incorrect Gaussians would be severely shifted as they "win" the extra observations.

This problem was extremely common and lead to the model performing extremely poorly as multiple Gaussians would be pulled toward single opponent positions, and other observations by all teammates would not be able to draw Gaussians to represent the opponent. This modification is certainly a theoretical improvement and would be well employed if visual detection of robots was greatly improved or the tracking of other objects (e.g. multiple balls) where vision is more capable. This modifaction in rUNSWift 2003 was not included in the opponent tracking system, only the standard hybrid matching method was used. This modification is implemented in the function applyKalmanMultiUpdatesHybrid() in KalmanInfo2D.cc, with the aStarSearch template class used to perform the A* search.

3.5.6 Opponent Tracking Evaluation

Opponent tracking for rUNSWift 2003 was an experimental area of localisation, where the introduction of data fusion techniques was hoped to provide the team with an accurate overall distribution of the opponent team on the field. The system built successfully fulfils this aim and provides a strong data fusion base to work with in the future. Fig. 3.15 shows the opponent tracking system with two blue teammates tracking four red opponents, the positions of the opponents are well represented in both the blue teammates' opponent models, even though each teammate can only see two opponents. Fig. 3.16 shows the same situation with the shifting of a single opponent to a separate position, this shift in position is accurately picked up by both of the teammates' models.

Fig. 3.17 shows the same situation again, where opponents have shifted such that they are all positioned in the first teammate's field of vision and not in the second teammate's. This shows that the transition from one teammate's field of vision to another teammate's field of vision is, in general, transparent across all models, and in fact that which teammate's field of vision an opponent is in is completely transparent across the distributed sensor network.

The opponent tracking system is certainly not a system of accuracy, but does

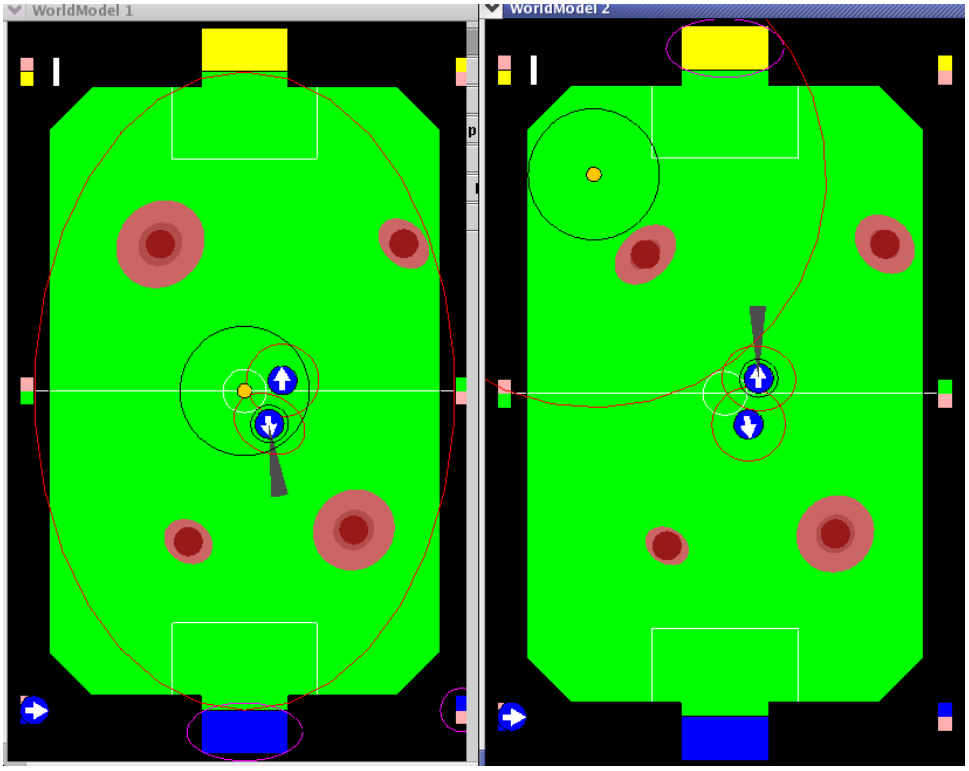


Figure 3.15: Opponent tracking with two teammates

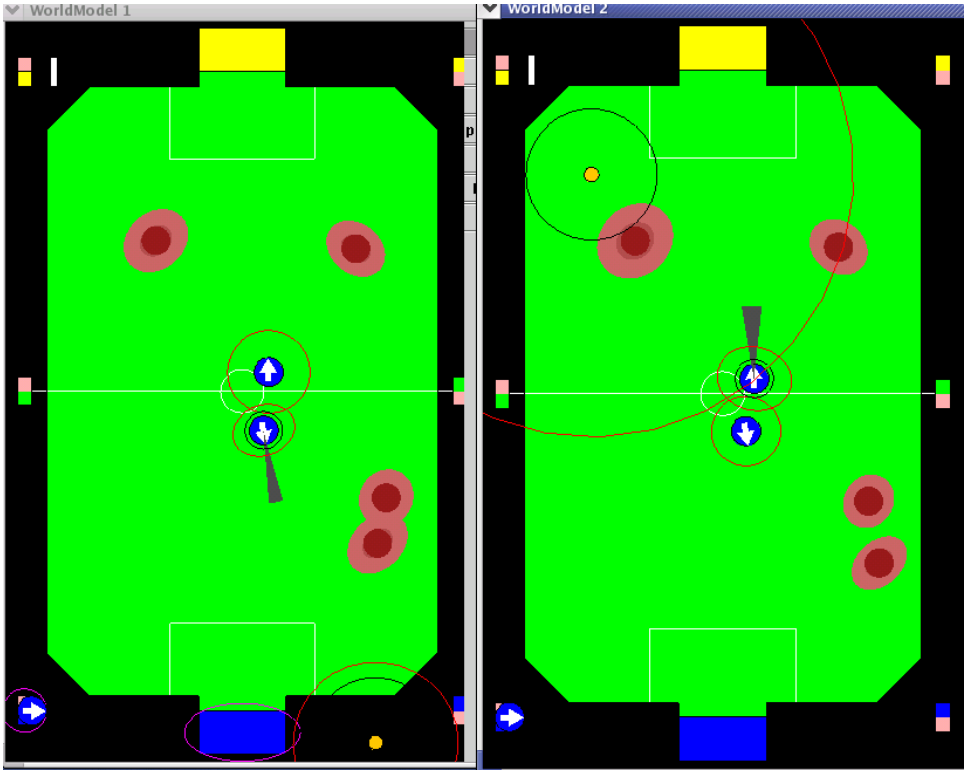


Figure 3.16: Same opponent tracking after shifting of one opponent

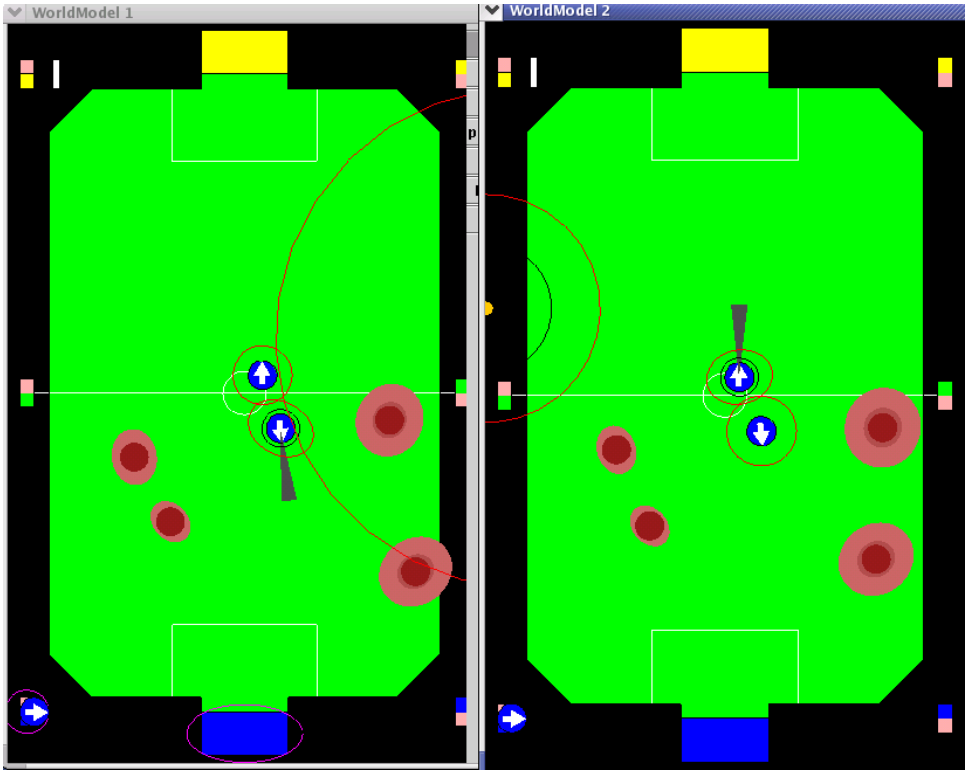


Figure 3.17: Opponent tracking where one teammate providing all information

represent the overall spread of opponent's very well. It also adapts to opponent movement well, even in the taking on and off of opponent robots. The lack of opponent robots (e.g. penalties) will only lead to a Gaussian merging with another opponent robot's position, which is not a problem for our requirements. Currently there is a small bug in the system which leads to Gaussians which lack observations to slide off the field, I think this is largely due to inaccuracies in converting to normal form every cycle for the estimation update, and will hopefully be rectified in the near future. The opponent tracking was used to some benefit this year, as behaviour used it in some circumstances to decide where to move the ball, and other times as a heuristic for where to initially look when performing a vision-based skill (e.g. see visual opponent avoidance kick). The improvement in consistency and accuracy of the opponent tracking system in the future will be an important facet of the rUNSWift team. The use of the opponent tracking system will hopefully grow, I believe it's use in defence and particularly in judging where to move the ball in order to push it up the field will become increasingly beneficial.

3.6 Conclusion

The rewriting of the localisation system was a success in both improving localisation and providing a platform for future years to work from. The self localisation model now incorporates a fully functioning extended Kalman filter, and facilitates the addition of arbitrary observations to the system. The addition of the ball velocity tracking system and the development of a new opponent tracking system were both successful leading to functioning systems that may be analysed and improved in future teams. There are some areas in particular that should be looked at in the future.

3.6.1 Multi-Nodal Distributions

Currently, the extended Kalman filter performs extremelt well given the beacon and goal observation for the correction update. Although it was not coordinated well, the model was found not to be ideal in the detection of line patterns and their use in localisation. The extreme multi-nodal nature of line patterns results in large errors if the wrong node is chosen, as one must be for the single node Kalman filter. Through conversing with other teams at the competition it seemed that a particle filter performed much better for line matching algorithms than the single node Kalman filter approach. Most of those used by other teams were Monte Carlo particle filters, which consist of a cloud of points approximating a probability distribution. In future years it would be advantageous to experiment in using hybrid Kalman filters, in which many weighted Gaussian distributions are used to represent a multi-nodal distribution. Although this would significantly enlarge the system's computational complexity it would also

produce the advantages of both the Kalman filter and particle filter approach.

3.6.2 Velocity Prediction Improvement

The velocity prediction was a significant step forward in the path to developing intelligent behaviour. The system developed this year works at a usable level, but requires extra improbability filtering to avoid the effects of false-positive observations. This situation also leads to the underuse of the estimation, since some true readings are filtered as false-positive. Further development in the area of velocity tracking could lead to great advantage, as more information may be used by the behaviour system, particularly about the future rather than current state.

3.6.3 Automatic and Empirical Variance Tuning

The localisation system currently uses a lot of variances that are rough estimations calculated by various theoretical methods. Last year's localisation report also described the possibility of automating the variance tuning process, specifically moving the robot around the field, informing it of it's current correct position and learning variance values for various objects. This idea can be extended since variances such as process noise may also be automatically calculated by the robot. In theory, the variances could be tuned to some degree without robot being informed of it's current position, since variance is a measure of the fluctuation in observed values, rather than the error. Empirical variance tuning is a similar aproach where values are taken by the robot, and variances may not be automatically shifted but rather edited after human consideration of the values taken. Even this would be a significant improvement on the current method of overestimating variance and hoping for the best.

3.6.4 Further Use of Localisation Information

The current behaviours does not use the localisation system to it's full potential. For the development of more advanced skills, such as global collaboration, passing, possession play and many others, the localisation system must be used to provide the complex information required. The information held by the system currently is accessed in particular circumstances, but there is no consistent deliberation of global position and possible medium-term actions. Although forcing the ball in a desirable direction is an extremely effective strategy, the consideration of the opponents' positions may be used to decide that time may be used, and the ball may be pushed in a direction that is more ideal because of the position of dynamic objects (e.g. push the ball upfield into space, rather

than a definite direction). These capabilities are possible in the near future and should be explored thoroughly.

Chapter 4

Behaviours

4.1 Behaviours Hierarchy

The behaviours module is essentially one enormous decision tree that combines information from the vision, world model and wireless modules in order to define the next set of actions to be performed by the locomotion module. It is the decision making module of the robot that processes the input from the agent's sensors in order to control its effectors. In fact, the decision tree became so large that it started being affectionately referred to as TOIB, which is an acronym for the Tree Of Infinite Branches. However there is of course a hierarchical structure to the tree which breaks it into smaller, more manageable sub trees.

The tree is split into 3 main levels. The top most level contains a number of strategies that can be used. By different strategies, we mean different philosophies to how all 4 robots on the team should play. For example, one strategy could have the robots spread out, passing the ball around. Another could have the robots being really aggressive, and another could have all robots playing really defensively. Essentially this level existed so that different strategies could be developed and played against one another. However, during a game, one might program the robots to change strategies depending on the current score, or how much time is left in the half.

Each strategy incorporates a number of roles that the robots can be executing. Examples include attackers, supporters and defenders. These strategies are typically dynamically switched between during a game, except for the goal keeper role which is fixed to a particular robot.

Each of the roles use a number of skills. This is where the hierarchy of the

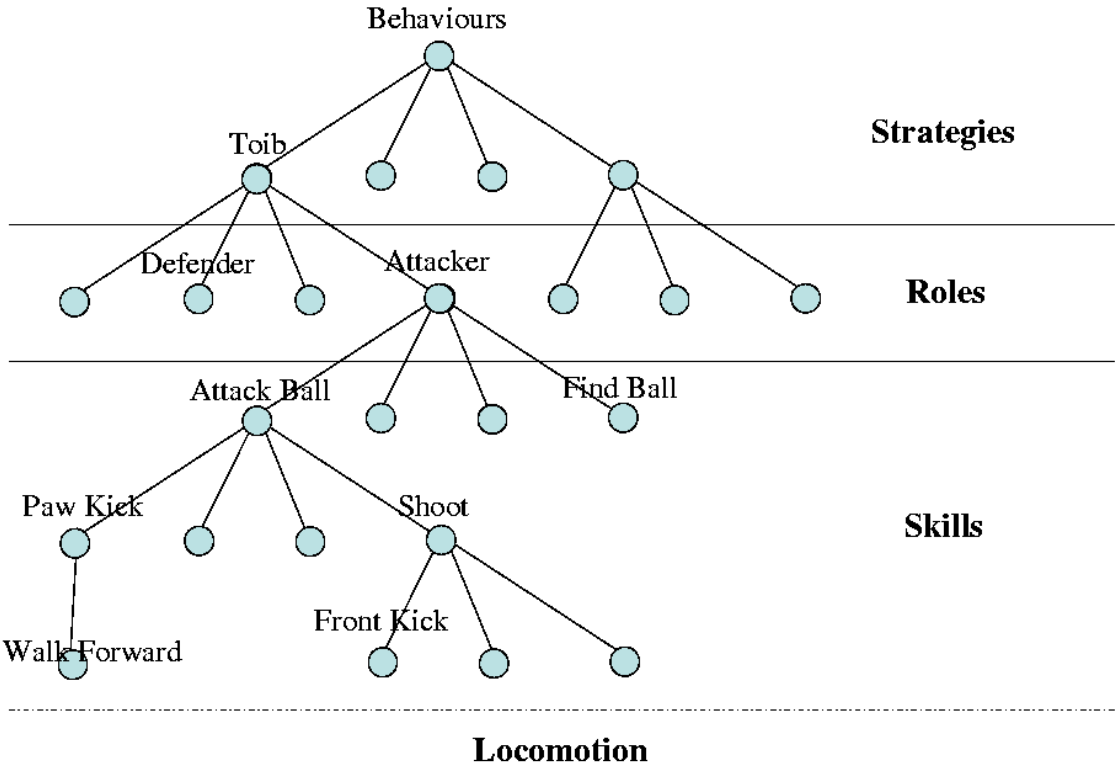


Figure 4.1: Hierarchy of the behaviours decision tree

tree gets messy as there are many skills of differing complexity, and many skills build upon other skills. Essentially many actions can be classified as a skill, including low level actions such as doing a kick. We draw a line, defining the walk and anything that consists of a simple playback of joint angles as being apart of the locomotion module. Thus you will find some kicks are explained in the locomotion module, and the other kicks which require some decision making are explained in this module.

4.2 Strategy Summary

4.2.1 Intelligent Positioning

The consolidation of behaviours into a single hierarchy accompanied a major reworking of the behaviour of the rUNSWift team. While many aspects of the aggressive design philosophy for individual players remained the same, much more emphasis was placed on team cooperation. There are 2 tiers of cooperation in our team strategy. The first works on a local level, defining close quarter interactions between the 2 robots that closely chase the ball at all times. The second operates on a global level and positions the 2 robots that work as one entity and a third robot appropriately across the field. It is important to note however that we do not equate team cooperation solely with a passing game. The robots are less concerned with explicitly performing long range passing and are more concerned with intelligent interactions and positioning which will place the robots in the best position to capture lose balls, and capitalize on any opportunities that arise.

It is our tactic to have two robots closely guarding and attacking the ball at all times, with the third robot in a wide supporting position. The two robots do a lot of shifting and reconfiguring of their positions relative to the ball and one another as they chase the ball. Collectively, this allows them to better maintain possession and control of the ball, as the robots support one another and take turns attacking the ball. Typically one robot will be attacking the ball, and the other will be closely supporting, getting ready to take over the attack. This is mostly used in overcoming the opponents' defensive moves. When the original attacking robot runs into the opponents' resistance, the supporting robot will then take over the attack.

This local cooperation approach complements one that involves passing on a global level. A typical attack scenario is one where the 2 attacking forwards bring the ball to one side of the goal in order to draw the goalie and the rest of the opponents there. The two attacking robots then try to get the ball into the open on the other side of the field, where a wide supporting robot is ready to then become the attacker and hopefully score a goal. Thus, the global positioning strategy is more about arranging the robots into positions where

they are likely to be able to intercept loose balls rather than having the robots deliberately pass the ball to one another.

The motivation behind passing less explicitly is speed. So far, we have not been able to make the robots pass and catch a ball quickly and reliably. Catching the ball was difficult because grabbing the ball is slow and factors such as the ball not rolling straight inhibited the robots from passing accurately. Speed and aggression has always been the philosophy of the rUNSWift strategy and thus the idea of a more global passing strategy was abandoned and replaced with one that involved more intricate positioning. It was anticipated however that other teams may employ this strategy. Catching the ball, and then adjusting a robots position so that it can then shoot the ball is slow. If used at the wrong time, the 2 rUNSWift attacking forwards are usually able to run up to the single robot that has caught the ball and overwhelm it before it makes the kick. The local cooperation strategy is typically able to keep the 2 attacking robots from becoming entangled amongst one another. Thus as one robot competes with the opponent robot for possession of the ball, the second robot is clear to take control of the ball as soon as it is knocked free.

rUNSWifts global positioning strategy does not adopt the conservative approach of permanently having a forward remaining back in its own half defensively but instead favours the principle that defence is best achieved through offence and that it would be undesirable to lose the resources of an active third forward. This strategy leaves rUNSWift exposed to an extremely problematic situation that arises when all three forwards are on the wrong side of the ball and opponent robots have possession of the ball. In the Robocup 2003 final against UPenn, rUNSWift found itself in such a situation several times because the UPenn forwards had a kick that successfully flicked the ball past rUNSWift's forwards.

rUNSWift developed a defensive behaviour to counter this problem: the forward with the greatest offset across the field from the ball quickly rushes to position itself behind the ball, while the two other forwards rush directly towards the ball. This defensive behaviour is dynamically triggered when all three forwards are on the wrong side of the ball. The behaviour is effective because when this situation occurs, an opponent robot is likely to be either closer to the ball than the forwards or is between the forwards and the ball. If all three forwards rush directly towards the ball, they will either become blocked by opponent robots or become entangled with one another. In order to ensure effective defence, one of the forwards needs to quickly get back to its own half, between the ball and its own goal to block a possible goal. As it is important that this robot does not become entangled or taken off the field, it circles around other robots and travels along the boundary of the goal box if it finds that its path would otherwise travel through it.

It is the combination of local cooperation skills and global cooperation skills that results in an effective team strategy. Although to the unfamiliar eye, the behaviour may not be obvious, subtle as it is, the behaviour is quite deliberate,

planned and effective.

4.2.2 Speed Accuracy and Aggression

Living up to its name, rUNSWift has again developed a faster walk. However, this faster walk is restricted to walking forward and turning at small angles. In order to maximize its effect on the game, the strategy is tuned so that it maximizes its usage of the fast walk.

One such skill that takes advantage of the new walk is the paw kick. Although the paw kick existed last year, its effectiveness has considerably increased this year with developments in the vision module, and a rethink in how it is implemented. Thus it was heavily used in this year's strategy as it allows the robot to propel the ball up the field whilst continuing to walk at full speed. This gives the robot a significant advantage over robots that first need to slow down in order to reliably grab the ball before executing a separate routine to kick it. The strategy concentrates on using the paw kick and the lightning kick to keep the ball away from the opponents and quickly move the ball upfield. Doing so stochastically increases rUNSWift's chances of scoring.

The paw kick and lightning kick are very fast, but provide only minimal directional control. Only when the robot is fairly certain that it is in the clear and that it will be able to grab and shoot the ball will it do so. Otherwise it will keep the ball moving and wait for another opportunity to grab the ball. Since these opportunities are infrequent it must capitalize on the opportunity when it arises. In order to maximize its chance of scoring a goal, a kick that visually aims the ball at gaps in the goal, thereby avoiding other robots and the goal keeper was developed. The kick is highly effective, however as one might imagine, the ball is often moved all the way to the goal mouth without an opportunity to aim and shoot. The entire strategy is aggressive, but this is where the explicit aggression comes in.

When near the goal mouth and confronted with a goalie, the robot uses an aggressive dribble to force the ball past the goalie and into the goal. The maneuver first requires the robot to grab the ball. It then turns towards the goal and walks at full speed with the ball between its front paws towards the target goal. This often results in an eventual goal.

Overall, it is competency of the individual robot combined with the intricate cooperation and positioning strategy that results in a highly effective robot soccer team. All implementation is driven by practical experience learnt though daily testing. It is the detailed fine tuning that resulted from this that led to a world champion robot soccer team.

4.3 Main Attacker Decision Tree

4.3.1 Ball information source

The behaviour of the main attacker in the current frame depends on the source from which information about ball heading and distance is obtained. There are three sources from which the main attacker can obtain ball heading and distance: vision, GPS and wireless. The attacker uses the information about ball heading and distance from one of these three, depending on which is the most desirable in the present situation. Visual information about the ball is relied upon whenever the ball is visible in the current camera frame. Information about the ball received from teammates via wireless is used when the variance of the wireless ball's position is not great and the ball has not been out of sight for too long. In all other cases, information about the ball from the GPS module is used.

4.3.2 Active localisation

A problem that rUNSWift encountered was the trade-off between keeping the main attacker's eyes on the ball and actively localising. On the one hand, the main attacker should keep its eyes on the ball as much as possible so as to minimise the chance of losing sight of the ball. However, the main attacker also needs to look at fieldmarks in order to accurately estimate its position on the field. After all, even if the main attacker has possession of the ball, but does not know its whereabouts, it will not know what to do with the ball that it currently possesses. Deciding when to take its eyes off the ball to localise from its position in relation to the beacoms is problematic.

After vigorous experimentation, rUNSWift determined the following conditions for when to actively localise:

1. The main attacker can see the ball
2. The ball is still far away and straight ahead
3. There are no opponents nearby
4. The main attacker has not recently actively localise

It is undesirable to actively localise when the ball is close. This is because after the robot localises and looks to where the ball was last seen, it will most

likely lose the ball, in that the ball will not be where it was last seen. When the ball is far away, there is a higher chance that when the robot looks to where it last saw the ball, the ball will still be there. Moreover, when opponent robots are nearby and contesting for the ball, it is more important for the forward to aggressively seek to gain possession of the ball, rather than actively localising.

When all four conditions are satisfied, active localisation is performed. However, the forward does not stay stationary during active localisation; it continues to hover towards where it last saw the ball. This ensures that it maintains its aggressive stance.

4.3.3 Main actions

When the main attacker receives visual information that the ball is in the current camera frame, it approaches it for an attack. The exact manner in which the main attacker approaches the ball is documented in the "Main attacker approach ball" section.

When the ball is not visible in the current camera frame, but wireless ball can be trusted, that is, the variance of the ball received from teammates is below a certain threshold, then the forward can do one of two possible things: move directly towards the wireless ball or perform a "locate ball" action.

There is a threshold of two standard deviations of the position of the teammate sending wireless information which determines which action the robot performs. Wireless information is acted upon where the distance between the robot and the ball is greater than this threshold. In this situation, the robot moves directly towards the wireless ball. In order to maximise the chance of catching sight of the ball, when the forward moves towards the wireless ball, its head rotates in a circular manner. The location of the wireless ball merely serves as a guide for the forward's global heading. Continuously tracking the wireless ball is ineffective in the event that the wireless ball information is incorrect.

However, if the distance between the robot and the ball is less than this threshold, then the forward does not rely on the wireless ball. Instead, it performs a locate ball action. This is very effective because the ball is very likely to be nearby, but the main attacker does not see the ball because the ball might be right next to it.

When the forward has only momentarily lost sight of the ball, that is, it has lost sight of the ball for no more than 10 frames, and wireless ball information is not to be trusted, then the forward hovers towards the GPS ball, that is, it continues to travel directly towards where it last saw the ball.

In the event that the forward has lost sight of the ball for more than 10 frames and the wireless ball information received from its teammates is untimely or the variance of the wireless ball received from its teammates is too great, the forward performs a locate ball action. Details on the locate ball behaviour can be found in the locate-ball skill section.

In all other cases the robot will use the main forward attack ball strategy to approach the ball. This strategy defines different actions to perform in order to get to the ball and is described in more detail below in the "Main attacker approach ball" section. Once the robot has grabbed the ball, it uses the main forward shooting strategy to determine which kick it should use to propel the ball down the field. This shooting strategy is described in the next section.

4.4 Main Forward Attack Ball Strategy

4.4.1 Introduction

There are several ways in which a rUNSWift main forward can approach the ball to execute its attack. The actions available for the rUNSWift forward are as follows:

- Left paw-kick, where the forward kicks the ball with its left paw;
- Right paw-kick, where the forward kicks the ball with its right paw;
- Get behind ball, which is a defensive behaviour to position the forward at a certain heading upon reaching the ball; and
- Hover to ball directly, which involves walking towards the ball with the intention of grabbing the ball for a kick

This is in contrast to the strategy employed by most teams at the Robocup 2003 competition where the main attacker moved directly towards the ball for a grab.

Details of the above behaviours are discussed in the Skills section, but in order to understand the conditions under which each of these actions are performed by a forward, certain characteristics should be discussed.

When a forward performs a left or right paw-kick, it does not slow down when approaching the ball. The forward approaches the ball at a maximum speed of 27cm/sec. However, both of these paw-kicks are not as accurate or powerful as the forward-kick, and can only move the ball forward because there is no fine-grained control over the direction in which the ball travels once kicked.

Hovering towards the ball involves slowing down once the robot gets within a certain distance from the ball. This is to ensure that the robot does not fumble the ball away, and so is able to reliably place the ball underneath its chin in order to execute a kick. The slow down ensures that the paws do not interfere with the ball. This approach is good for reliable shooting and the kicks that can be executed once the ball is successfully grabbed can be more powerful than the paw-kick.

Getting behind the ball is even slower. It takes a much longer time to approach the ball as it does not follow the shortest path to the ball. However, it ensures that by the time the robot has reached the ball, the robot is not facing its own goal, which eliminates the danger of the robot accidentally fumbles the ball towards its own goal.

The type of ball approach chosen depends not only on where the robot is on the field and where the ball is relative to the robot, but also on the position of the closest robot from the opposing team and on how confident the robot is of its position.

Hovering to the ball is considered a default case. The code is structured so that the trigger conditions for the paw-kicks and get behind ball routines are checked. If none of these checks are satisfied, the main attacker defaults to hovering to the ball for a grab.

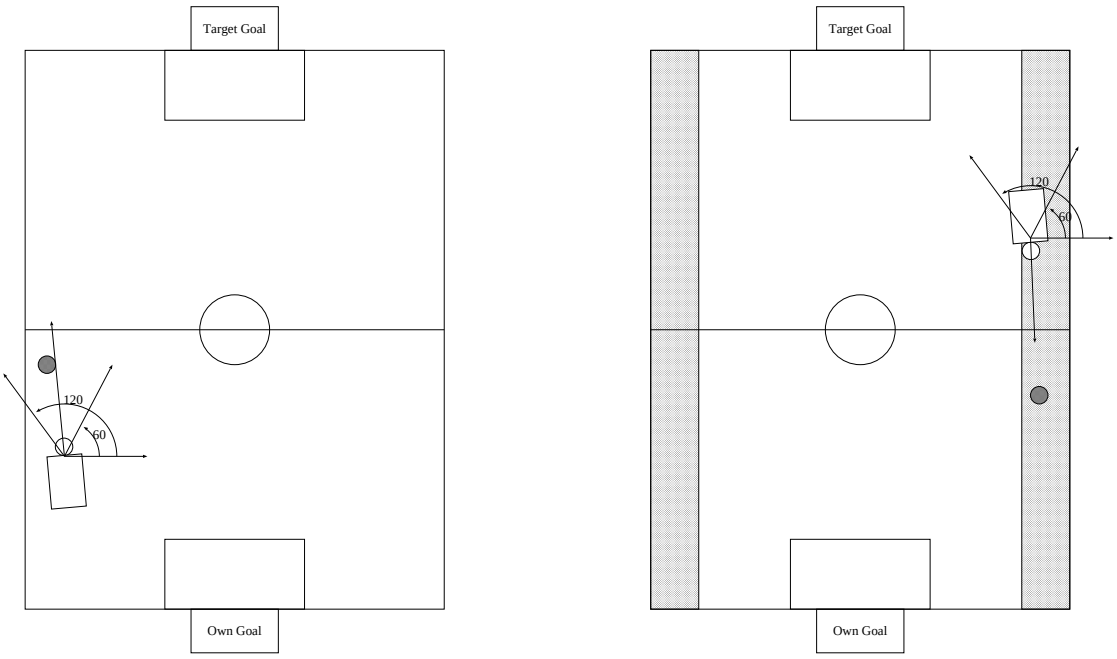
4.4.2 Paw Kick Cases

Case 1: Left and Right Edges

When the ball is on the left edge of the field and the robot is also on the left edge and facing upfield, the robot approaches the ball very rapidly for a paw-kick. Facing upfield is defined to be a state of possessing a global heading of between 60 degrees and 120 degrees as shown in Figure 4.2a. A further requirement for triggering this paw-kick is that the ball must be within 30 degrees of the robot's own heading. This condition is to ensure that the paw-kick does not actually miss the ball; balls that are very close but have a great deviation from the robot's heading can miss easily upon a paw-kick. A similar trigger condition also exists when the ball and the robot are on the right edge.

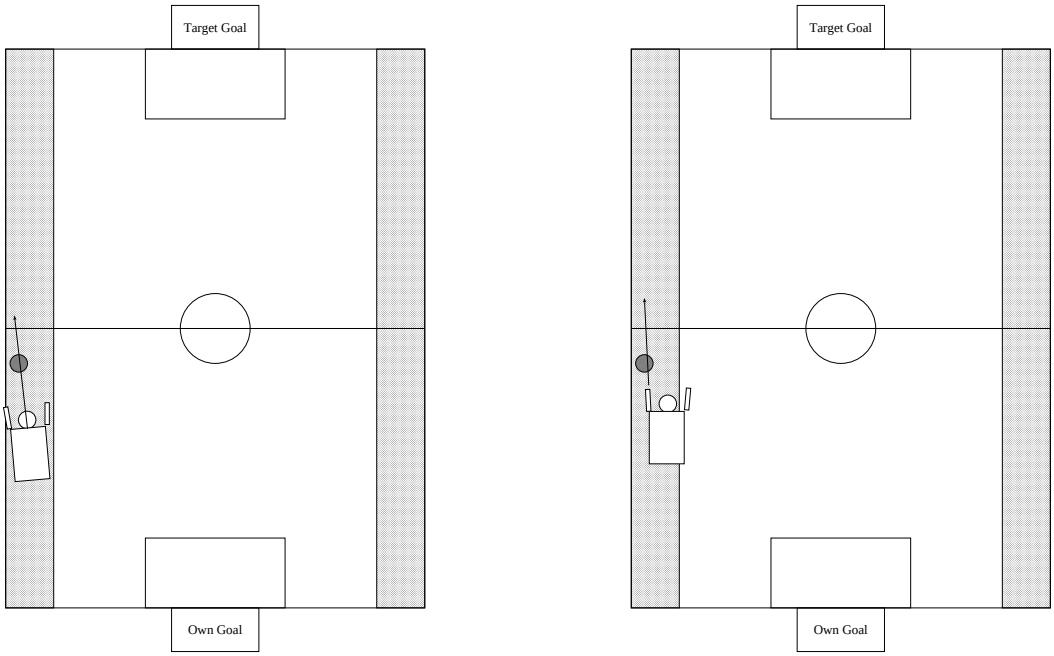
A paw-kick in such scenarios is particularly effective because the robot's priority is to get the ball upfield towards the target goal as quickly as possible. This is particularly true in midfield play when the robot needs to move the ball behind opponent forwards.

A further reason for why a paw-kick is effective is because if the robot goes directly for the ball in an attempt to position it underneath its chin, the left paw gets caught on the left edge and the robot becomes stuck, giving opponents time to position themselves defensively, respond to the attack, or to approach



(a) (b)

Figure 4.2: (a) Paw Kick Condition satisfied (b) Paw Kick Condition not satisfied



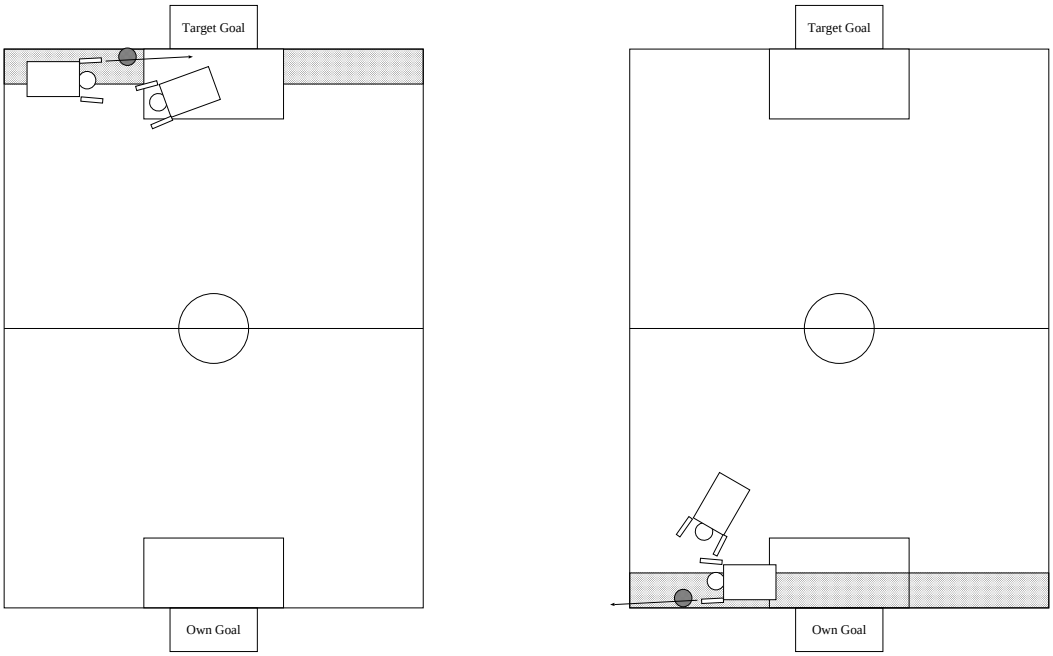
(a) (b)

Figure 4.3: (a) The robot's left paw gets caught on the left edge of the field, allowing opponent robots time to move to defensive positions to block off the attack. (b) The robot executes a left paw-kick, driving the ball upfield.

the ball and block off the attack. This situation is depicted in Figure 4.3a.

A further reason for why the paw-kick is very effective is because there is a "follow through" to the kick, which sustains the attack. After a forward executes a normal forward kick or a turn kick, the robot needs time to recover. This is time-consuming. However, with paw-kicks, because the forward does not slow down on approach and does not have to get down to kick the ball, after the kick, it is continues to move towards the ball at full speed. It is therefore able to sustain and continue the attack, without giving opponent robots time to recover.

An alternative approach that was experimented with was getting the robot to approach the ball at such an angle that by the time it reaches the ball, it is 90 degrees to the edge and hence has less difficulty in positioning the ball underneath its chin without its paw catching on the edge. However, this was found to be ineffective because it lacks the aggressiveness that is a key component of rUNSWift's game strategy.



(a) (b)

Figure 4.4: (a) Paw kick on top left edge gets ball past opponent goalie. (b) Paw kick on bottom left edge gets ball away from own goal.

Case 2: Top Edge

When the robot is on the top edge of the field, its paws can get caught on the top edge. This is the same problem that it faces when it is on the side edges. In the situation depicted in Figure 4.4a, the most desirable action is to perform a paw-kick, which is both aggressive and quick.

It has also been found that this approach is particularly effective in moving the ball past the goalie.

The exact conditions for when this is triggered are as follows:

- The ball is in the top left edge;
- The robot's global heading is between 0 degrees and 30 degrees;
- The ball is to the left of the left goalbox edge; and
- The robot is also positioned to the left of the left goalbox edge.
- The nearest opponent is close by.

When it is apparent that the target goal is clear, that is, the nearest opponent is far away, the paw-kick is not triggered because of the unpredictability of the direction in which the ball travels after the kick. Since the robot is at the target goal with the ball, it needs to select an action that is sufficiently accurate and reliable in order to score. Because no opponents are nearby, the robot should take more time to grab the ball for a reliable kick.

A similar trigger condition is present for when both the ball and the robot are at the top edge as shown in Figure 4.4a.

Case 3: Bottom Edge

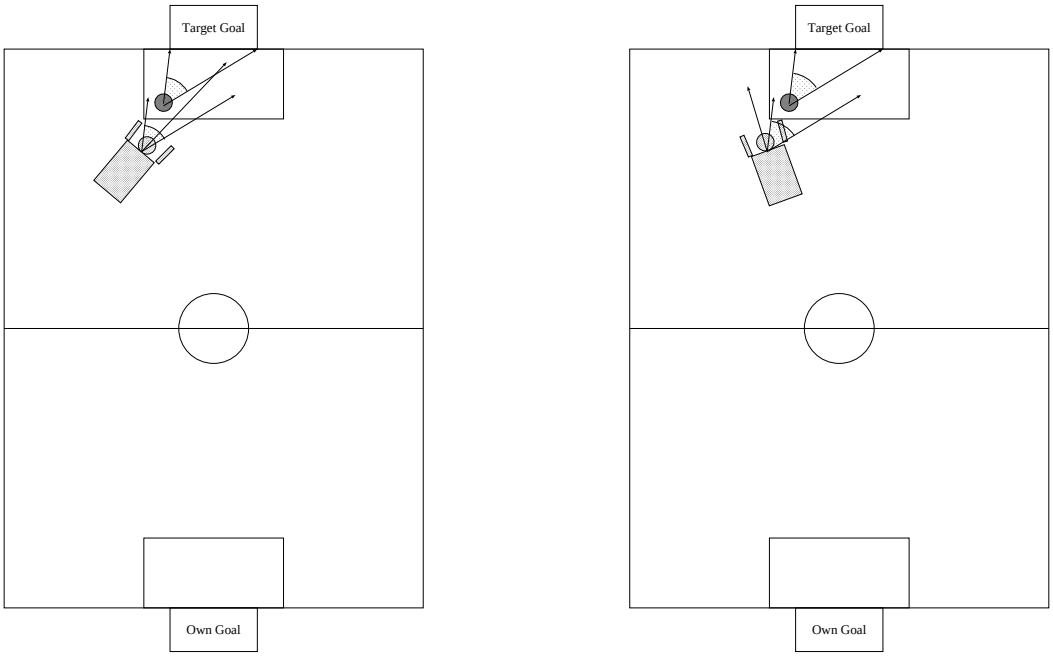
There is a difficult defence scenario when the ball is in the bottom left edge, as shown in Figure 4.4b.

If the robot is also positioned in the bottom left corner between the ball and the its own goal, a paw-kick is performed because the robot should act aggressively to clear the ball from the goal as quickly as possible, and because the robot might be contesting for the ball with opponents. The strategy applies when the ball is on the bottom right edge.

Case 4: For Goal

Cases 4, 5 and 6 involve using the paw-kick to get the ball past an opponent robot as quickly as possible.

In the case where the ball is inside the target goalbox, a goalie is inside the target goal and the robot is facing the target goal with the ball lined up



(a) (b)

Figure 4.5: (a) Paw-kicking for goal. (b) Paw-kicking in this scenario will miss. Hence the robot will go for a grab instead.

in between, the robot attempts to paw-kick the ball into the goal rather than slowing down to reliably grab the ball for a front kick. At this point, speed is essential because any slowdown might allow the goalie to move into a good defensive position to block off the attack. This scenario is shown in Figure 4.5a.

The robot first calculates the heading from the ball to the two target goal posts. If the robot's own heading falls within these values then a paw-kick would theoretically result in a score.

The robot then checks whether the ball is close enough to it, and also whether the heading to the ball relative to the robot is not too significant. This is to ensure that the paw-kick will successfully connect with the ball. In Figure 4.6, even though the robot's global heading lies within the two goal post headings, a paw-kick will not successfully connect because the ball is too far to the robot's left. An attempt to paw-kick in this situation will either miss the ball completely or lightly tap the ball to the left. Both of these results are highly undesirable because the ball would now lie behind the robot and the robot would have lost

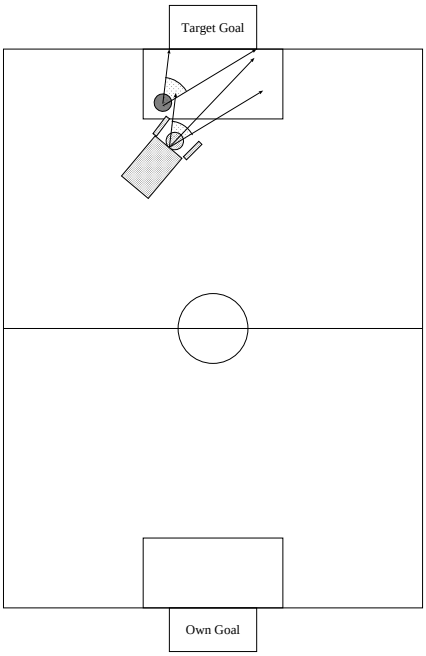


Figure 4.6: A paw-kick miss occurs when the heading to the ball is too significant.

sight of the ball.

Before triggering the paw-kick, it is important to ensure that opponent robots are nearby. If no opponent are nearby, the robot has the time to grab the ball to reliably forward kick it into the goal. In determining whether opponents are nearby, the robot is more sensitive to opponents that are positioned inside the goal box, than opponents outside. This is because an opponent that is inside the goalbox is assumed to be the goalie.

Case 5: Opponent Near Ball

This case operates on the principle that the robot needs to act more aggressively, the closer opponent robots are to the ball. The distance between each GPS-detected opponent from the ball is calculated; if the minimum distance is less than a certain threshold, the robot then checks to see if it is facing upfield. If its heading to the goal is such that kicking the ball would not move it towards

the goal, the paw-kick is not performed.

Case 6: Charge Condition

The last condition is similar to the condition described in Case 5, except it relies on visual opponent detection alone rather than opponent tracking information from the GPS module. If the robot sees the ball in the current camera frame and it has seen an opponent robot within the last 12 camera frames, and the robot is facing upfield, then the paw-kick is selected.

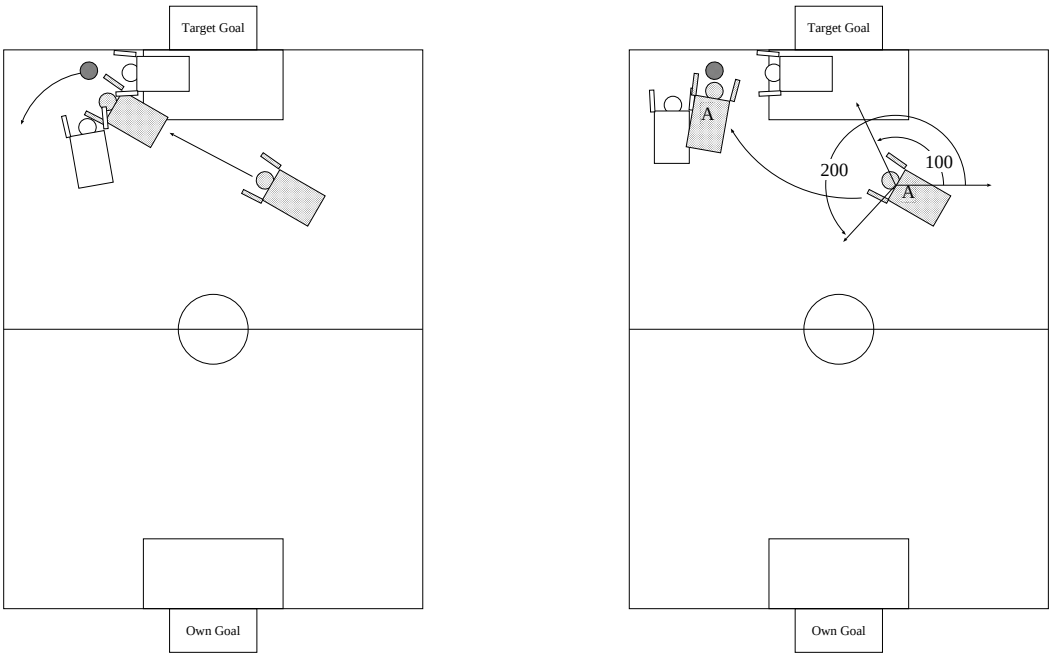
4.4.3 Get Behind Ball Cases

Case 1: Top Corners

When the ball is in the left target corner and the robot is facing the corner with a global heading that is greater than 90 degrees, as depicted in Figure 4.7a, going directly towards the ball for a ball grab to execute a kick is undesirable. Performing a paw-kick is also clearly undesirable, as it will result in the ball moving further away from the goal.

Going directly for the ball is ineffective in a game scenario because by the time the robot has reached the ball, its heading is unsuitable for an attack as it is facing away from the target goal. Furthermore, taking into account the fact that scrums occur very often in corners and that opponent robots will defend their goal vigorously, by the time the robot has reached the ball, it is likely to be wedged between opponent robots that are also contesting for the ball.

A typical scenario is illustrated in the diagram shown. When this occurs, it is very likely for the ball to emerge on the wrong side of the scrum, being down the side edges. This is why when the robot detects that its global heading lies between 110 degrees and 210 degrees and the ball is in the top left corner, it moves behind the ball so that by the time it reaches the ball, its heading is more suited to attacking the target goal. This is depicted in Figure 4.7b. Clearly, getting behind the ball is more time consuming than going directly towards the ball, which is why the former action is only activated once the forward is a certain distance from the ball.



(a) (b)

Figure 4.7: (a) Going directly towards the ball is undesirable, especially when opponent robots are nearby. (b) Instead of hovering directly towards the ball, the robot performs a get behind ball so that by the time it reaches the ball, it's global heading is more favourable.

A similar case applies to the top right corner.

Case 2: Bottom Edge

There is a similar condition for getting behind the ball as described in Case 1, that is, in a scenario where the ball is in the top corners and the robot needs to face the right way for an attack. When the ball is in the bottom edge, the robot moves to get behind the ball in order to position itself defensively.

Figure 4.8a shows such a scenario, where the ball is located in the bottom edge and the robot's global heading lies between 200 degrees and 340 degrees, that is, the robot is facing downfield and towards the ball. Instead of moving directly towards the ball, the forward moves behind the ball, thus ensuringn that when it reaches the ball, it has also positioned itself between the ball and its own goal.

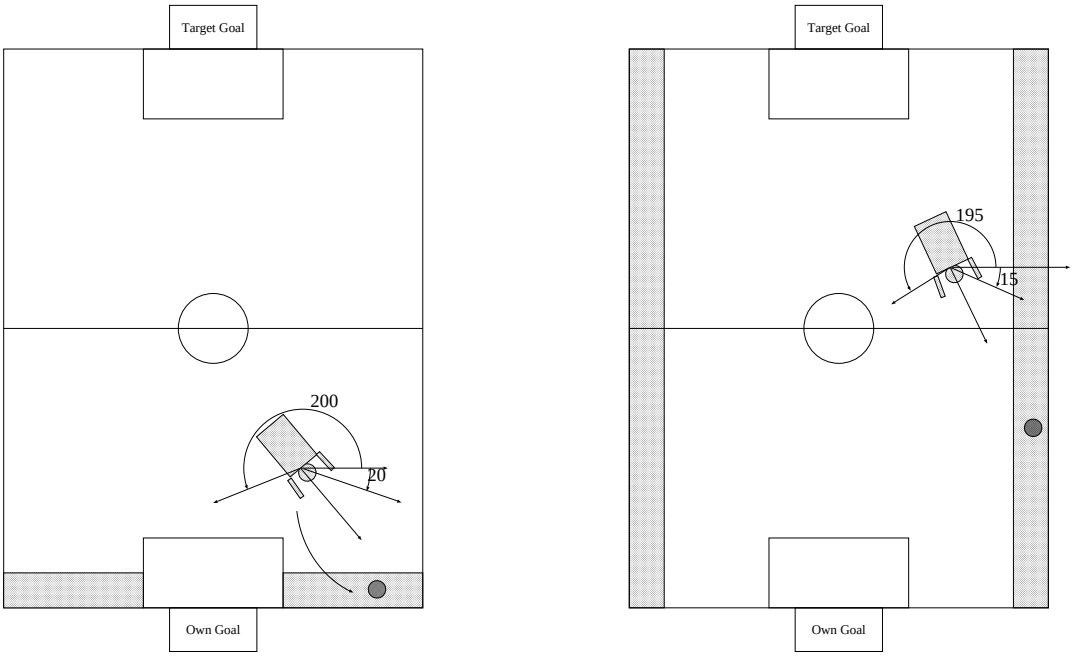
This manoeuvre also effectively adds an additional layer of defence because when opponent robots are nearby, the forward is positioned to prevent them from pushing the ball past the goalie.

An emergent property of this behaviour is that nearby opponent robots that are also trying to reach the ball are pushed away from the ball by the forward. This results in the rUNSWift forward being a step ahead in reaching the ball, as shown in Figure 4.8a.

Case 3: Side Edges

Furthermore, when the ball is on either of the side edges, and the robot is facing downfield with a global heading between 195 degrees and 345 degrees, the robot gets behind the ball to approach the ball at a more favourable angle with the edge. This situation is shown in Figure 4.8b.

This is especially important because scrums frequently occur on the sides and when opponent robots are also contesting for the ball, a forward that moves directly towards the ball can become wedged between opponent robots. The forward is then facing the wrong direction and unable to move freely. Not only is



(a) (b)

Figure 4.8: (a) Get behind ball performed when the ball is in the bottom edge region. (b) Get behind ball performed when the ball is in the side edge region.

the forward clearly in no position to move the ball upfield, it is also blocking the ball from being moved upfield by teammates that are in a better position to attack the ball.

A further reason for why attacking the ball directly is ineffective when the forward is facing downfield and opponent robots are also contesting for the ball is that controlling the ball reliably becomes impossible. Because the forward is facing towards its own goal, an attempt to grab the ball to execute a 180 degree turn-kick is likely to fumble the ball towards its own goal.

Case 4: Facing Own Goal

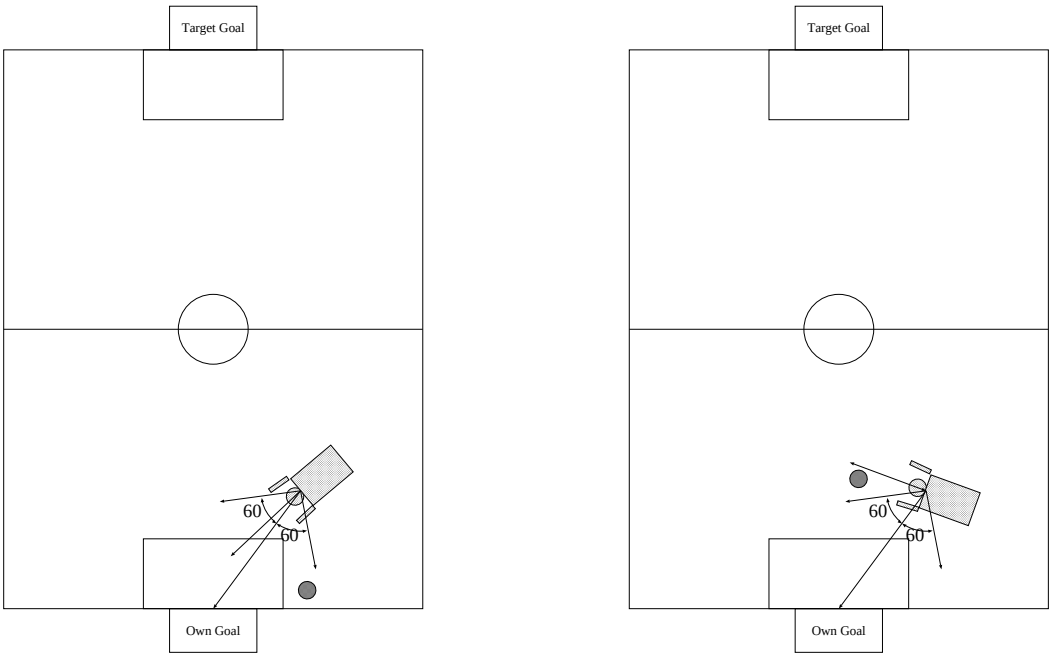
The final condition for a forward to move behind the ball is when the ball is directly in front of and dangerously close to the forward's own goal If the forward moves directly towards the ball to grab it, the forward can inadvertently knock the ball into its own goal.

The forward knows that it is facing its own goal if it possesses a global heading that lies within a certain offset from its heading to its own goal. When it has moved behind the ball enough that its global heading lies outside these limits, the forward hovers directly towards the ball as Figure 4.9b depicts.

The desired attacking angle, that is, the desired global heading of the forward upon reaching the ball, is equal to the heading to the ball's from the robot's own goal. This ensures that the robot places itself between the ball and its goal.

More detailed information concerning the get behind ball manouevre, including how the decision is made as to whether the forward should circle to the left or the right of the ball, can be found in the Skills Section of this report.

The conditions under which the forward moves behind the ball are determined very judiciously. Although moving behind the ball is an extremely effective defensive behaviour, the forward takes time to move into the correct position. Therefore, in the strategy for the main attacker, the forward attacks the ball directly for a grab when the ball is in midfield and not along the side edges, as shown in Figure 4.10. A forward that moves behind the ball too often allows the opposition time to close in.



(a) (b)

Figure 4.9: (a) Get behind ball is performed when the robot is facing its own goal and doesn't want to directly approach the ball for fear of fumbling the ball into its own goal. (b) This diagram shows the condition failing, hence the robot will hover directly towards the ball.

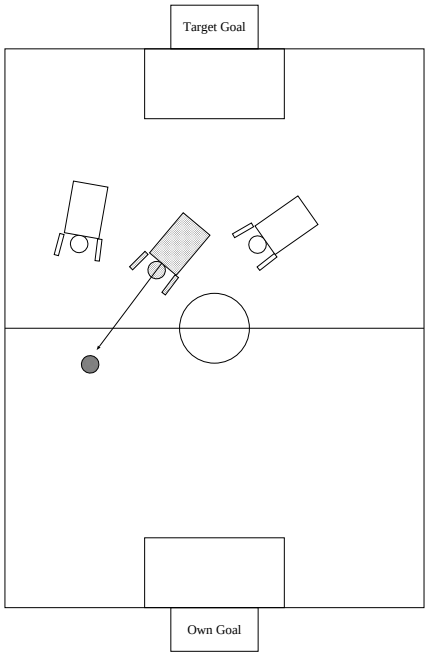


Figure 4.10: The forward attacks the ball directly instead of moving behind the ball during midfield play.

4.4.4 Hover to Ball

The main attacker hovers towards the ball when none of the paw-kick and get behind the ball conditions are satisfied. The hover to ball skill directs the robot on what it believes is the shortest path to the ball. In most scenarios, this involves a direct path to the ball, but under certain conditions, the robot uses the balls velocity to anticipate a point where it can intercept the ball. The robot also tries to avoid walking into teammates and opponents as it walks towards the ball in order to grab it. Once it has grabbed the ball, ie has it under its chin, it then executes a strategy which decides which kick it should use. The section on the main attacker shooting strategy describes the decision tree that selects which kick action to execute.

The level of noise in the distance and heading estimates to the ball obtained from the vision module increases when the ball is close to the robot. This is because the robots head bounces around as it walks to the ball. Changes in the height of the head change the perceived location of the ball. In an attempt to minimise adverse effects caused by noisy data, a weighted averge is taken for these values over the last three frames. This value is then passed to the hover to ball routine.

Further details on the hover to ball behaviour are documented in the Skills Section of the report.

4.5 Main Forward Shooting Strategy

4.5.1 Introduction

The main forward shooting strategy centres around determining the most desirable action upon obtaining possession of the ball. Ensuring the correct action is the key to an effective forward because the main forward must ensure that it exploits the advantage of having gained possession of the ball after arduously contesting for the ball.

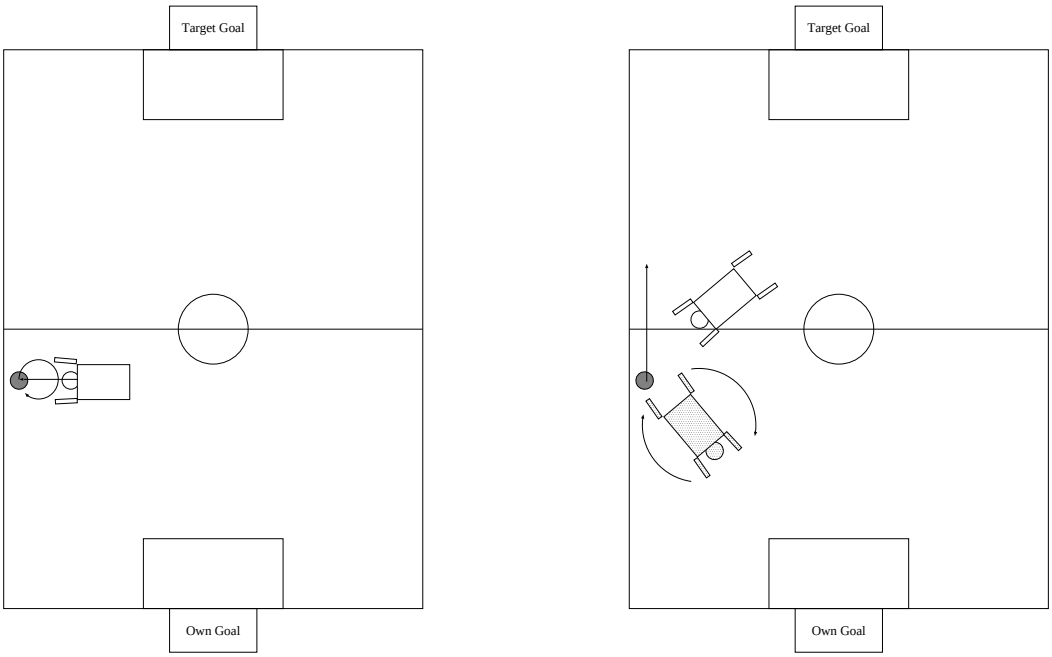
After the robot has successfully grabbed the ball, the robot must ensure that the most appropriate kick is performed given the current state of the playing field. The selection of kicks at the main forward's disposal are:

- locateball kick;
- turn kick;
- lightning kick;
- voak and;
- dribble

4.5.2 Side regions

When the robot and the ball are on the side regions as shown in Figure 4.11a, the robot performs a locateball kick action which involves turning with the ball towards the target goalbox. The locate ball kick is similar to the turn kick, however after performing the kick, the robot follows through my spinning into the locate ball routine. If the kick was successful, the robot will most likely find the ball immediately, and thus stop spinning. Otherwise the robot will keep spinning until it finds the ball.

This kick is most effective when opponent robots are nearby. After the ball has been kicked, even if an opponent robot has successfully blocked it, the fact that the main forward continues to spin creates a high chance that the backpaws kick the ball in the right direction, i.e. upfield, and often up onto the triangluar side wall and around the opponent robot.



(a) (b)

Figure 4.11: (a) The robot performs a locate-ball kick on the side regions. (b) The robot's back-paw kicks the ball upfield behind opponent robot.

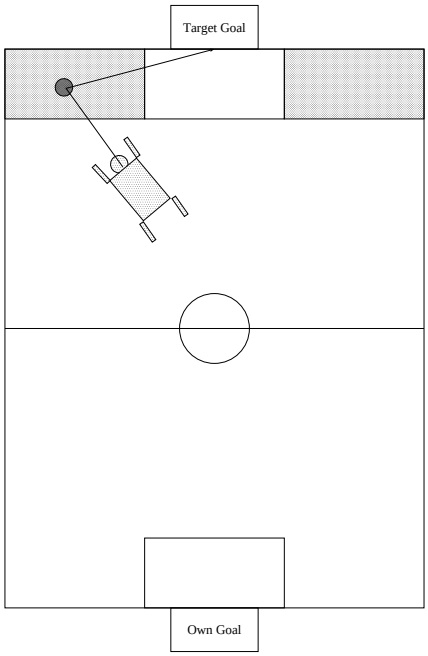


Figure 4.12: Spin dribble is performed whenever the robot has possession of the ball in the shaded regions.

4.5.3 Top corners

When the robot is in the top corners as shown in Figure 4.12, the robot performs a spin dribble. This involves actively localising while the ball is still within its grasp, and spinning until it is heading directly towards the target goal. It then dribbles the ball into the goal.

Actively localising allows the robot to obtain a more accurates estimate of its global heading in this situation. This is important as the robot precalculates the angle at which it needs to spin to face the target goal. This cannot be vision-based because upon spinning, the robot must keep its head down to ensure that the ball remains in place, and therefore cannot see any fieldmarks.

The regions indicated in Figure 4.12 are also regions where scrums are most likely to occur, especially when the opponent team has a strong goalie. During scrums, the robot might need to contest for the ball for long periods of time and might therefore have a very inaccurate estimate of its own heading and location.

When the forward is prevented from freely turning due to contact with opponent robots, such as being leg-locked, information from odometry can distort the robot's belief of its global heading.

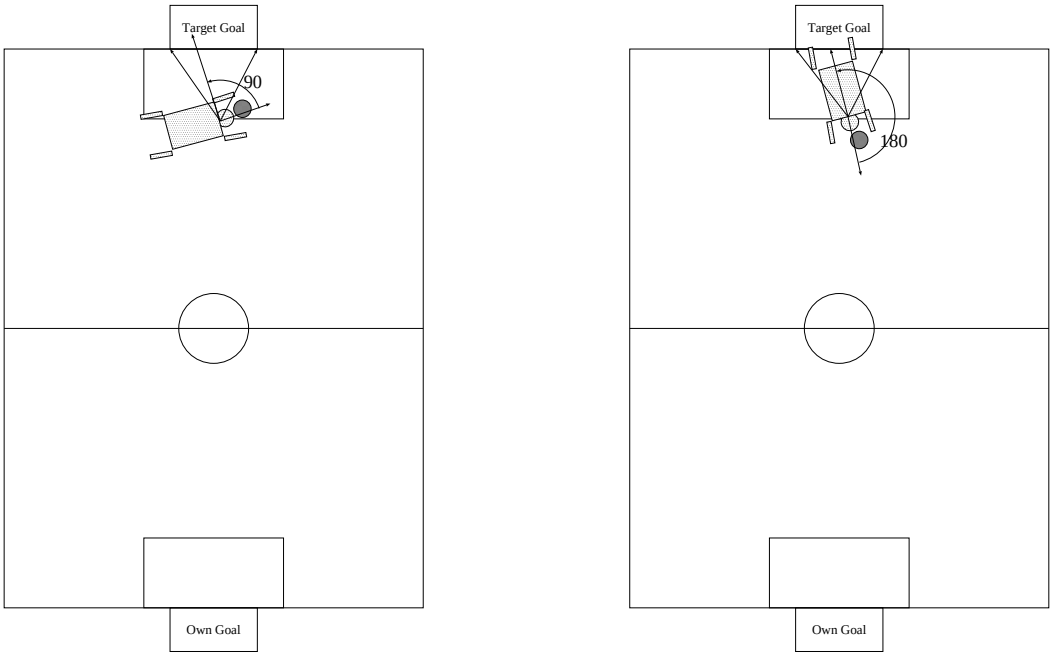
The spin-dribble is preferable to the turn-kick in this scenario because spin-dribble has a finer degree of control over the direction in which the ball travels when kicked. Careful and accurate control over the ball is required the closer the robot is to the target goal. Moreover, a dribble allows the robot to maintain possession of the ball and is faster and more effective than the chest pushes that were used by previous rUNSWift teams.

In contrast to the spin-dribble, turn-kicks only release the ball at two angles: 90 and 180 degrees from the robot's heading. This limited range at which the ball can be kicked means that the turn-kick is unable to deal with a situation where the heading to the target goal is not at either of the two angles. This means that a turnkick in the top region might disastrously result in the ball moving downfield so that all the effort spent in moving the ball upfield behind opposition is then wasted.

4.5.4 Target goalbox

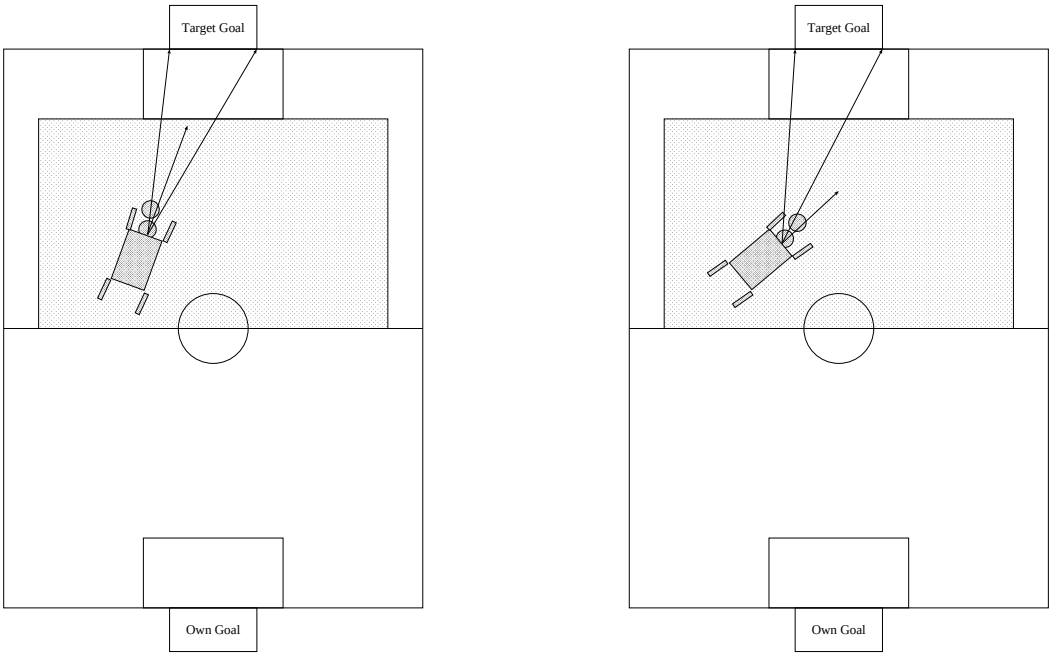
However, when the robot has possession of the ball inside the target goalbox, as shown in Figure 4.13a, the robot has the option of performing 90 and 180 degree turnkicks to score. Once the main attacker has reached the target goalbox, it is no longer in danger of being pushed inside a scrum by opponent forwards as they are removed from their own goalbox. Because the forward now only has to contest with the opponent goalie, the danger of executing a turn-kick in the wrong direction is ameliorated.

First, the heading to the goal posts are calculated. If an angular offset of 90 degrees anticlockwise from the robot's global heading lies within its heading to the goalposts, then a 90 degree left turnkick is selected. The same checks are applied for the right 90 degree and 180 degree turnkicks with offsets of 90 degrees clockwise and 180 degrees respectively.



(a) (b)

Figure 4.13: (a) The condition for a 90 degree turn-kick to the left is satisfied. (b) The condition for a 180 degree turn-kick is satisfied.



(a) (b)

Figure 4.14: (a) The global heading of the robot lies within the heading to the two target goal posts. (b) The global heading of the robot does not lie within the heading to the two target goal posts.

4.5.5 Target half

When the robot is in the target half and it is already lined up with the goal as shown in Figure 4.14a, where the robot's global heading lies between the heading to the two goal posts, the robot performs either a lightning kick or a visual opponent avoidance kick. Because the visual opponent avoidance kick takes a significantly longer time to execute than the lightning kick, it is only selected if the closest opponent is a certain distance from the robot. The advantage of the visual opponent avoidance kick over the lightning kick is that the former provides a greater amount of accuracy in scoring as it is able to aim in the direction of where it sees the target goal. This is done by taking its chin from the ball to look up and aiming for gaps in the goal, thereby avoiding the goalie.

However, when the robot detects that it is not in the clear, it chooses the lightning kick, the faster of the two kicks as it involves significantly less setup

time.

In the event that the global heading of the robot does not lie between the heading to the two goal posts, as depicted in Figure 4.14b, the robot first checks to see if it should perform a visual opponent avoidance kick. Since a time limit of 3 seconds is imposed on how long the ball can be held, visual opponent avoidance kick is only executed if the heading towards the target goal is not too significant. As mentioned above, because the visual opponent avoidance kick has a long setup time, it is not executed when the nearest opponent is close enough to shut the robot down. Hence, when opponents are nearby, a lightning kick is performed instead.

Similarly to the case when the robot is inside the goalbox, when an angular offset of 90 degrees to the robot's own heading either clockwise and anticlockwise lies within the robot's heading to the two goalposts, a left turn kick or a right turnkick will be executed for clockwise and anticlockwise offsets respectively.

When an angular offset of 180 degrees is applied to the robot's global heading and that vector lies between the two goalposts, a 180 degree turnkick is chosen. The default direction in which to turn is usually the one that will take the robot less time to face the target goal. However, in the case where an opponent is detected to be closeby, the robot turns away from the opponent in an effort to move the ball upfield successfully, by moving the ball away from the opponent.

When neither a 90 degree turn kick nor a 180 degree turn kick is estimated to be able to kick the ball between the two posts, the heading to the target goal centre is used as a guide. A goal heading between -135 and 135 degrees will trigger a 90 degree turn kick; a goal heading greater than or equal to 135, or less than or equal to -135 degrees, will trigger a 180 degrees turn kick.

4.5.6 Defensive half

The defensive scenario when the ball lies in the robot's own half is very similar to the general attacking case outlined above. However, because the robot's priority is to move the ball upfield rather than to score reliably and accurately, instead of basing kicking decisions on its heading to the goal posts, the robot considers its own heading to two imaginary posts located a field-width distance away from the centre of the target goal. For example, in Figure 4.15, because the robot's global heading lies within the heading to the two imaginary goalposts, the robot will execute a 90 degrees turnkick to the left.

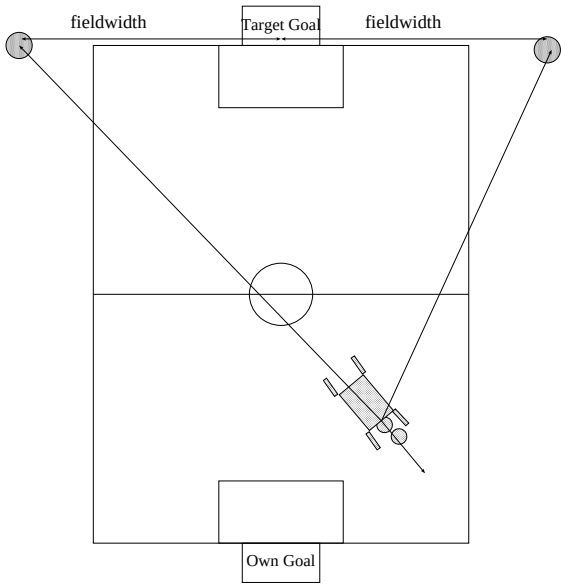


Figure 4.15: The forward's heading relative to its heading to two imaginary goal posts is considered when the ball is located within its own half.

Moreover, because the forward is in the defensive half of the field, the target goal is too far for it to attempt scoring. Hence the visual opponent avoidance kick is unused; only lightning kick is used.

In deciding the direction to turn during an execution of a 180 degree turn-kick, the robot first considers if it is in the bottom most defensive quarter. If it is, then it always turns from its own goal. This is the safest direction and avoids the possibility of scoring an own goal. If it is in the second quarter, the default direction is away from its own goal, but when opponents are near, the robot away turns from the closest opponent instead.

4.6 Global Robot Interactions

4.6.1 Introduction

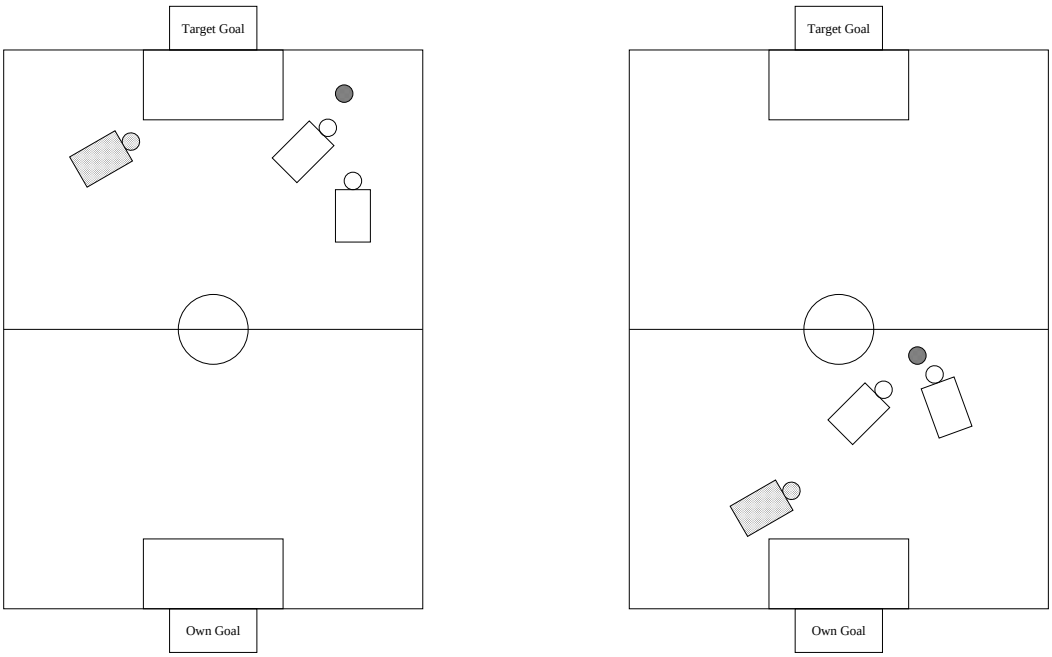
There are two forms of team cooperation in rUNSWift's strategy: global cooperation and local cooperation. In the global view of the team strategy, the two forwards that chase the ball are considered one entity. This entity and the third robot space themselves out to best cover the field. Typically the third robot stays on the oppose side of the field, away from the 2 attacking forwards, and provides long distance support to these forwards. Whether a forward assumes the role of a long distance supporter is dynamically determined by several factors; first, its position on the field relative to the ball; and second, on information it receives from teammates via wireless Ethernet. Within the entity described above, that is the between the 2 forwards that chase the ball, the local coordination strategy applies which defines interactions between them. In short, it describes which of the 2 robots should attack the ball and which should move into a close supporting position. The local coordination strategy is further described in the next section.

4.6.2 Rationale

The rationale for this global team strategy is to ensure that at any point in time, two robots maintain possession of the ball while a third is positioned to either aggressively support an attack or remain conservatively behind the ball. In a typical attack scenario as shown in Figure 4.16a, the long-distance supporter positions itself on the other side of the field from the ball, slightly behind the ball in the y-direction, close to the target goal box. The objective of the long distance supporter in this situation is to get into a position where it is able to receive a cross from its teammate and score behind the opponent goalie. However, when it senses that the ball is located in its own half, such as in Figure 4.16b, the long-distance supporter needs to position itself much further behind the ball defensively.

4.6.3 Long-distance supporter role determination

This global team strategy is heavily reliant on the wireless communication system to determine which robot is best suited to assume the role of the long-distance supporter. Each robot estimates its own distance to where it calculates the ball is located, and broadcasts this distance, together with its actual



(a) (b)

Figure 4.16: (a) Typical attack scenario. (b) Defensive positioning.

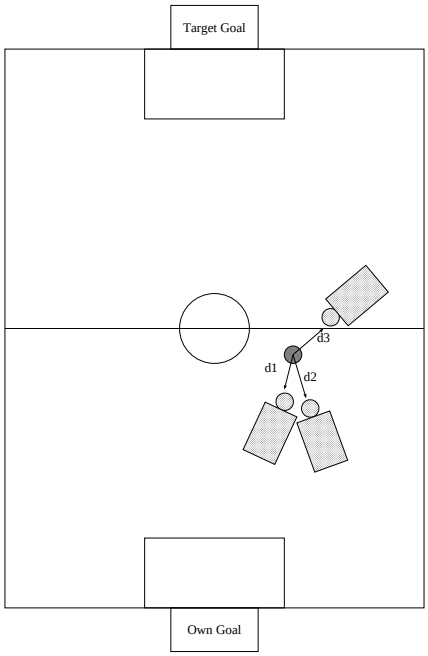
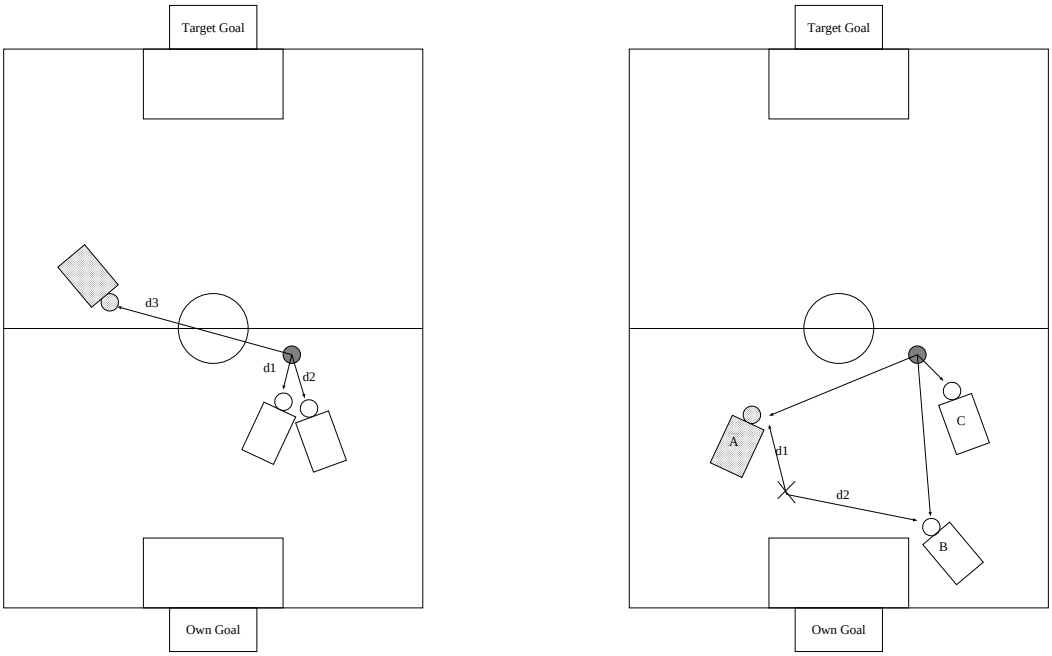


Figure 4.17: Dynamic role oscillation between all three forwards.

field coordinates, to its teammates. However, it is problematic to assign the role of the long-distance supporter to a forward based solely on which forward is furthest away from the ball. The strength of dynamic role determination lies in its flexibility to adapt to a changing environment, but it is possible for oscillations in role distributions to occur. The robots' actual distances to the ball are often quite close in a game and, because of noise in the vision system, not only can the wrong robot be chosen to be the long-distance supporter, but the role assignment can fluctuate between closely located robots, as depicted in Figure 4.17. This causes robots to be caught in a stagnant situation where they are undecided as to whether they should chase the ball or provide long-distance support.

This situation can be avoided by more carefully managing the allocation of the role of long-distance supporter to robots. The objective is to assign this role to the robot that is clearly furthest away from the ball, that is, it is further away from the ball than any other teammate by a fixed offset that is sufficiently large to remove the effect of noise in vision, as shown in Figure 4.18a.

There are three situations that require attention when assigning this role.



(a) (b)

Figure 4.18: (a) The robot is clearly furthest away from the ball. (b) The long-distance supporter destination position is used as determining factor.

A simple case is where a robot is clearly furthest away from the ball; it is then assigned the role of long-distance supporter. Another simple case is where the robot is clearly not furthest away from the ball, that is, there exists a teammate that is clearly furthest away from the ball.

The third case is one where there is some uncertainty as to whether a robot is neither clearly furthest away, nor *not* clearly furthest away from the ball. This senario is depicted in Figure 4.18b. In this situation, the position where the long-distance supporter is to place itself serves to solve the uncertainty of which robot is assigned this role. Of the three forwards in the rUNSWift team, the two forwards that are furthest away from the ball, ie. robots A and B, are selected from comparison. The robot that is closer to the desired position, i.e. robot A, is assigned the role of long-distance supporter. The newly-assigned long-distance supporter then broadcasts to its teammates its new role.

This technique turns out to be very effective not only in resolving role assignment oscillations between robots that are equidistant from the ball, but also

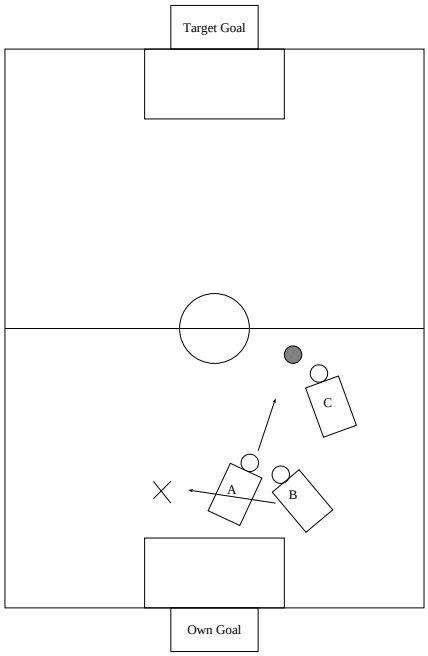
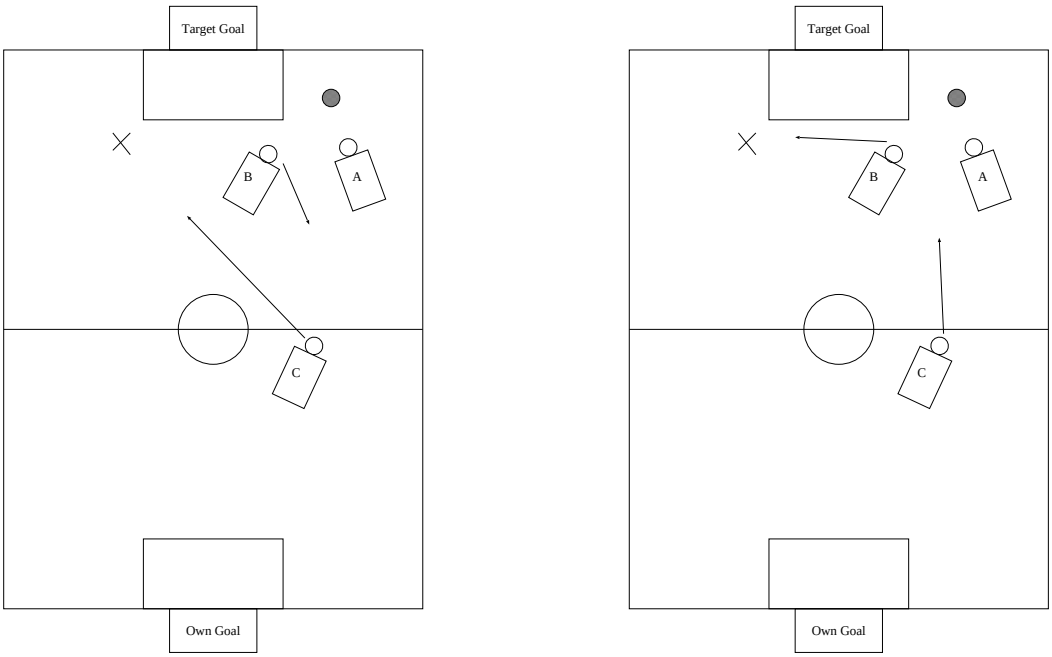


Figure 4.19: Leg-lock becomes common when the wrong forward is chosen to move to the long-distance support desired position.

in ensuring that robots arrive at strategic positions quickly. Moreover, because the destination of the long-distance support is taken into account when robots are close to one another, the chance of robots getting into one another's way and becoming leg-locked is minimised.

An exception to this rule in assigning the role of long-distance supporter occurs when the ball is close to the target goal. In such a scenario, the long-distance supporter positions itself very aggressively near the target goal, on the other side of the field from where the ball is located. This position is shown as point X in Figure 4.20a. When the above algorithm is applied to determine which robot becomes the long-distance supporter, robot C, being clearly the furthest robot from the ball, is chosen. However, robot B is in a much better position to move to point X, and robot C, coming up from behind, is in a much better position to move upfield to provide close-in support to robot A. Therefore, instead of choosing the robot furthest away from the ball, the robot that is closest to the long-distance support position, robot B in Figure 4.20b, is assigned this role.



(a) (b)

Figure 4.20: (a) Incorrect role assignment. (b) Correct role assignment.

In an attempt to ensure that there is only one long-distance supporter assigned and that continuous role switching does not occur, the long-distance supporter signals to its teammates, via the wireless communication system, that it has been assigned this role. However, because of delays in wireless communication, it is possible for two robots to both momentarily believe that they have been assigned the long-distance support role and therefore broadcast this role. In order to avoid this, each robot, upon receiving a long-distance support role signal from a teammate, compares its own player number, a number that is used to uniquely identify each robot, with that of its teammate. The robot with the higher player number remains the long-distance support.

Delays in wireless communication in robotic soccer are undesirable in this dynamic environment, where the position of the ball rarely remains static. Such delays can result in robots receiving information from teammates that is untimely and obsolete, and thus, not to be trusted. In order to minimise the impact of delays in wireless communication, if the most recent signal received from a teammate is older than 20 camera frames, the information in this signal is ignored during role determination. In the case where a robot has no timely information concerning its teammates' location (i.e. information received in the last 20 camera frames) or only has timely information about one teammate, then the robot does not become the long-distance supporter, but instead chases after the ball.

4.6.4 Long-distance support positioning

The most effective placement for the long distance support player was determined by implementing different positioning techniques and playing them against each other. These positioning methods were vantage point positioning, defensive positioning, hybrid positioning and X positioning, and are discussed in the following sections.

Vantage point positioning

In vantage point positioning, the forward moves to one of four possible static positions, A, B, C and D, as shown in Figure 4.21. Points A and B are vantage positions from which to attack the target goal, whereas points C and D are standpoints from which to clear the ball.

When the player senses that the ball is in the top right quadrant of the field,

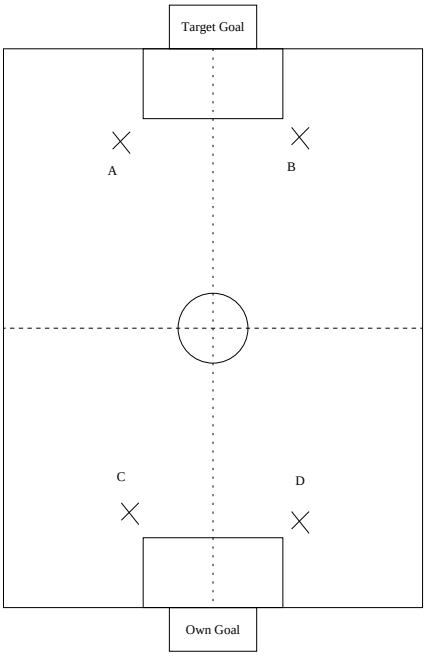


Figure 4.21: Vantage point positioning.

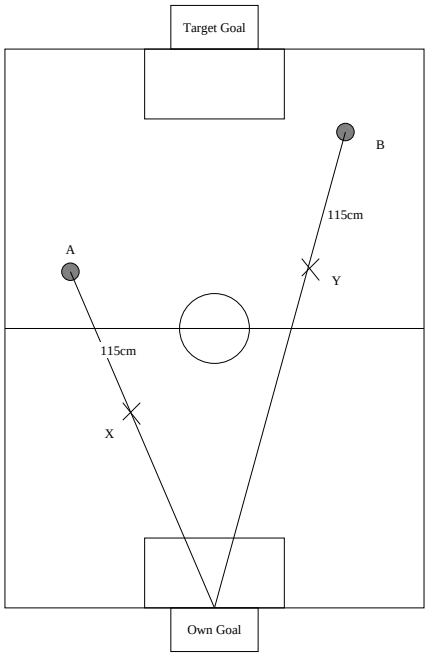


Figure 4.22: Defensive positioning.

it moves to point A, located on the far side of the field from where the ball is located, in preparation for an assault on the target goal. Similarly, the player moves to point B when the ball is located in the top right quadrant.

In a defensive scenario, when the ball is located in its own half of the field, the long distance supporter moves to point C if the ball is in the bottom right quadrant or point D if the ball is in the bottom right quadrant.

Defensive positioning

In defensive positioning, the long distance supporter positions itself conservatively, directly between the ball and its own goal. As its teammates chase the ball, it waits defensively behind for the ball to come to it, blocking attempts from the opposition to score.

As Figure 4.22 shows, a line is drawn from the ball to the middle of the supporter's goal. The supporter's destination is a point on this line 155cm behind the ball. If the ball is located dangerously close to the defending goal, the calculated position can lie inside the goal-box or outside the field. When such a scenario arises, the supporter relies on the two forwards chasing the ball and the goalie to defend the goal, and moves to the clearing vantage point that lies on the far side of the field from the ball.

Hybrid positioning

A problem with the defensive positioning method described above is that every time the ball is located close to the target goal, the long distance supporter becomes idle while its two teammates struggle to get past the opponent's defence. In other words, it behaves too conservatively when the chance of the opposition scoring is small. The hybrid technique modifies defensive positioning to make the supporter more aggressive by taking advantage of the attack vantage positions in Figure 4.21.

The field is split into two regions, as shown in Figure 4.23. The forward uses defensive positioning in the bottom region of the field. However, when the ball is in the top region, the long distance supporter switches into hard attack mode. Instead of positioning itself on point X, 155cm behind the ball and directly between the ball and its own goal, the supporter selects the attack vantage point that is on the far side of the field, i.e. point B, and positions itself on the line BX. The exact position is determined by linearly interpolating between points B and X so that as the ball moves further upfield, the desired position moves towards point B, which is the attack vantage point.

X positioning

In X positioning, four lines are drawn to form an X, as shown in Figure 4.24a. The lines AC and BC, formed by connecting the attack vantage positions to point C, define a locus of positions for the long distance supporter during an attack. Similarly, during defence, the supporter moves along the lines DC or EC, formed by connecting the clearing vantage positions to position C.

During defence, when the ball is on the left half of the field, the supporter positions itself on line CE. Similarly when the ball is located on the right half, the supporter positions itself on line DC. The exact position is defined by a

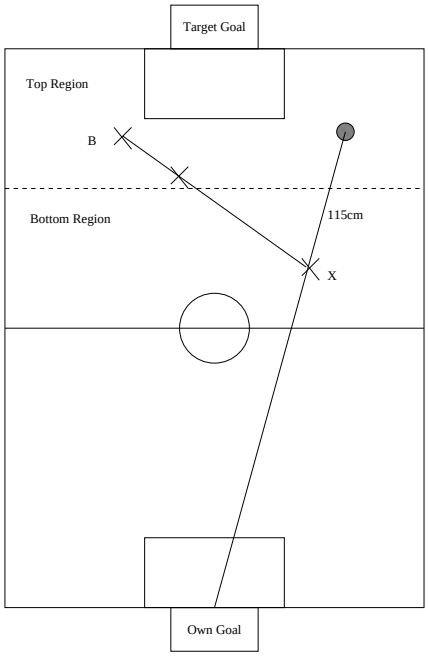
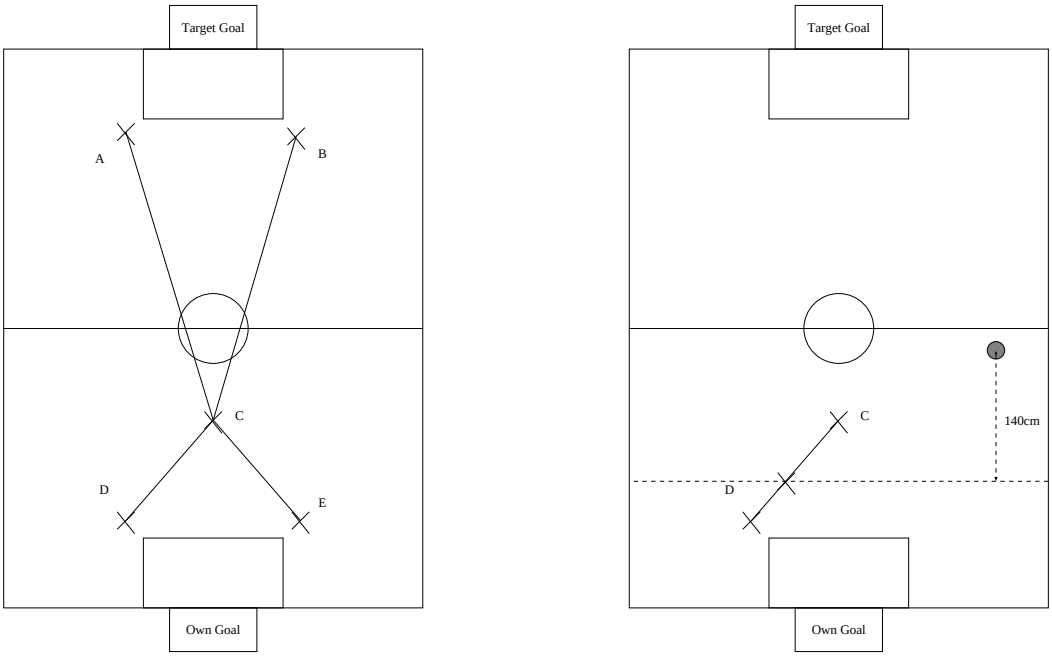


Figure 4.23: Hybrid positioning.



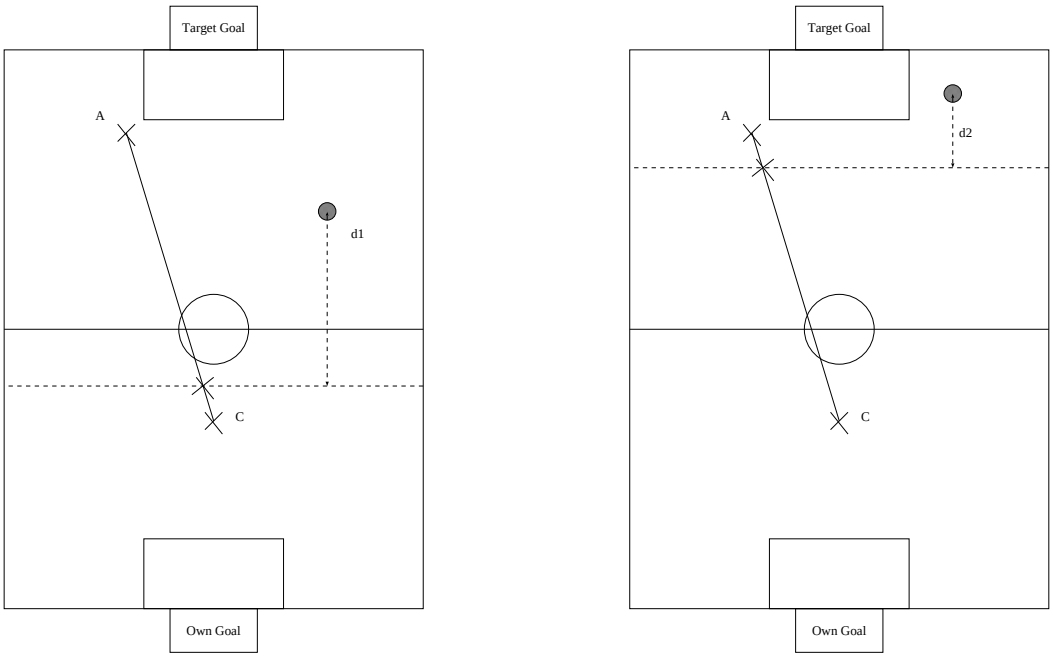
(a) (b)

Figure 4.24: (a) X positioning critical points. (b) X positioning defensive mode

fixed y distance behind the ball, capped at the two endpoints as shown in Figure 4.24b. The backoff distance used during the Robocup competition was conservatively fixed at 140cm.

The attacking scenario follows a similar rule where the long distance supporter positions itself on the line that is on the opposite side from the ball. However, rather than staying a fixed y-distance behind the ball, the forward has a backoff y-distance that is a function of how far the ball is upfield. As the ball moves upfield, the backoff distance decreases, allowing the supporter to locate itself more aggressively near the target goal. The X positioning method was found to be more effective than the other techniques and was therefore used in the Robocup 2003 competition.

In all of the above positioning techniques, as the long distance supporter moves towards the desired location, its heading is continually adjusted to ensure that it faces the ball. This allows the supporter to quickly change its role whenever the ball becomes close. Moreover, the supporter periodically takes its eyes away from the ball to actively localise by glancing at the beacon that it ex-



(c) (d)

Figure 4.25: (c) X positioning attack mode: the long-distance support positions itself on the line, a certain distance behind the ball. (d) X positioning attack mode: the long-distance supporter positions itself closer to the ball as well as the target goal as the ball moves upfield.

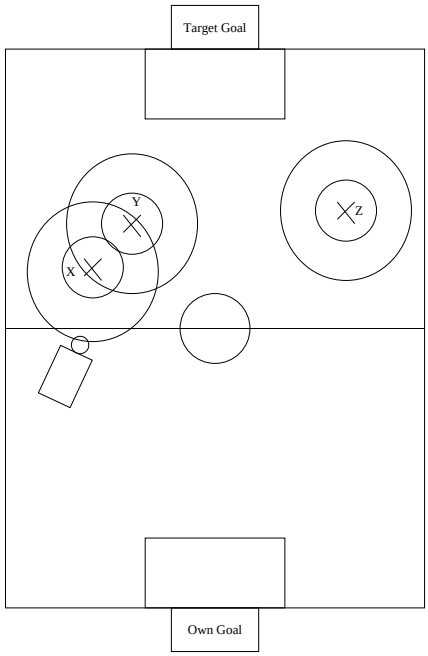


Figure 4.26: A bias to remain stationary when the supporter has moved within a distance of 10cm from the desired position.

pects will reduce its heading variance the most. Once the supporter has reached an accuracy of 10cm of the desired location, it remains stationary until it is no longer within 20cm accuracy of the current desired location, either because the supporter's position has been updated, or because the ball has moved. In Figure 4.26, the supporter moves towards point X, the desired position, until it is within the small circle centred at X, with radius 10cm. Once it is within the small circle, the robot becomes stationary, and has a slight tendency to remain stationary. For example, if, on the next camera frame, the calculated desired position is at point Y, because the robot is still within the large circle centred at Y, it remains stationary. However, if the newly calculated desired position is far enough from the robot's current position, such as point Z which is greater than 20cm away from the robot's position, the robot will move towards that point.

4.7 Local Interactions and Cooperation

In addition to possessing a global form of cooperation between the robots, where the robots are spaced out, the close-in cooperation and local interaction is an important feature of the UNSW/NICTA strategy.

4.7.1 Overview

As delays of 500ms were common in the wireless communication system, the close in cooperation is heavily reliant on the vision module. The fundamental skill from which this strategy is based is the visual back off. Essentially this means to step back from a teammate robot upon visually seeing the teammate. The strategy works by letting two robots chase after the ball. In most cases, when a robot cannot see any of its teammates, it will simply continue to attack the ball. However, it is the decisions that are made when a robot can see its teammate that produces the local cooperation.

The first component in the strategy is the trigger. That is deciding when you have visually seen a teammate that you should possibly back away from. This triggers an algorithm that results in 1 of 3 actions:

- Attacking the ball, because you are in a better position to attack than your teammate.
- Backing off to the support position, because the teammate is in the better position.
- Circling behind the ball to break symmetry. Ie when you cannot decide to do one of the above actions.

The decision as to which action to perform is made based on three aspects. The robots position relative to the field, the ball and its teammates. The overall effect is subtle but exceptionally effective.

This discussion shall now describe basically how the algorithm works, and then elaborate by discussing the finer implementation details and exception cases.

4.7.2 Introductory Description

First of all, we introduce the concept of a "Desired Kick Direction" (DKD). Depending on where the ball is relative to the field, the robots will want to move the ball in a different direction. This desired direction will help in determining which robot is in the better position relative to the ball, and thus which robot should attack the ball. Generally, the DKD is away from the robots own goal, and towards the target goal. Fig. 4.27 illustrates the DKD.

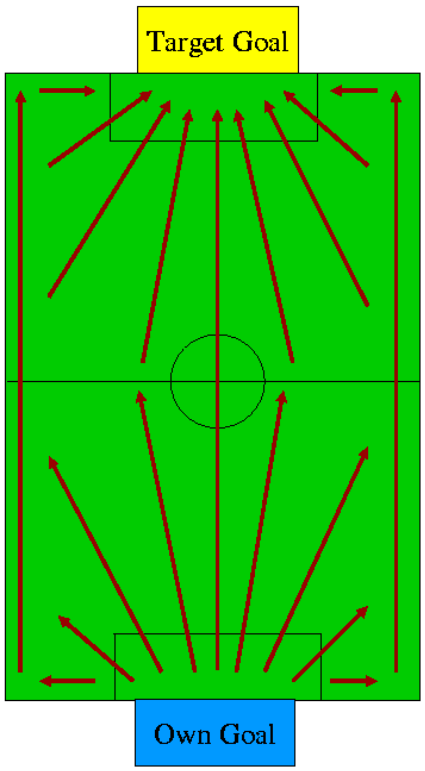


Figure 4.27: Desired Kick Directions

After the DKD has been determined, we draw a star, centered on the ball and pointing in the direction of the DKD. This splits the area around the ball into regions, and it is the teammates positions relative to each other on this star that determines which is to attack and which is to support. On this star we essentially work in polar coordinates, with the ball as the origin and the DKD as the positive x-axis. That is, it is the relative angles away from the DKD line and the distance to the ball that affects the decision.

Generally, the behaviour is defined based on the following rules: -

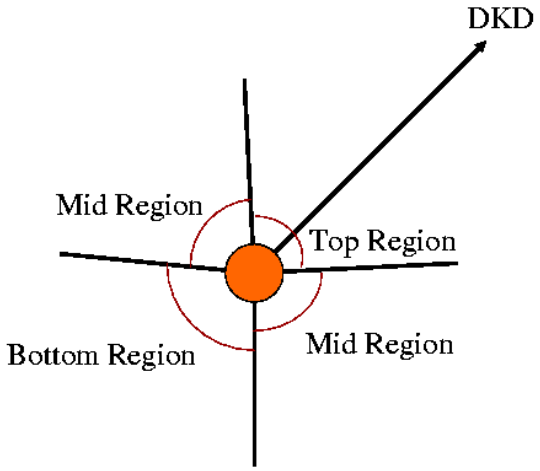


Figure 4.28: Star, dividing the area around the ball into regions

- If a robot is in the bottom region of the star, and the other robot is not, then that robot will attack the ball and the other robot will go to the supporting position.
- If both robots are in the top region, then the one who is closer will attack the ball, and the other will get out of the way as the closer robot will probably perform a 180 degree kick. After getting out of the way, that robot will move into the support position.
- If both robots are in the bottom region, then the robot to attack the ball is determined based on the following:
  - Which robot can actually see the ball. (as opposed to wireless knowledge)
  - Which robot is closer to the ball.
  - Which robot is closer to the DKD. (in terms of polar angle)
  - Which robot has the lower player number.
- In all other cases, both robots will closely circle around the ball towards the DKD until one stops seeing its teammate, and thus is free to attack the ball, or one enters the bottom/attacking region. Circling around closely allows the robots to maintain a defensive position relative to the ball.

No matter which action is chosen, the robot will always be facing the ball.

If a robot decides to attack the ball, it will simply execute the main forward attack strategy. If the robot is not attacking the ball, it will position itself to prevent the ball from traveling towards the defending goal, and to get ready to

take over the attack towards the target goal. This supporting position is based on where you are on the field, and relative to the position of the ball.

If the ball is in the top quarter of the field, the supporting robot will position itself in an "L" shape to the side and behind the ball, such as that depicted by points A and B. The robot will go to point A or B depending on which side of the DKD line it is on. If it is to the left of the line, it will go to the left most position, ie A, otherwise it will go to point B. The reasoning behind this is that this will keep the dog from obstructing goals by traveling between the ball and the target goal. Probabilistically, it will also have the dog traveling to the closer support position, and it will have the dog closer to the center of the field, and between the ball and its own goal. The support position is back more than it is to the side. This is so that the supporting robot will more often be behind any loose balls, ready to take over the attack.

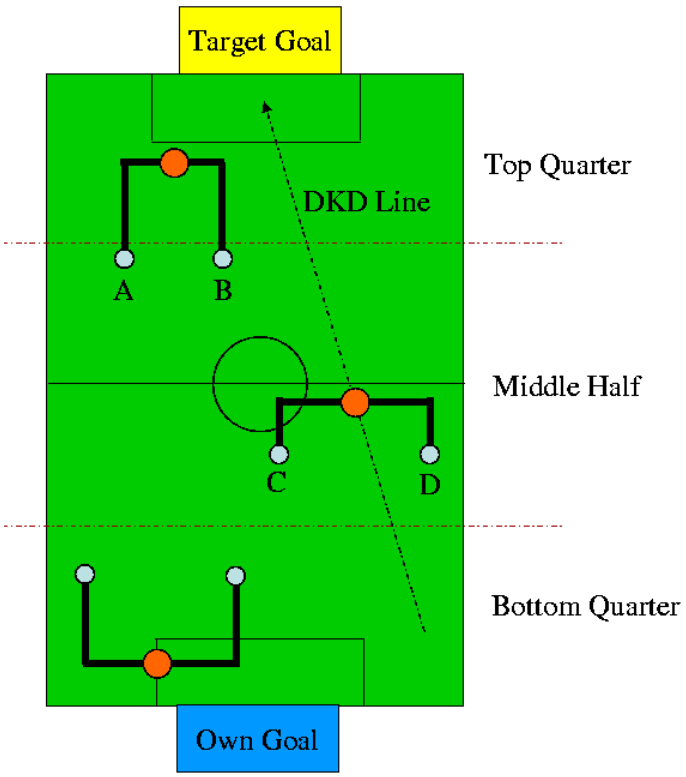


Figure 4.29: Various "L" support positions

If the ball is in the middle half of the field, the supporting position will be in a wider L shape such as that depicted by points C and D relative to the ball. Again the robot will choose between points C and D depending on which side of the DKD line it is on. The L is wider here so that the supporting robot is

wider, ready to intercept “sideway passes”.

In the bottom quarter of the field, ie the defensive quarter, the support position is in an L shape in front of the ball. Instead of positioning itself defensively between the ball and the defending goal, it is away and in front of the ball. This is so that the support robot stays out of the attacking robots way, minimizing the chance of getting leg locked with that robot and inhibiting the defense. It is also then able to capitalize on any opportunities where the ball becomes momentarily loose from the opponent attacker.

4.7.3 Defensive Support Positions

An emergent property of the up field “L” shape in the bottom third of the field is that the two robots tend to move in a defensive circular path around the bottom corners.

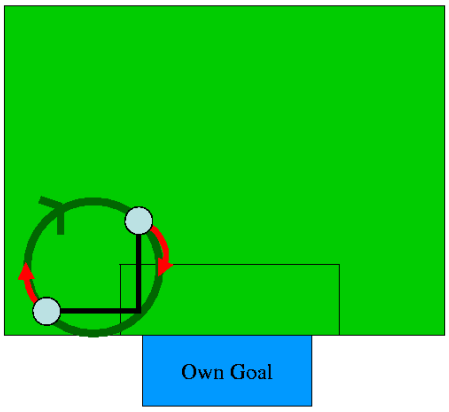


Figure 4.30: Defensive curcular path around the bottom corners

Typically, the attacking robot will be in the in the bottom corner trying to stop an opponent from progressing any further with the ball. Meanwhile, the supporting robot will wait in the supporting position. As the first robot loses control of the defense and moves out of the bottom region of the star shape, the supporting robot will take over the attack and the first robot will then assume the support position. In this way, the robots exchange positions, moving in a circular fashion until the ball is out of the corner.

A special case that needs to be discussed is what happens to the supporting position when the ball is near the side edges. In this case, the supporting position will be against the side edge and behind the ball.

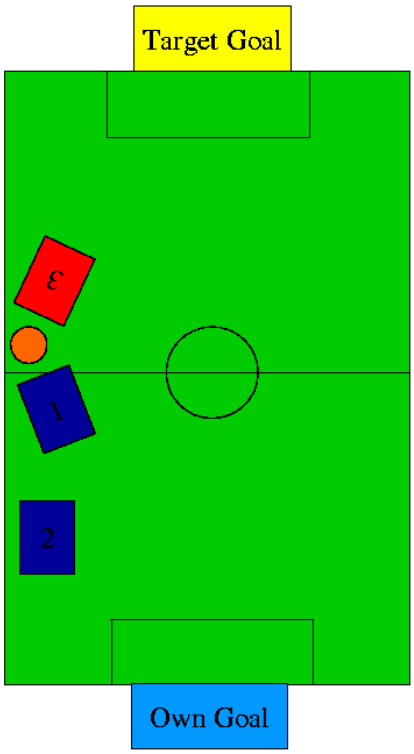


Figure 4.31: Side edge support positions

Often the attacking robot will get into a scrum on the side wall with one of the opponent robots. In the case where the opponent robot wins and gets the ball past the attacking robot, the supporting robot is there as a second line of defense to stop the ball from advancing too far towards the defensive goal, and as a second chance to clear the ball. Combined with the locate ball kick (see Skills section) this was effective in stopping the frequent plays that ended up on the side edges.

The behaviour of local cooperation in the UNSW/NICTA strategy is a result of having Desired Kick Directions, Star shaped region dividers, L shaped support positions and the ability to visually see teammates and opponents. It is the combination of local cooperation skills and global cooperation skills that results in an effective team strategy. Although to the unfamiliar eye, the behaviour may not be obvious, the behaviour, subtle as it is, is quite deliberate, planned and effective.

4.7.4 Symmetry Breaking Using Get Behind Ball

As mentioned above, there are cases where a clear decision cannot be made deciding who is in the better position to attack the ball and who should move into the support position. This occurs when one robot is in the mid region of the star and the other is in the top (case 1 of Fig. 4.32), or, when both robots are in the mid region (could be in the same mid region or opposite mid regions). In these cases the strategy tells the robots to circle around behind the ball until a clear decision can be made. That is, one robot enters the bottom region, or one robot stops seeing the other robot. If one robot is in the top region and the other is in the mid region, then presumably the robot in the mid region should find itself in a position to attack first. Similarly if they are both in the mid region (either same side or opposite), the robot who is closer to circling into the attacking region will probably be able to attack first.

When the robots are on the opposite mid regions of the star (case 2 of Fig. 4.32), it is easy to imagine that both robots will be seeing each other, and the first to loose sight of the other as it circles around behind the ball will be the first to be allowed to attack. However it is not always the ideal case that both robots are seeing each other and both are circling around the ball. Often only one robot can visually see the other robot and thus only it decides to circle around, whilst the other robot continues to attack the ball. There are 2 remedies that we use in this situation. The more intricate solution involves transferring wireless information between the 2 robots allowing each to know whether or not the other is backing off (be it get behind or support). The simpler solution involves adding a backwards vector to the get behind ball routine so that if one robot is attacking the ball, the other move away from it as it circles to avoid becoming leg locked with it. The robot has a skill that works out the walk direction needed to circle around the ball. We simply add a vector to this walk direction that points away from the ball to the robot. This solution had its good

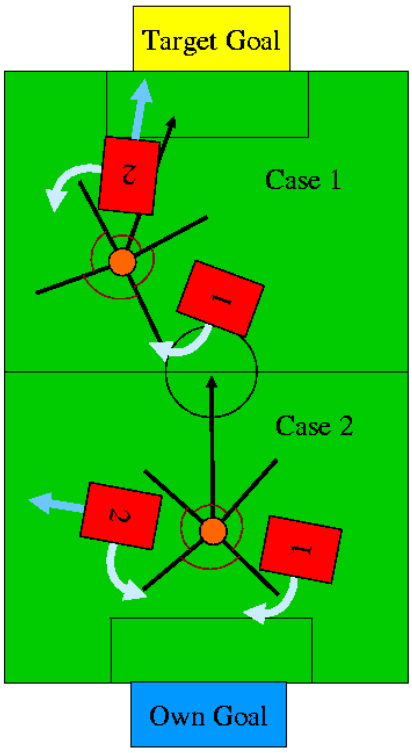


Figure 4.32: The diagram shows the get behind behaviour of the robots depending on their position in the backoff star. Light blue arrows show backoff direction. Darker blue lines show backwards vector.

points, but when both robots are correctly circling around the ball, it left the ball not as well defended, causing the robots to occasionally lose possession of the ball. In order to retain the advantage of the backwards vector and minimize the disadvantage, a compromise was made, where only the robot that had the greater distance circle around would apply the backwards vector.

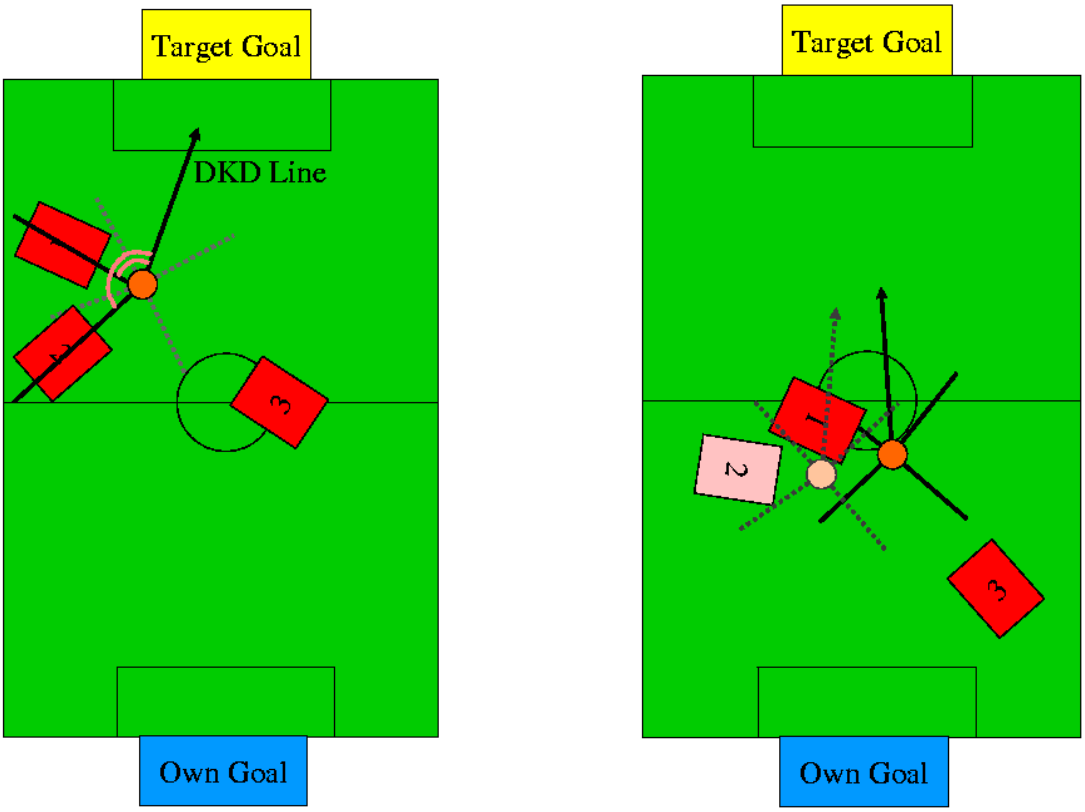
4.7.5 Back Off Trigger and World Model Teammate Matching

The trigger that causes the execution of the local interaction algorithm described above has been greatly simplified this year. If a robot can see both a teammate and the ball within 1 meter of itself, then it will run the local interaction algorithm. There are absolutely no locks in the algorithm. Every single camera frame, the algorithm must run again upon the trigger triggering again. This maximizes the responsiveness of the robots and keeps the role switching very dynamic.

Previously, the robot needed to see a teammate for a certain number of frames before it would back off. Once it had decided to back off, it would continue backing off for a fixed number of frames. There are two locks here. One that starts the back off, and one that sustains the back off. This reduces the robots responsiveness, but stopped any indecisiveness caused by flickering between backing off and not backing off. The difference between the 2 strategies is that the old trigger triggered the back off immediately. The new trigger merely triggers another strategy that decides between the 3 main actions described previously. That is attack, support or get behind.

The trigger needs to return the location of a seen teammate, as that information is used in the local cooperation strategy. In the event that more than one teammate can be seen, and thus there are potentially 3 robots running the local cooperation strategy, the robot that has the larger polar angle from the DKD line (with ball as origin) will be returned. This will be the teammate that is in the better position to attack the ball, and thus the robot that is seeing the teammate is more likely to go into the supporting position. The rational is that it is better to have 2 robots supporting than 2 robots attacking and getting in each others way.

Once a teammate is chosen, visual information about its location is fed into the local cooperation strategy. Before doing so however, the robot tries to match this teammate to one of the teammates in its world model. If a high probability match is found, the local cooperation strategy is run using the more accurate world model information about the teammates location instead. The ability to match a visual teammate to a world model teammate means that a whole lot of additional information is also available for use in the local cooperation algorithm. The robot will also know where the teammate thinks the ball is



(a) (b)

Figure 4.33: (a) Teammate selection for backoff algorithm when more than 1 teammate can be seen. Robot 3 which can see both robot 1 and 2, selects robot 2 to input into the backoff algorithm because robot 2 is in a better position to attack. (b) World model matching on teammates. The dark robot and ball represent visual information. The lighter robot and ball represent the world model information. If its probable that the world model robot is the same robot as the visually seen one, the world model robot is used instead.

and where the teammate thinks this robot is. The robot will then be able to calculate more accurately what the teammate is going to do by calculating which region it is in using details supplied by the teammate. Being able to match a visual teammate to a world model teammate also means that explicit back off information can be communicated between the robots.

4.7.6 Wireless Back off Information

A component of the wireless strategy information that gets passed between the dogs is a single bit that indicates whether or not a robot is currently backing off, ie supporting or getting behind the ball. This is useful in cases where a robot thinks that it is in the better position to attack and thus the teammate should support, but the teammate isn't actually seeing this robot and thus is attacking anyway. Using the wireless back off information, a robot that wants to attack will be able to infer that its teammate is attacking, and thus refrain from attacking. This is dangerous however, because wireless information can be up to 12 camera frames old before it expires. Instead of backing off into the supporting position, the robot circles around the ball into the optimal attacking position, still closely guarding the ball but trying to give the teammate some room to attack.

4.7.7 In Face Back Off and Side Back Off

There are another 2 minor types of back off that are used in this local cooperation strategy. If a teammate is directly in a robots face, and the robot has decided to play the support role, then the robot will take a couple of steps directly backwards until the teammate is no longer in its face. For a teammate to be "in a robots face", it has to be less than 35 cm away from it, and within a 90 degree angle in front of it. This is used to prevent or discontinue leg locks when robots are very close to one another.

The final type of back off is the side back off. Since the robots spend most of their time facing and chasing after the ball, if 2 robots end up side by side, running after the ball, it is unlikely that they will see each other, but very likely that they will converge and become leg locked. The side back off is thus based on world model information, and as the name suggests occurs when 2 robots are side by side. When this situation is detected, one of the robots will start to walk a little bit slower so that the other robot will be in front of it. Now that the robot is behind the other robot, it will start backing off to the support position as normal.

The decision as to which robot should perform the side back off is based on distance to the ball and the robots player number. If one robot can confidently

say that it is further from the ball, then that robot will slow down, otherwise it is the robot with the higher player number. Since the move reduces the robots aggressiveness, the robot must be confident that it is side by side with another robot, and so the side back off trigger is quite tight. It will trigger anytime there is a teammate less than 50 cm away within 50 degree angles to its sides. However, the variance on the teammateś location (and its own) has to be small. The diagram below illustrates the expected behaviour as a robot transitions between a side back off, and the normal back off.

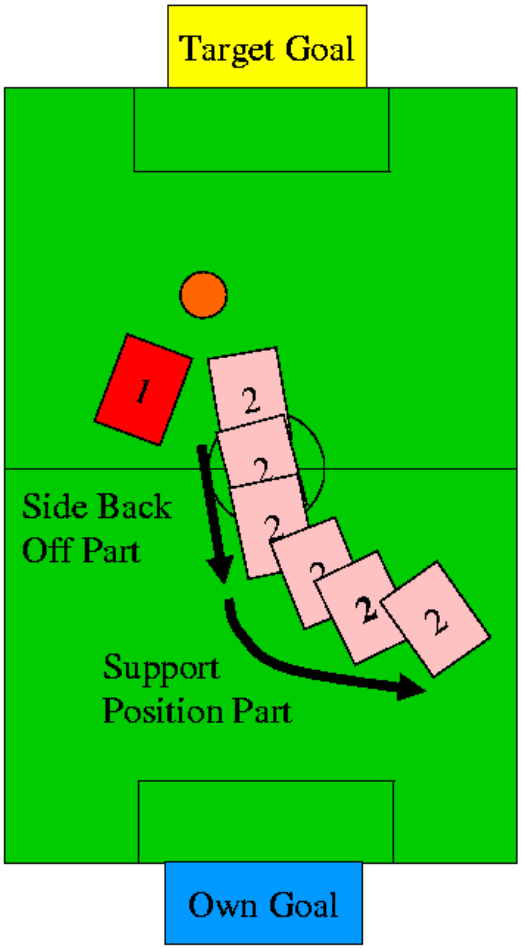


Figure 4.34: Transition between side backoff and support positioning.

4.8 Bird of Prey

The rUNSWift code in 2003 created a forward behaviour that was overall very aggressive, the main aim is to be first to the ball, and keep it moving in the most desirable direction possible. This requires the full attention of all the forwards, two directly attacking the ball and one out of the play waiting for an offensive opportunity. Unfortunately, in this particular team strategy, there is no place for a defensive player, i.e. a player to sit well behind the play and catch balls that drop behind the other players.

The Bird of Prey (BOP) was a behaviour that was developed to counter the lack of a defensively assigned player in the game. It is a dynamic defensive player that capitalises on rUNSWift's superior speed to place itself between the defensive goal and the ball as quickly as possible. The Bird of Prey consists of two main processes, determining when cover defense is required and which player should act as Bird, and what actions the Bird takes.

4.8.1 Deciding on Activation of the BOP

Determining the activation of the BOP requires determination of when defense is critical and cover defense is required, if the specific robot is the player in the best position to provide that cover defence and when the BOP should be stopped, since it is a lock mode (emulates a state in a decision tree).

The decision to activate and deactivate the BOP are very similar, and in theory, could actually be the same decision. To avoid switching between active and inactive at the inevitable boundary between these, we actually make them slightly seperate, adding hysteresis on some conditions and dropping others in the decision to become inactive. The keys to the decision are: -

- Number of Players - the BOP first checks how many forwards are currently on the field. The locus of the BOP is not straight to the ball, and hence is not the fastest route to the ball. If the robot has no communication with any other forward the idea of cover defense is abandoned, as it is best to go straight to the ball and try to gain control first. An example of this might be if two forwards are currently penalized and hence there is only one left to go for the ball. It also checks how long since all three players were communicating, if this is less than 3 frames it is assumed a wireless problem and BOP is allowed.
- Position of the ball - The position of the ball is the logical state to consider when deciding if cover defense is required. We are particularly interested in the ball's position relative to our teammates (including the robot itself). If the ball is generally behind the team cover defense is required.

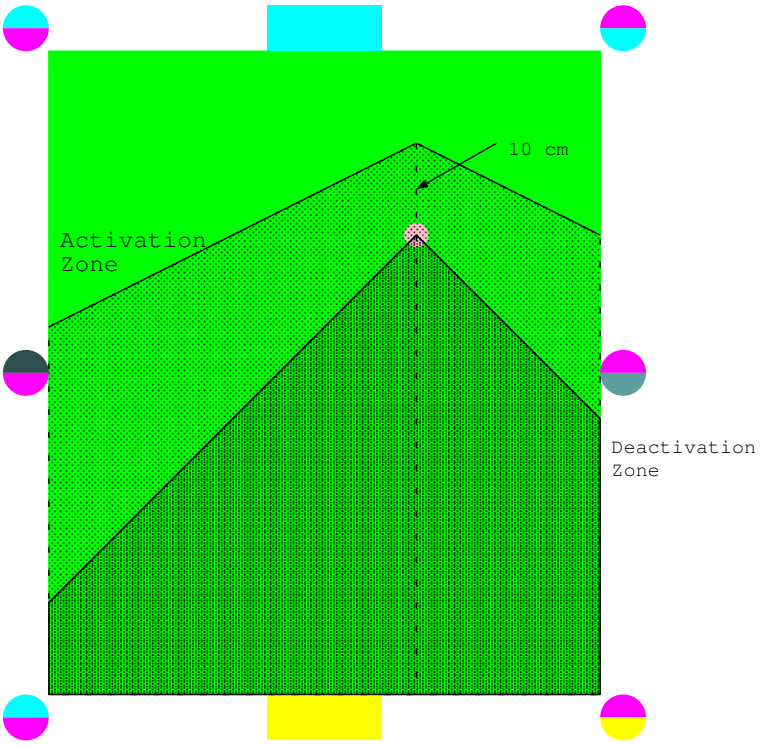


Figure 4.35: BOP Regions for checking for defending forwards

Fig. 4.35 shows the particular areas we look for teammates, excluding our goalkeeper, to determine if the ball is behind the team. We look in the area enclosed by 150 degrees, centered straight down the defensive direction, 10 centimeters in front of the ball, if none of our forwards are in that area cover defence is required. This area allows room around the ball, hence if one of our players has possession the defense is not initiated. The angle allows for the fact that a teammate across the opposite side of the field than the ball, but level with it, is not in an adequate position to counter an enemy offensive maneuver. For deactivation the angle is reduced to 90 degrees and the distance to 0 centimeters, this forms a hysteresis so there is no boundary to continually cross. While the robot itself is included in the activation check, it is not included in the deactivation check. A check is also performed to determine if the ball is in the goalbox, in which case it is the keeper's domain and cover defense is not initiated.

- Most eligible BOP - For most of this year, particularly in competition, the BOP was only performed by a single robot on the field. The idea is to provide cover defense while the other forwards move straight to the ball, gaining possession, or at least slowing the opponent robots down untill the BOP can be on the defensive side of the ball. A key to the success of the BOP is the ability of the chosen robot to avoid contact with other players while moving swiftly into the cover defense position. The BOP is chosen as the forward with the greatest x-distance (i.e. furthest laterally) from the ball. This allows the BOP more room to perform it's curved locus into cover defense, means that the bulk of players is usually avaoided and also usually leads to the player furthest from the ball to move into cover defense allowing the closer forwards to slow the play. This works particularly well with rUNSWwifts formation of two forward on the ball and the striker laterally offset from the ball's position, as the striker usually takes the BOP role.

The conditions do not include the condition of the BOP cover defense considered to be attained and the BOP to move out of BOP mode and attack the ball. This will be covered in the BOP action section.

4.8.2 The BOP Action

The BOP action is extremely important due to it's purpose, and robustness is required. There are some special cases where inaccuracies generally lead to large problems in the BOP locus, these are checked in the code, apart from these the BOP is built on very simple ideas.

The first thing to consider is that it would be of great advantage to keep the BOP as a local behviour, this entails a decoupling of the behaviour from global position of the robot or the ball. While this cannot be totally achieved, it can be for the general locus. The locus works on a simple concept, maintain the ball

at a particular heading untill we reach an adequately defensive position. For example, for the competition the static heading was set to 45 degrees, this forms a curve that circles the ball , spiralling towards it. Apart from the two cases (go left or right) decided on global information, this behaviour is a product of our position relative to the ball. It also adapts to the movement of the ball very well, so the BOP robot will stay in field from an opponent pushing the ball down the side, eventually forcing the ball into the side or corner.

To decide which side to maintain the ball on, we draw a line from halfway between the center of the goal and the ball's x coordinate, through the ball. The side of this line that the robot is on is opposite to the side we want to maintain the ball heading, and is the side that the locus will curve around. The line also forms the position we would like the robot to be in before attacking the ball. Fig. 4.36 shows BOP loci given a position of the ball. These loci are the desired direction the BOP action calculates, we can see they generally circle around the ball to the approach line, when close enough to the line a circular locus is taken as described below. We use an angled line because we would like the cover defensive player to push the ball away from the goal and up the field, approaching the ball from a directly defensive position (i.e. have the line parallel to the y-axis) sometimes results in the ball being forced to the side as the opposition attempts to push it in exactly the opposite direction. This may lead to the ball travelling infield and hence a much more offensive position for the opponent attackers. The angular line leads to a less direct push against the opposition offensive, hence if the ball does get forced laterally, it is most likely to be towards the wall, a much better position defensively, than infield.

The main locus of the BOP does not necessarily join tangentally with the threshold line, hence the maximum speed circular locus is used. This is the circle that the robot can walk around at full speed. It is a semi-calibrated locus that is approximately 18.0 centimeters wide. Once the robot is within the range of the threshold line such that a circle of this radius will hit tangental to it the robot enters the tight turning circle locus untill it's facing the ball. The calculation of the circle relative to the threshold line is done by taking the perpindicular distance to the threshold line, when this is less than the radius of the circle, tight turning mode is required. The exactness of this procedure is not critical as any minor errors lead to a slightly different point of attack, still moving the ball away from the defensive goal. This measure allows the dog to perform the entire BOP action without switching walk mode due to tight turns, and allows it to maintain full speed, making the move a swift and decisive action.

4.8.3 Goal Box Evasion

The BOP is generally called when the ball is positioned in our defending half, since it must be behind all our forwards. Unfortunately, the looping locus often pushes the active BOP robot well back behind the ball, and often in the center of the field, this often leads to the robot entering the defensive goalbox. This

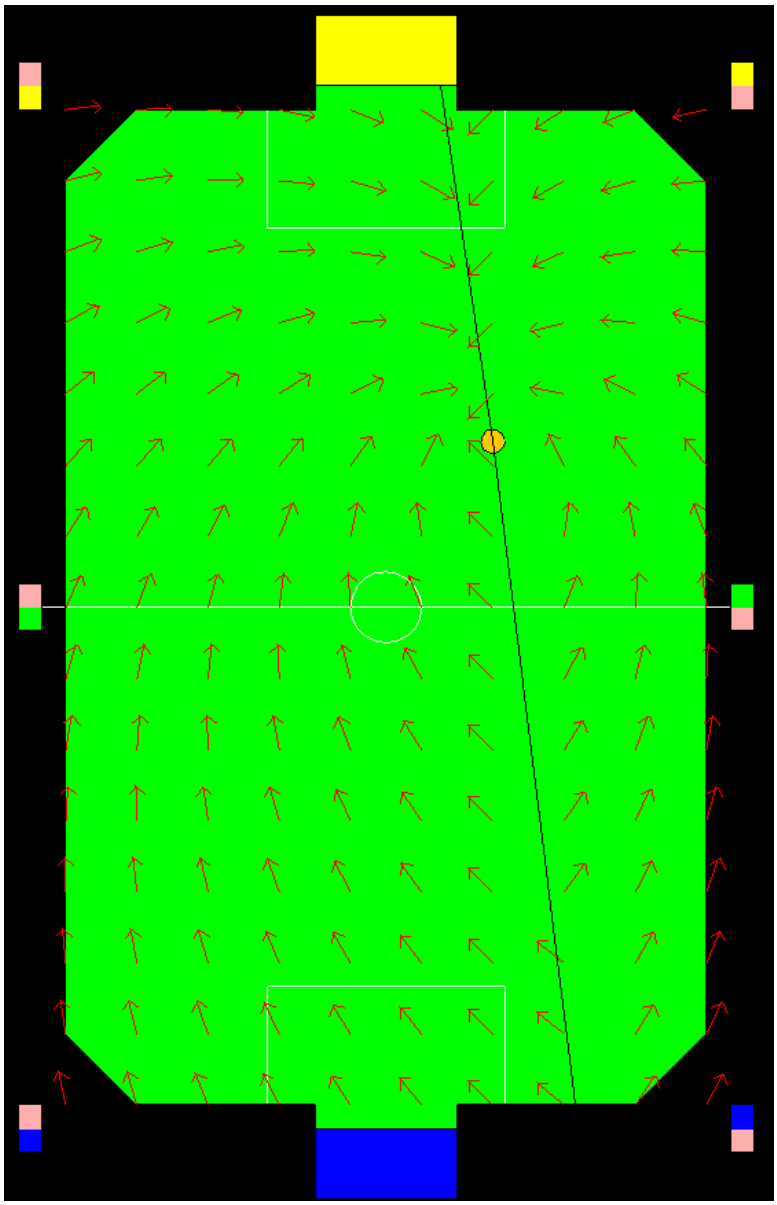


Figure 4.36: BOP Locus for a red player shown in red from discrete samples

situation is made even worse since it occurs when the ball is in one of our defensive corners, a frequent event since the corners attract the ball.

Due to the harsh line taken to the illegal defender rule in 2003, i.e. any illegal defender would be taken from the field for 30 seconds, this situation is unacceptable. While avoiding the box requires extra complexity, draws the BOP away from the smooth action desired and also adds more global dependence to the BOP, the continual removal of players from the field occurs alarmingly often when the case is not specially considered.

To avoid the box a simple check is done as to whether the robot is about to enter the defensive goal box. This check is again based on the tightest turn possible, with hysteresis to avoid switching repetively between box evasion and normal BOP mode. The checks, and actions, are performed as follows :-

- Normal BOP → Box Evasion - This is the main decision to enter goalbox evasion mode. This transition takes place if we are within 25cm of the top goalbox line, between the goalbox edges and our desired heading is between 0 and 180 (negative y-direction). Since the behaviours structure is a decision tree this condition is continually checked, for hysteresis purposes the distance to be within the top line of the goal box to maintain goalbox evasion mode is 50cm.
- Box Evasion - Box Evasion mode simply consists of trying to maintain a heading perpindicular to the y-axis (across the field), in the general direction of our desired heading (from normal BOP). The beginning of this will require a tight turn, this is done at maximum turn whilst maintain the maximum walking speed mode (offset walk for the competition). Hence the robot does not need to drop into lower speeds for maneuverability and can maintain momentum and a smooth path. As discussed above the robot may drop out of Box evasion if it's estimated position becomes more than 50cm upfield from the top of the goalbox, otherwise the robot eventually enters Post Evasion mode.
- Box Evasion → Post Evasion - If the robot does not eventually follow a path away from the goalbox, then the stopping case becomes the point where the goalbox ends. The Post Evasion mode is entered if Box Evasion is required, though we are very close to the end of the box in the direction we are moving. Transition to Post Evasion Mode occurs 10cm from the edge of the goalbox.
- Post Evasion - Post Evasion consists of again turning at maximum speed towards the end of the field, untill our desired heading for normal mode is met. Post evasion mode is usually necessary when the ball drops into our defensive corner, an extremely dangerous position. The smooth action, and cutting of the corner (not this circular locus begins before the edge of the goalbox), allows the BOP robot to move swiftly around the goalbox and defend our goal. The normal BOP then takes over, in this position it usually leads the BOP to hit the back wall before the ball, which is

a positive artifact since it stops the opponent attackers pushing the ball along the backline to our goal, a very common attacking maneuver.

Goal box evasion worked reasonably well, the robots often streaked the top of the goalbox maintaining two feet inside, the legal limit. The algorithm works without having to counter away from the box if the robot's estimated position falls into the box, this avoids alot more shifting between states, particularly very negative states moving in directions that would dramatically slow our chase of the ball. Unfortunately there seems to be a point along the top of the goalbox where self-localisation becomes inherently bad, it may be caused by partial goal observations mixed with the fact that the one solid beacon we can see provides only two dimensions of data, whilst three are need for exact localisation.

4.8.4 Conclusion

The Bird of Prey was an extreme success in practice and especially in games. It allowed the rUNSWift team to play it's naturally offensive game whilst maintaining a viable defensive presence. It made the most of the rUNSWift strengths, i.e. speed and simplicity, maintaining a local state to avoid global complexities. The final against UPennalizers showed the robustness of the Bird of Prey as it became the difference as rUNSWift maintained a lead given a high level of offensive pressure from UPenn.

The BOP is diffinitely a behaviour future rUNSWift teams will want to look at, and build on. The future should see a similar defensive action involve interaction from all the team's forwards rather than cover defence by a single player. The BOP was tried on multiple robots simultaneously, culminating in the infamous "Flock of Birds" formation, but it was found to be slow in some circumstances, when robots directly attacking the ball would perform better. A defensive action involving the closest player maintaining the direct chase to the ball, a BOP from wide and the second ball-chasing robot doing a seperate skill is clearly possible and should be explored in the future.

4.9 Dynamic Role Switching

With the introduction of different roles, we must ensure that role switching is decisive and is congruent with our philosophies on speed and aggression. Delays in wireless transmission can be up to 500ms and thus, to run election algorithms over the network to negotiate allocations of roles is too slow.

Transferring wireless strategy information is crucial in the role determination. The information transferred over the network however is less about sending explicit role calls but more about sending the necessary information for teammates to reproduce another teammate's role decision locally. Since wireless information can be up to 12 frames old, the reproduction of the decision may not be exact. Heuristics are used to make the best of the limited accuracy by biasing aggression whenever there is doubt.

In terms of global cooperation decisions, it is more tolerable to have delays in selecting a wider supporter and a bird of prey. Whenever a clear decision cannot be made to allocate this role, it is simply foregone and all three forwards attack the ball using the local interactions strategy to define behaviours between them. The local cooperation's strategy thus needs to be dynamic and react decisively to change. Reactions need to be decisive rather than simply quick, as simply quick may lead to a lot of role fluctuations, leaving no robot attacking the ball. Thus the local interactions strategy needs to be robust and carefully planned to get the right balance between reaction to change and decisiveness.

2 local interactions strategies were developed concurrently. One involved mainly using the vision system to trigger the strategy, and the other explored using wireless to govern the behaviour of the close supporter. In early implementations both methods were performing fairly equally well. However as implementations progressed, it was found that finer control could be implemented using the vision method rather than the wireless method simply because information refreshed more often, allowing the robot to be more reactive.

The role switching thus has 2 tiers of dynamic allocations that correspond with the 2 tiers of cooperation. More flexibility is allowed in the allocation of roles that participate in the global cooperation strategy, and thus these are based more on wireless information. The local interactions strategy however needs to be robust and well balanced in order to really be effective. It is fundamentally based on vision but then also uses wireless information in order to refine the behaviour when it can. It was rUNSWifts ability to not over focus on developing a predominantly global strategy and its ability to separate it into levels of cooperation that made the role switching and cooperation effective.

4.10 Goal Keeper Strategy

4.10.1 Fundamental concepts

rUNSWift's goal-keeper strategy revolves around a few fundamental concepts, established initially to capture rUNSWift's belief of what makes an effective goal-keeper:

- It should stay between the ball and its own goal when the ball is far away;
- It should attack the ball when the ball lies dangerously in the goal box;
- It should attack the ball when the ball is close to the target goal, but only when opponent robots are not nearby;
- It should effectively clear the ball by kicking it through gaps;
- It should return to its goal after successfully clearing the ball; and
- It should take its eyes away from the ball to actively localise whenever possible in order to obtain a more accurate estimate of its location on the field, clearly important in order for it to position itself correctly between the ball and its goal.

4.10.2 Detailed Description

In rUNSWift's strategy, when the goal keeper loses sight of the ball in the current camera frame, it first glances at the last known position of the ball and positions itself defensively on the point of intersection between a line connecting the ball's last known position and the centre of the goal, and an ellipse centred at the goal mouth, with major axis 35cm and minor axis 5cm as shown in Figure 4.37. The global heading of the goal keeper is maintained to face the GPS ball.

However, if tracking the GPS ball does not lead the goal keeper to catch sight of the ball, then the goalie proceeds to determine whether it should use information concerning the ball's location received from its teammates.

In the case where the variance of the wireless ball is below a certain threshold, the goal keeper decides to trust wireless information and hence positions itself directly between the wireless ball and the centre of its own goal. Again, the goal keeper positions itself on the intercept between the line connecting the

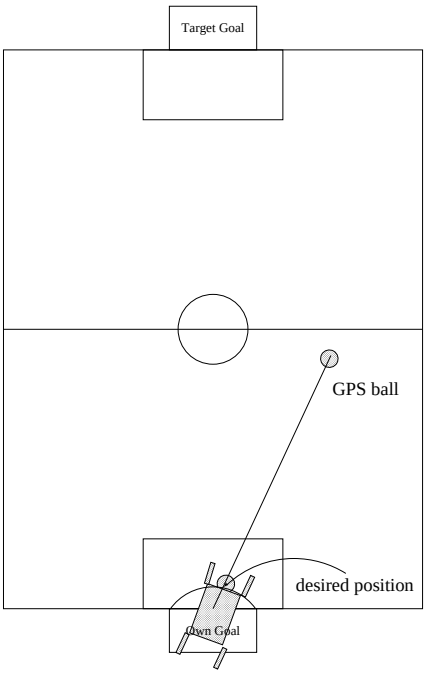


Figure 4.37: When the goal keeper relies on GPS ball, its desired location is the point of intersection between the line from the goal centre to the ball, and the ellipse.

shared ball to the goal centre and the ellipse, facing the ball. However, instead of tracking the wireless ball, the goalie continuously rotates its head to search for the ball, thereby increasing the probability of catching sight of the ball by increasing the area of the field that it searches.

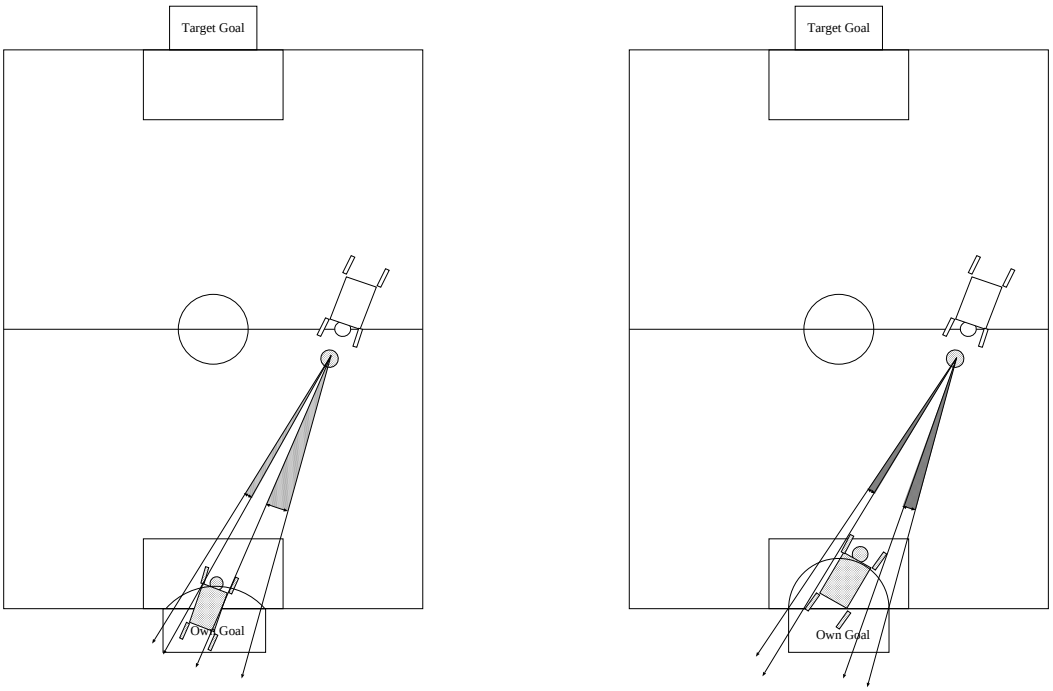
When the variance of the wireless ball is above the threshold and hence the wireless ball information is considered unreliable, the goal keeper positions itself in the default position which is the centre of its own goal, with a default heading which faces the opponent goal. In rUNSWift's strategy, this default position was determined to be (field width/2, wall thickness + 10). The keeper remains in this position and heading until it receives reliable ball information; this occurs when it either catches sight of the ball or receives wireless information concerning the location of the ball from its teammates.

The positioning for the goal keeper is different when the ball is visible. In all the instances mentioned above, where the ball is not visible, the goal keeper positions itself on an ellipse. This is intended to be defensive as the minor axis of the ellipse ensures that the keeper is positioned near the goal opening. However, when the ball is visible, the goal keeper positions itself further forward between the ball and its own goal, by finding the point of intersection between the line drawn from the ball to the goal centre, and a semi-circle of radius 35 centred at the centre of the goal. By coming out further from the goal, the goal-keeper effectively ensures that it reduces the size of gaps between itself and its own goal at which opponent robots can aim, as depicted in Figures 4.38a and 4.38b.

Furthermore, scrums in corners occur very frequently and hence many goals are scored from the side; thus, the goal keeper's positioning is modified so that after it has calculated the intercept between the line through the ball and the goal centre and the semi-circle, the final desired position is determined by adjusting the intercept so that its position is weighted more heavily towards the ends of this semi-circle.

When the heading of the point of interception from the goal centre is between 60 degrees and 120 degrees, the desired position for the goal keeper remains this point of interception. Weight redistribution towards the ends of the semi-circle only occurs when the heading of the intercept from the goal centre lies outside of these two limits, as shown in Figure 4.39.

A further important aspect of goalie positioning is the goal-keeper's localisation ignores the far beacons in the target half of the field as well as the target goal. This is because the far beacons are too small in the camera frame for the goal-keeper to obtain a reliable distance estimate; as the distance increases, the



(a) (b)

Figure 4.38: (a) The gaps through which opponent robots can aim when the goal-keeper positions itself on the ellipse. (b) The gaps through which opponent robots can aim when the goal-keeper positions itself on the circle, are reduced, as the goal-keeper is positioned closer to the ball.

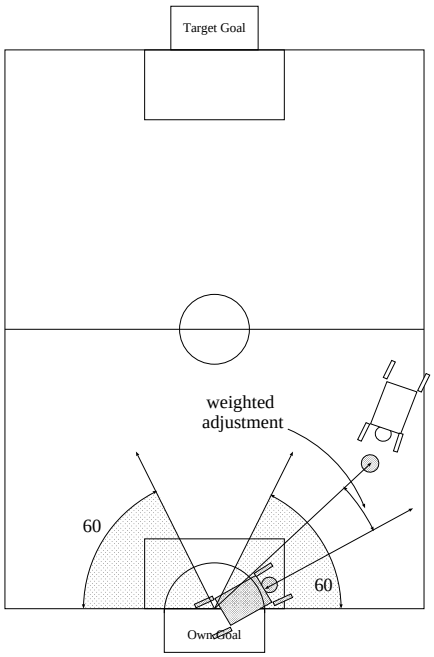


Figure 4.39: The weight redistribution adjusts the desired position of the goalkeeper and thus ensures the keeper pushes itself on the wall to close possible gaps on the side.

error in the distance estimate increases exponentially. It was discovered that if the far beacons were taken into account when determining the goal keeper's localisation, the keeper would continuously back into its own goal as the distances to the far beacons are systematically underapproximated.

When the goal keeper can see the ball in the current camera update, it can perform one of three actions: perform a block to prevent a score, attack the ball, or remain in its position on the semi-circle.

Goal block

The goal block is an action where the goal keeper spreads its arms in an attempt to prevent a fast-moving ball that is moving towards the goal, from entering the goal. Although this is useful at times, the condition for triggering this behaviour is very conservative and well-tuned. This is because it takes time for the goal-keeper to recover from the mobility lost from spreading its arms wide.

- Firstly, the ball must be either within 54cm from the goalbox or within 80cm from the goal keeper's position. This ensures the goal-keeper does not attempt a block when the ball is too far away for it to be a threat.

- Second, the velocity estimation of the ball is checked for reliability. If the estimation is reliable and shows that the ball is travelling towards the goal-keeper, the block is triggered. The innovation vector measure for the ball velocity is divided by the length of the ball's velocity estimate and this value is used to determine whether the velocity estimate is to be relied upon.

- Next, the number of frames for the ball to reach the goal-keeper is calculated in order to estimate the x-value of the ball upon reaching the goal keeper. The range in the x-axis where the goal-keeper has a chance of blocking the ball by spreading out its arms is computed. If the x-value of the ball lies outside this range, the keeper does not perform a block but instead walks to a better location to block the ball. Furthermore, if the ball is computed to collide with the goal keeper without the need for a block, the keeper does not perform a block.

- The final condition that must be satisfied in order for blocking to be triggered is when the ball is close enough to the goalie and travelling sufficiently fast to be able to collide with the goalie within the next second. This condition prevents the keeper from prematurely blocking.

Attacking the ball

The goal-keeper considers moving away from its desired position on the circle to attack the ball in two situations; first, when the ball is within 10cm of the goalbox; and second, when the keeper is inside the goalbox, the distance to the ball is within 35cm and the closest opponent to the ball is 40cm further away from the ball than the keeper.

However, there are a number of exceptions that prevent the goal-keeper from behaving too aggressively:

- In the event that the goal-keeper is inside its goal-box, the ball is in the corner and an opponent forward is closer to the ball than the goal-keeper, the keeper should remain in its defensive position and attempt to draw a foul.
- Moreover, if a rUNSWift main attacker has signalled via the wireless communication system that it is attacking the ball and is facing away from its own goal, then the keeper remains in its place inside the goal-box.
- The goal-keeper is restrained from moving more than 45cm from its goal-box to ensure that it not stray too far from its own goal.
- As well as this hard limit, there also exists a condition that has the effect of ensuring that the further the goal-keeper is from its goal-box, the closer the ball has to be to the keeper before it is allowed toattack the ball. As depicted in Figure 4.40, if the goal-keeper's position is such that its point on the graph lies outside of the shaded region, the keeper returns to its goal instead of attacking the ball.

Defensive stance

When none of the above "block ball" and "attack ball" conditions are satisfied, the goal-keeper remains in its defensive position and periodically looks at the four closest beacons to actively localise. The goal-keeper actively localises off the beacon that is estimated to reduce the its heading variance the most, before it glances back at where it last saw the ball.

However, before it actually starts to actively localise, the keeper first checks that the ball is not within 20cm of it. This is to ensure that when the ball is close, the goal keeper does not take its eyes off the ball. Second, if, using

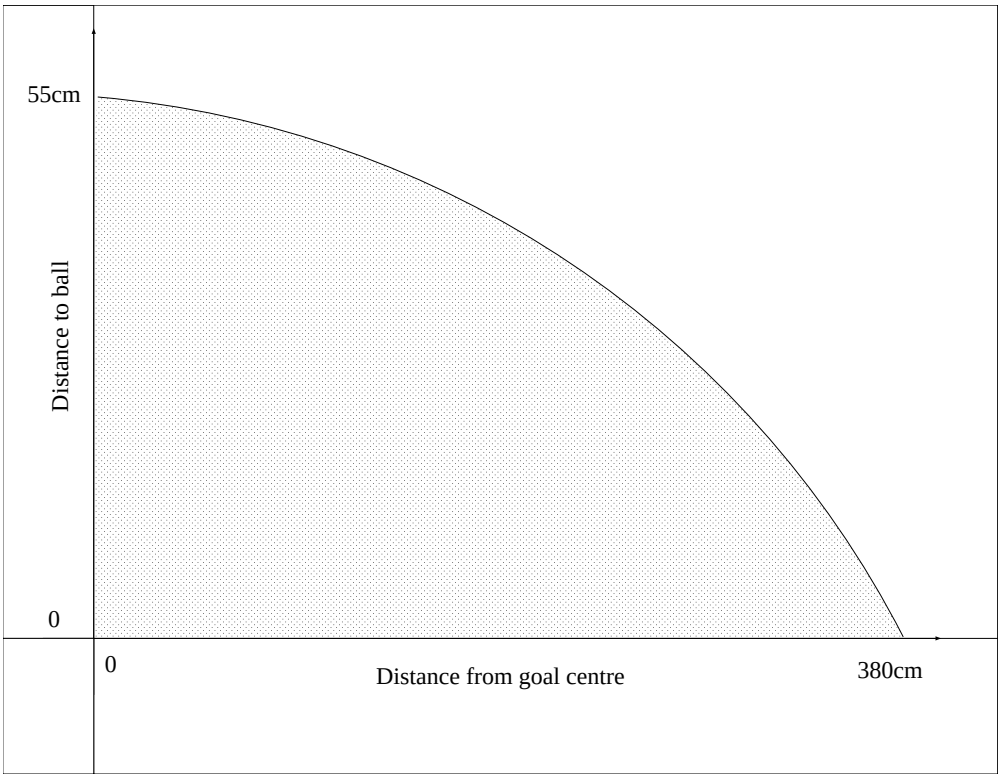


Figure 4.40: If the goal-keeper's position is such that its point on the graph lies inside the shared region, the ball is close enough for the keeper to continue its attack.

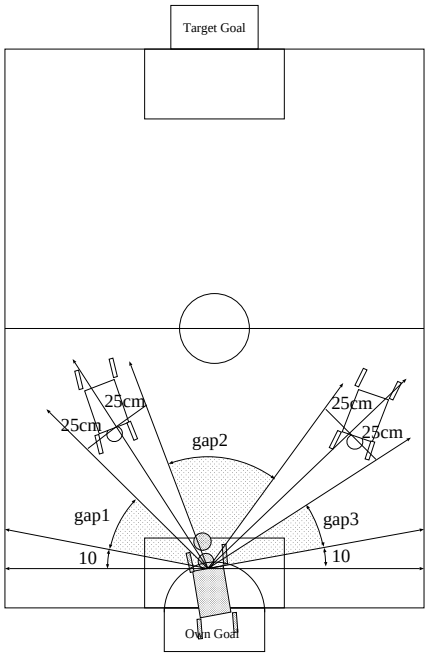


Figure 4.41: The goal-keeper will attempt to clear the ball through gap 2.

velocity prediction, the goal-keeper determines that the ball is heading towards it, then it does not take its eyes off the ball.

After the goal-keeper has gained possession of the ball, it first computes the gaps that are created by robots within 300cm of itself that are in front of it, as shown in Figure 4.41. The gap through which the goal-keeper aims to clear the ball is the gap that requires the least amount of time to turn to and is also sufficiently large to shoot through, that is, its size is above a certain threshold that is sufficient large for clearance.

When no gap exists that is sufficiently large, the goal-keeper performs a chest-push. On the other hand, if a gap has been selected, the goal-keeper turns towards the gap until its global heading is inside the gap by five degrees, in which case, the keeper performs a front-kick.

Chapter 5

Skills

5.1 Locate Ball

The locate ball skill refers to the general strategy used to find the ball when the robot cannot see it. There are 4 phases to the locate ball routine which are used depending on how long the ball has been lost for.

The first phase is used if the ball has only recently been lost. It involves having the robot stand on the spot whilst circling its head to search for the ball. The robot always starts by looking directly below it as it is fairly common for the robot to lose sight of the ball after it has grabbed it. The robot then circles its head in the direction that it last saw the ball. Whilst doing this, the robot actually stands a little taller than it usually does. This makes it easier for the robot to look directly down with without getting its head stuck on a hidden ball. This is especially required in the corners, where the ball could be directly below the robot but sitting on the corner ramp and thus be higher relative to the robot. The length of this phase is the amount of time it takes for the head to make 2 revolutions.

If the ball isn't found after the initial stationary search, the robot moves into the second phase of the search routine. In this phase, the robot is turning slowly on the spot whilst continuing to circle its head in search of the ball. The direction in which the robot turns is based on which side of the field it is on. If it is on the left half of the field it turns clockwise, otherwise it turns anticlockwise. It might seem more sensible for the robot to turn in the direction that the world model says the ball is in, but this direction choice was used because any accidental contact with nearby balls will usually push the ball towards the target goal. The head must circle in the same direction as the robot is turning otherwise

there are blind spots in the search circle. Like the first phase, the length of the phase is the amount of time it takes for the head to make 2 revolutions.

In the third phase, the robot makes 1 revolution turning quickly on the spot with its head no longer circling, but turned on its side, looking in the direction that the robot is spinning. The robot turns in the same direction as it did in phase 2 and is almost always able to find the ball if it is in clear view. In a game situation however, the ball is not often in clear view. A nice property of this style of spin is that the robot doesn't spins past the ball if it sees it. With the head turned on its side, it is looking in a direction perpendicular to the robots body. This means that the head will be pointing at the ball before the body does, giving the body time to rotate around, perfectly lined up to chase the ball.

If the ball is still not found, the robot enters the final phase of the locate ball routine and stays there until it finds the ball. In this phase, the robot turns slowly and circles its head like it did in the second phase. Since a robot cannot see a ball all the way across the length of the field, it spins towards a point near the center of the field rather than circling on the spot. Each robot is programmed to spin to a fixed point on the field. An improvement would be to make this point dynamic, or to use some sort of potential field between the robots to define where they spin to. In a game situation, the robots do not stay in this phase for long as they will either find the ball or be wirelessly told where the ball is. Because of that, we didn't spend the time to implement a smarter strategy to spread the dogs out.

The robot changes phase depending on how many camera frames have past since it last saw the ball. Thus, if it catches only a glimpse of the ball, it will start from the first phase again, circling its head on the spot to try and see if it was indeed the ball. This is good as the robot often only see the ball for 1 frame when it is spinning in circles and circling its head. However it could be a problem if the robot continuously catches glimpses of phantom balls as it will keep stopping the robot from spinning in a circle to find the ball. This is the tradeoff the programmer must consider. You definitely do not want to make the robot need to see the ball for 2 frames before it recognizes it as the ball as it often legitimately only sees the ball for 1 frame during its search.

5.2 Ball Tracking

Ball tracking involves only the head of the robot, and it is performed independently of the robot's other movements. The goal of this skill is to retain visual focus on the ball, and if possible maintain the ball in an area of view that would allow our ball recognition routine to calculate the most accurate readings.

Since ball recognition already provides us with the position of the ball, we should be able to point the camera at the ball's perceived position and thus achieve ball tracking. However, that would require close to perfect ball distance and heading calculations, which cannot be obtained reliably due to various noise introducing factors, such as variations in ball recognition, bobbing of the camera and errors in robot head joint readings. If the calculated ball position is not equal to the actual position of the ball, then in trying to track it with our camera, we could potentially be moving the camera off the ball.

So instead, we perform tracking based purely on the image position of the ball, and we try to aim the camera at the center of the ball, on the image plane. This is very reliable and simple to implement. The only complexities arise from translating our movement on the image plane to pan and tilt combinations. However by tracking the center of the ball, we can run into two potential problems. The first is that by tracking the center of the ball, it requires the head of the robot to be tilted at an angle such that we cannot easily see features on the field, including landmarks and other robots. Landmark seeing is vital to our localisation model and subsequently game play, as we would often find our robot lost after extended periods of ball chasing because it has not seen any landmarks, and thus fail to perform correct behaviours even though it was in possession of the ball in an advantageous position. This is not only frustrating to watch, but also very undesirable in competition play. Another problem is that by tracking the center of the ball, the robot would sometimes bump the ball with its camera when the ball is close, causing it to fumble and fail the ball grab.

The solution to the two problems mentioned is to track the top edge of the ball. This is a partial solution to the first problem, for although the landmark situation is substantially improved, we still do not see enough of the field often enough, and we need to rely on the Active Localisation skill. For the second problem, this method is a good solution. By tracking the top of the ball, it allows just enough room for the ball to roll under its chin, and the ball will lead the robot head to close in and secure the ball nicely within its grasp.

At the beginning of the year, we were suffering from frame rate issues that caused the robot to often fail in keeping up with a moving ball. As a temporary fix, instead of tracking the ball, we point the camera at where the ball should be if it continued to travel at the perceived speed, which we affectionately termed pre-cognition. We have found that it is possible for the robot to track incredibly

fast objects using this method. We are not certain how fast exactly, as we felt it was not practical for competition play, however the robot was able to track a ball kicked at full strength across the field by a human. The draw back was that this prediction is very sensitive to speed changes and tends to make the robot twitch unneccesarily with close balls. We fixed this twitching by carefully calibrating the regions at which pre-cognition kicks in. However this was an unneccesary introduction of unpredictable element in our behaviour as balls do not travel at that speed in competition play, so it was removed once our frame rate issue was fixed.

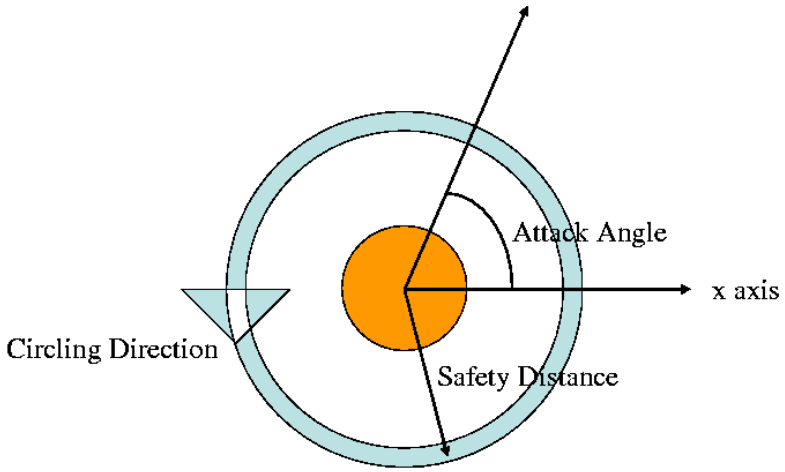


Figure 5.1: Get behind ball parameters

5.3 Get Behind Ball

The get behind ball is a defensive skill which is used to make the robot circle behind the ball, whilst facing it at all times. It is useful in situations when the robot is facing down field towards its own goal, and doesn't want to risk accidentally knocking the ball backwards whilst going for a grab. The maneuver isn't particularly quick, so it is mainly used when the ball is near the edges of the field where the robot may not have the room to walk right up to ball, especially if it is coming in at an angle diagonal to the wall. It is also used near the own goal where it is crucial that the robot doesn't accidentally knock the ball any closer to its own goal. In the other cases where the robot may be facing downfield, it simply risks walking directly to the ball where it then performs a 90 degree or 180 degree turn kick as appropriate.

The skill is parameterized with 3 values, the attack angle, safety distance and direction in which to circle around the ball. The attack angle is the heading that the robot wants to be facing at the end of the maneuver. For example, to move behind the ball and line up the target goal, this angle should be equal to the angle from the ball to the goal. The safety distance is the radius of the circle that the robot travels around, and the direction specifies whether the robot should travel clockwise or anticlockwise around the circle. Combined, this information tells the robot a target point that it wants to end up at, and a direction to travel to get there.

In order to make the robot walk on a circular path, there is a moving point,

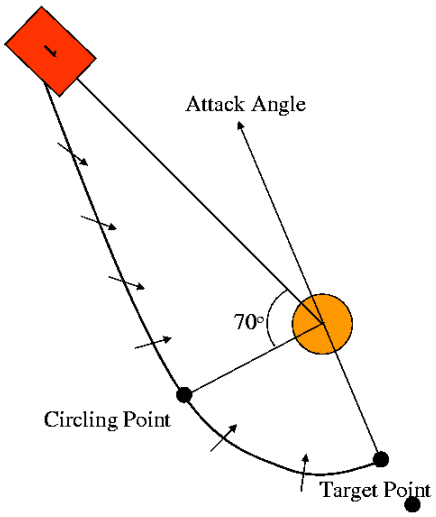


Figure 5.2: Circular path around the ball

called the circling point which the robot tries to walk to. Essentially this point defines a direction the robot should walk towards, and guides the robot around the circle until it reaches the target point. This moving point is defined as the point on the circumference of the circle, 70 degrees away from the vector from the ball to the robot in the direction that the robot wants to travel. As the robot moves, the point moves and the robot follows this point until it is closer to walk towards the target point. The result is a smooth circling motion that guides the robot around the ball.

Since all points and walk directions are recalculated on each camera frame, the get behind skill can be used to get behind a moving ball. In situations where there is an opponent pushing the ball downfield, the get behind ball can use used to follow the ball, walking sideways until the robot can swing defensively behind the ball. The move however is a little bit too slow and clumsy to be totally effective, and the robot often ends up getting into a scrum with the opponent robot before it can swing defensively around behind the ball. A future extension to the skill could be to relax the condition that the robot must face the ball as it circles behind it. Instead the robot could be walking forwards whilst turning its head to look at the ball. The robot is faster and more agile whilst it is walking forwards. Similar to the bird maneuver, it could then turn and swing around behind the ball, intercepting it and stopping the opponents attack.

5.4 Paw Kick

Paw Kick is redeveloped this year to take advantage of better close range ball recognition. It is also faster by moving the robot directly towards the ball, and lining up in the process rather than first line up, then move towards the ball. The advantage of paw kicks is in its speed. The robot continues to advance from line up to execution to follow through, neither the ball nor the robot is standing during any of the mentioned phases. It has virtually no setup and execution time, which makes it an ideal tool for fast break away and in retaining possession whilst moving aggressively forward. The goal of the game is to move the ball into the opponent's goal region, this cannot be achieved without ball possession, so the amount of time a team is in possession of the ball directly correlates with the chance it has of scoring, and possession is achieved by getting to the ball first. Our robots have an advantage in this area due to its walk speed, however once it has reached the ball, it must slow down to grab the ball and then to setup for the next shooting action. In doing this, we lose much time to opponents robots that could then catch up to hinder our shot, or to dispossess our ball. And much of the advantage due to our walk speed would then have been lost. By skipping over the ball grabbing phase, paw kick shortens the amount of time and distance traversed that is needed to gain possession, and by allowing the robot follow through without slow down, it means if our robot is infront, then there is no way for the opponent to catch up. And if we are approaching a defending robot, then the speed afforded by this skills gives us a chance at cutting straight pass them, where we would end up at a scrum had we chose to grab the ball instead.

There are two main disadvantages with paw kick. The first is that it lacks power, the current implementation does not take into account leg stance, and as such it cannot guarantee a full forced strike with each lunge. The second, and the bigger disadvantage lie in its inability to aim. We have experimented with several aiming methods, they fail or succeed at varying degrees, but none of them are reliable enough to be of use. However, this should not indicate that aim able paw kick is infeasible. Rather it should only serve to point out that there exists ample space for improvement over the current implementation. Currently, paw kick is still restricted for moving the ball forward, although it is reliable in doing this.

The first step in the execution of paw kick is to approach the ball such that the ball is lined up with the desired kicking paw. The user can assign a specific paw, or the paw can be decided automatically by assigning the paw that requires least change in the robot's direction. To avoid oscillation between choosing left and right paw, hysteresis involving ball position is used. The robot approaches the ball by aiming for an offset from the ball, such that the robot will collide the ball with its legs rather than its chest, as it would during a normal ball approach.

Since we control the point of collision between robot and ball via an offset, by changing the offset position, it is possible to gain some directional control by

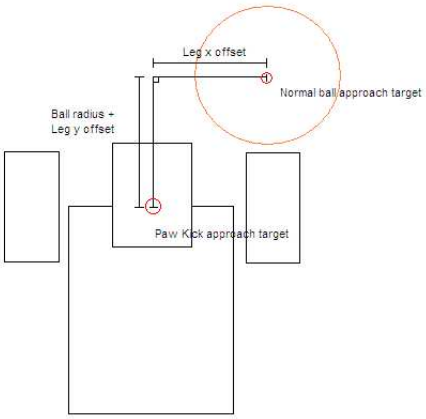


Figure 5.3: Difference in approach target between paw kick and ball grabbing

moving the offset away from the center. For example, in Fig 4.1 if we lengthen the x-offset, then the robot will strike the edges rather than the center of the ball with its paws, this would cause the ball to move forward and right at the same time. We have experimented this idea with limited success. It is possible to hit the ball forward with roughly 30 degrees freedom by changing the x-offset value. However, we were not able to obtain any reliability in the direction or execution of the kick, and there is a chance that the robot will miss the ball altogether. Another approach we experimented with was to center the ball between the robot's paws, but then execute a quick turn to flick the ball in the desired direction. This approach gave good speed and power when it is executed correctly, however it did not succeed often enough for it to be useful.

Once the ball is lined up and well within range for the forward lunge, the robot will commit to a charge at the ball at full speed. The charge is to prevent the robot from slow downs and hesitations in lining up, as ball distances will continue to fluctuate due to the bobbing of the camera as the robot moves. Ideally, we want to time the lunge such that it synchronizes with the forward stride of the striking ball, and we want to execute the lunge when the ball is in a position to receive maximum force from the paw. This is something that has not been implemented, and would greatly improve power in the kick.

5.5 Off Edge Kick

The off edge kick was developed to work with paw kick. Paw kick enables our robots to make very fast breaks down the side lines, however once the break has been achieved, they are unable to get the ball off the edge again, until the corners, in which ball handling is substantially more difficult, allowing other robots time to catch up, and our advantage is reduced. So we developed an off

edge kick that could pop the ball back into the field after a successful paw kick down the edges, before we come into contact with the corners.

Off edge kick is executed by walking the robot up to the ball such that the ball lie between the robot's leg and the boundary, the robot then side steps into the ball whilst turning away from the boundary at the same time to prevent the ball from rolling backwards. This has the effect of squeezing the ball onto the white boundary, and because the robot was turning away from the boundary, it adds a forward vector to the ball that causes it to roll back down the boundary before the robot. If executed correctly, the ball should now be in a prime position for the robot to grab and execute a kick at opponent goal.

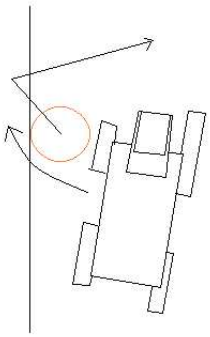


Figure 5.4: Off edge kick execution

There are two drawbacks that prevented this skill from being used in the competition. The first is the tight conditions in which the skill can be executed successfully. It is very demanding on the positioning of the ball as well as the robot relative to the ball. Whilst it is possible to gain reasonable position of the ball relative to the robot, it is difficult in obtaining the position of the ball relative to the edge as we do not use edge detection in competition play and so whether a ball is close to an edge is inferred entirely from the robot's localization. The second is in the difficulty in determining when the skill should be chosen over other skills. The off edge kick was not used in the competition.

5.6 Field Edge Spin

The intuitive ball approaching method for a robot is simply to get as close as possible to the ball, and to align the ball as close as possible to our center. This translates to move forward while turning to keep the ball in sight and in front at all times. And this is true for most situation. However there is one specific set up in which we would wish to turn away from the ball instead, and in doing so voluntarily lose sight and control of the ball.

The case is when we are facing up the field towards opponent goal, we are very close to an edge and the ball is trapped between our body and the field boundary. By turning into the ball as would the intuitive reaction, we would end up pushing the ball onto the boundary and down the edge behind us towards our own goal. This is most undesirable, and potentially disastrous if it occurs along the base boundaries of our own goal because that could cause own goals as well as offering an opportunity for opponent to score. And even if our robot manages to chase and recover the ball after, it would now be facing the wrong way, and dangerous to our goal if its localisation was wrong, or if it was being hassled by nearby opponents.

A possible solution is for the robot to simply back off in that situation until it was in a position to grab the ball properly. The draw back is that by withdrawing from the ball, it allows room through which opponents can break through to not only steal the ball but to pass through our defence.

The second solution is for the robot to perform a stationary spin away from the edge. By spinning in the opposite direction of the ball, it causes the robot to kick the ball down the line in the direction of opponent goal with its back legs or body as it spins. Another advantage is that our robot does not concede any opening as it remains stationary blocking off one of the edges as it spins.

The first method is better if there are no nearby robots, as it would retain possession of the ball, however robot detection is unreliable and in an unpredictable environment, we wish to make decisions based not on the decision that reaps greatest reward should it succeeds, but the decision that is least disastrous should it fail. Successful execution of our first solution will retain possession of the ball, whilst a failed execution could have an opponent robot break through our ranks with a ball. Our second solution will lose the ball by moving it some distance towards the opponent goal, and guarantees not to concede space. So it is the second solution that we adopted for it is more robust to different gaming conditions.

Execution of the spin is simple in that it enters a brief lock to tear itself away from the ball, then it drops into the find ball routine which continues the spin, and any signs of a ball will bring the robot out of the spin to chase it down.

5.7 Goalie Out of Goal Spin

When the goalie for whatever reason ends up facing its own goal, it is very disastrous for competition play, and we would want it to come out of that position as soon as possible. However it is not possible to rely on standard behaviors for rectification, because by being in that position it would mean its localization was already off.

It is possible to detect this event based on visual cues. If we see our own goal, then that would indicate already that we are not facing outwards. However when our goalie is defending its goal by pressing itself against one of the corners that join the goal to the white boundary, we can sometimes catch a portion of the inner side walls of our goal. So we needed to be careful in selecting the criteria that defines when the goal is large enough to mean we are now facing into the goal, and we use a combination of width, height and area.

Once triggered, we can then command the robot to start turning. A very simple approach that we first tried was to have the robot turn until the condition that triggers this actions is no longer true, in other words when we are no longer seeing a large slab of own goal before us. However, shadows on the depths of a goal can decrease the size and density of the goal coloured blob. Another problem is that as the robot attempt to turn around inside the goal, the robot's camera can bump onto the inner walls of the goal, which severely disrupts vision because sufficient light fails to enter the camera when the camera is very close to the goal walls. And so by relying on visual cues to maintain the turn, we often find the robot stuck onto one of the goal walls. We got around this problem using a lock, which although manages to get the robot out of the goal, any sort of lock is undesirable because it means the robot is unresponsive to changes in the game whilst the lock is in effect. Thus neither of them were valid solutions.

We eventually settled on the infra red distance finder that is installed on the snout of the robots. This feature was not used for other purposes because it required very accurate aiming at objects, and its measurements prone to error over anything but short distances. Since the wall of the goal is so wide and tall, we can ensure that the infra red sensor will always be on the goal. Furthermore, there will be a very noticeable change in the readings between goal and non goal objects because the goal is much closer to the robot.

To exaggerate this difference, we position the robot's camera at a tilt such that it is able to see the goal if it was infront of it, but the sensor would pass over the top of all objects on the competition field once it has successfully turned away from the goal. Direction of spin is based on where the robot had last seen the ball, and the robot will continue to spin until either the infrared sensors cease to sense any obstacles, or the robot catches glimpses of landmarks. This has the desirable effect in that it will stop at one of the defensive corners of the goal, and has proven to be a reliable method at preventing the goalie from being stuck in its own goal.

5.8 Dribble

5.8.1 Rationale

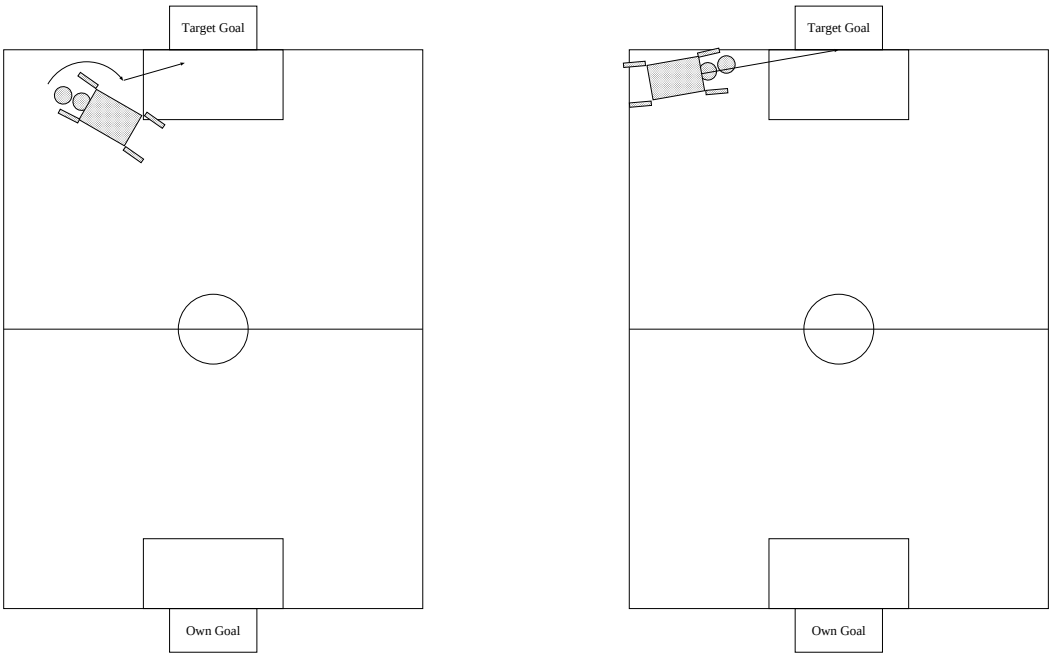
In previous rUNSWift teams, the chest-push was used whenever the robot had possession of the ball close to the target goal. The rationale for the chest-push was that if the chest-push failed in scoring, the forward is still able to recover possession as the ball would have only travelled a short distance from the chest-push.

However, this year's rUNSWift has developed a form of dribble that has replaced the chest-push. The dribble was found to allow the robot to maintain possession of the ball as well as become even more aggressive in its attack. Instead of the forward pausing and merely pushing the ball forward with its chest, in the dribble, the forward holds the ball very loosely and walks into it at maximum speed. The paws are positioned to guide the ball forward, that is, to prevent the ball from straying to the side. The head of the robot is lifted up from the ball grab so that the ball can move freely but still under the control of the robot.

Because the forward is charging forward at maximum speed, this form of attack was found to be quite effective during the Robocup 2003 competition. Like the chest-push, the forward can still recover balls that have not gone into the goal, because it is moving at a maximum speed towards where the ball is travelling. Unlike the chestpush, the attack is more aggressive, because the ball moves faster, has further range, and the robot does not need to incur the cost of post-chestpush recovery time.

5.8.2 Skill

The dribble was the main form of scoring for rUNSWift at the 2003 Robocup competition. In order to use the dribble to score, the forward spins with the ball underneath its chin until it is facing the target goal before it dribbles forward, as depicted in Figure 5.5a. The robot needs to keep its chin on the ball while spinning to keep it in place; thus, it is unable to use visual information to know when to stop spinning because it cannot see anything but the ball. Hence, the amount to spin is precomputed before spinning starts, based on the robot's global heading prior to spinning.



(a) (b)

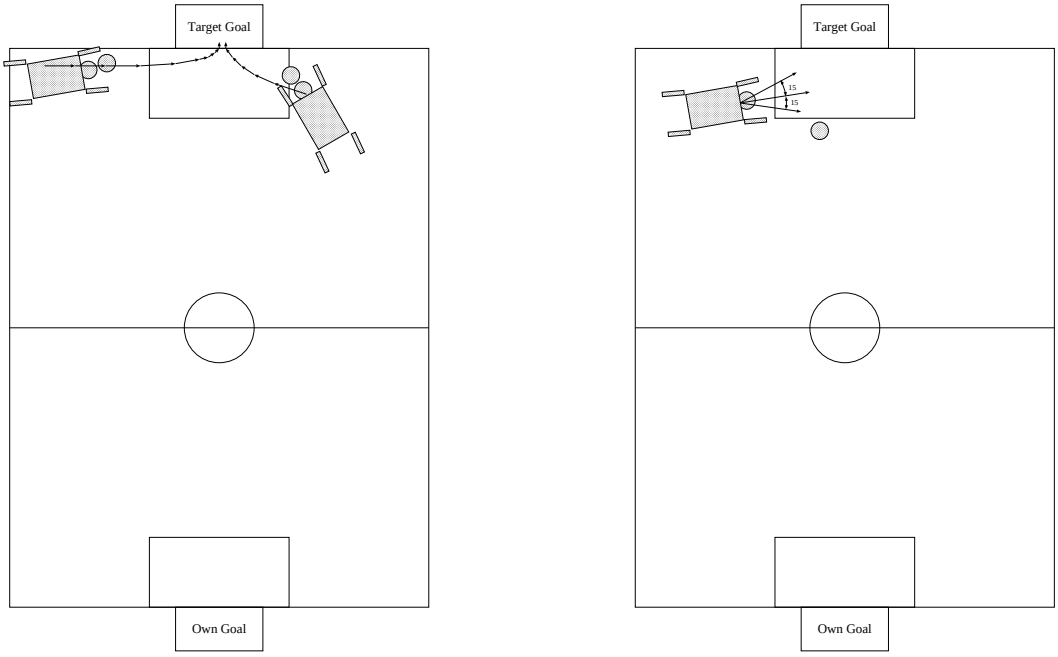
Figure 5.5: (a) The forward spins before dribbling towards the goal. (b) The forward's left paw can get caught if it dribbles directly towards the goal.

Therefore, this method relies heavily on having an accurate global heading before the spin. Otherwise, the robot would look up after the spin and discover that it is not facing the target goal, and then abort the dribble. Because of fierce competition for the ball, the robot is likely to get pushed around a lot and its estimate of its global heading may be far from accurate. As a result, before spinning, once it has grabbed the ball, the robot uses the smart form of active localisation to discern its bearings accurately.

Dribbling directly towards the centre of the target goal can result in the robot's paws getting caught on the edge as shown in Figure 5.5b. As a result, the exact direction in which the forward dribbles the ball is determined by a vector in the vector field shown in Figure 5.6c.

The following is a more comprehensive list of steps that a forward undergoes when it executes a dribble:

- It holds the ball for a sufficient number of frames to ensure that the ball will not pop out;
- It performs smart active localise with the ball still in its grasp;
- It moves its head back down and spins until its global heading, obtained from GPS, is between its heading to the two goalposts;
- It looks up to see if the target goal is in view. Looking up also allows the ball to move freely when it moves forward;
- If the target goal is not in view, possibly because it is obstructed by opponent robots, it determines the direction in which to dribble based on its position in the vector field.
- It dribbles in the direction of the vector until it cannot see the ball or if it can see the ball, its heading to the ball is too great, i.e. the ball has veered to the side, and charging in the direction of the vector will not hit the ball forward, as depicted in Figure 5.6d.



(c) (d)

Figure 5.6: (c) The vector field used to determine the direction in which the forward dribbles. (d) When the forward senses the ball has veered to the side, the forward aborts the dribble.

5.9 The Turn Kick Series

5.9.1 Motivation and Description

As the name suggests, the turn kick is a kick that can propel the ball 45, 90 or 180 degrees to either side or back by literally turning into the ball after it is trapped between the robots front paws.

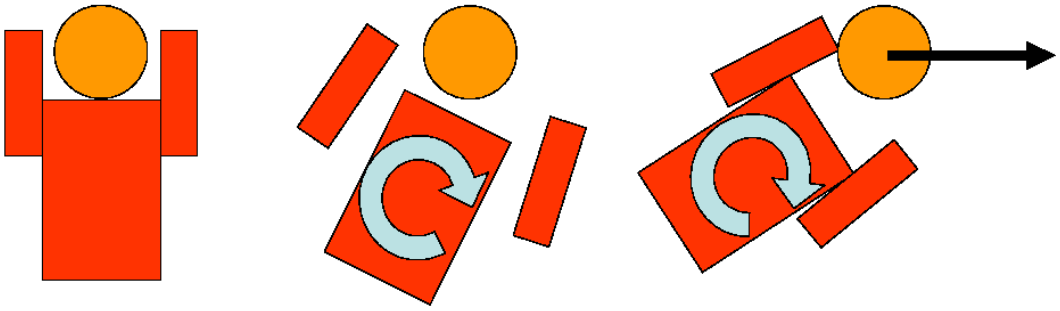


Figure 5.7: The execution of a turn kick. The robot first traps the ball under its head. It then takes a step into the turn, opening up its legs and allowing the ball to roll out slightly. As it takes another step, it kicks the ball.

The emphasis on the turn kick has always been simplicity and speed. Since the maneuver only uses the walk parameters, ie to tell the robot to turn, the kick requires very minimal recovery time. Unlike the front kick or the lightning kick, the robot doesnt́ need to lie down to perform the kick and thus doesnt́ need to stand back up after the kick. Since the robot stays in a walking stance at all times, the turn kick has the advantage that the robot can immediately continue to chase the ball after it has kicked it. This philosophy is similar to that for the paw kick. Both types of kicks are speedy, "recovery-less" kicks. The paw kick however has a considerable advantage time wise, in that it doesnt́ need to first grab the ball.

Previous versions of the turn kick very simply only involved ordering the robot to turn once it had the ball. This year, to improve the reliability and power of the turn kick, the timing of its execution was experimentally calibrated very precisely by investigating the walk, and the theory behind how it works. By doing so, the robot can now consistently punch the ball out from between its paws rather than fling it out as the old method did. This however comes at the cost of additional setup time, although the time taken to grab the ball is still the main delay in the execution of the turn kick.

5.9.2 How it Works

The key to the reliability of the turn kick is being able to hit the ball in the same way every time. This is done by first grabbing the ball so that the ball starts off in the same position relative to the robot. With the ball trapped under the robots head, the robot starts to turn in the direction it wants to kick the ball. At some point in the turn, the robot lifts its head, releasing the ball which then starts rolling out from between the robots paws due to the centrifugal force applied on the ball from the spinning robot. This needs to be timed so that the ball rolls out the right amount for the robot to punch it with its front paw as it turns. The timing is achieved by synchronizing the time in which the robot starts lifting its head with the legs being in some specific point on the walk locus.

When a robot is turning on the spot, the front paws trace the pattern depicted in 5.8.

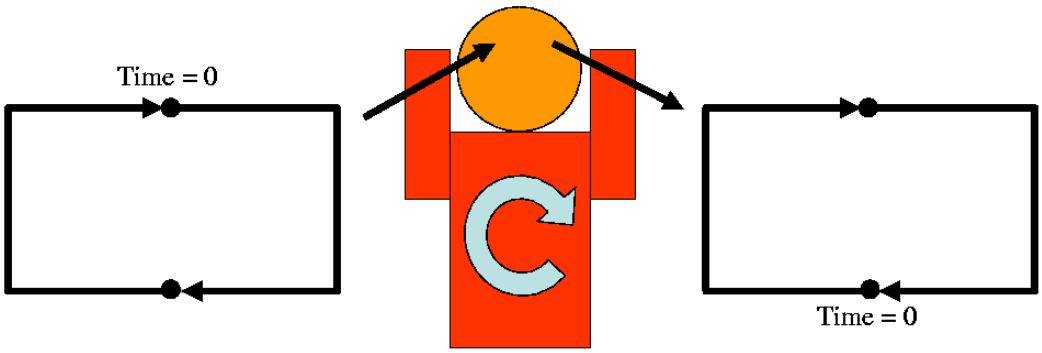


Figure 5.8: Diagram showing orientation of turn locus. Both paws circle in the same direction however are 180 degrees out of phase.

Both paws spin in the same direction, but are out of phase. That is, whilst one paw is at the top of the locus, the other paw is at the bottom of the locus. In order to make the robot turn in one direction, the loci are orientated slightly diagonal to the robots body.

The leg lift of the front paws is very small, as most of the turning power is generated by the hind legs, and by alternating the side on which the robots weight is resting. Thus, the rectangular locus shown above is exaggerated. The front legs look more like they are slightly diagonally opening and closing, moving together and then apart as the robot turns. In order to perform the kick, we want to time the release of the ball, so that it comes in contact with the appropriate paw as they close up, ie move together. If the robot wants to

kick to the left, then it spins to the left and punches the ball with its right paw. If the robot wants to kick to the right, it spins to the right and punches it with its left paw.

5.9.3 Locus Based Description

Let us now elaborate on the explanation above by showing exactly what the timing is with references to walk loci, using the 90 degree right turn kick as an example.

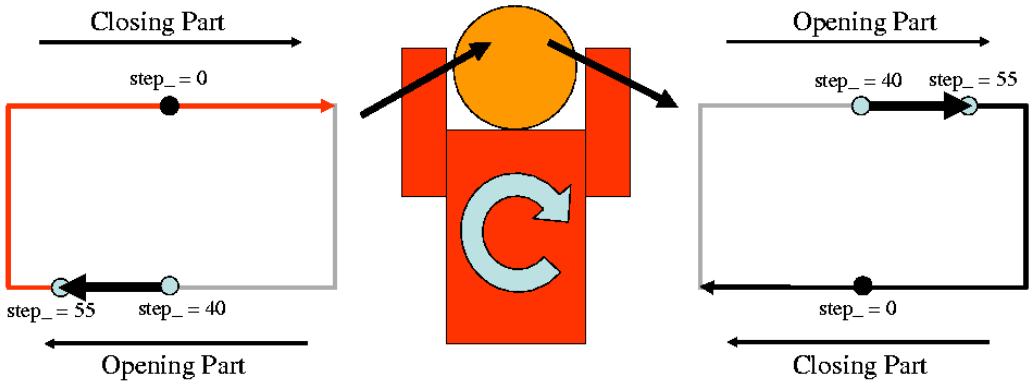


Figure 5.9: 90 degree right turnkick. The blue dots show critical points in the execution of the turn kick. The thick black arrows indicates points where the robot is turning. The thin red arrow indicates the follow through of the kicking leg. The thin black arrow inidicates the follow through of the non kicking leg.

Fig 5.9 shows loci for a robot turning to the right. Step_ is a variable whose value represents what point on the loci the legs are at. The turn kick has only been calibrated for a PG value of 40 (see locomotion section) which means step_ is a value between 0 and 80. Remember, you can only change walk direction on a half step. For a PG value of 40, the half step points are at 0 (or 80) and 40.

In order to perform a 90 degrees right turn kick, the robot needs to start turning to the right at step_ = 40 so that the first step is on where the legs are opening up with the left leg on the ground. If the robot doesn't have its paws at step_ = 40 at the start of the kick, it continues walking forward until it can start to turn. At step_ = 55 the robot raises its head so that the ball can start rolling out as it continues to open up its front legs. The robot then follows through with the turn, punching the ball as the legs move through the closing part of the loci. When step_ gets back to 40, the kick is complete.

If the ball is released too early, it rolls out too far and so when the left paw swings across to punch the ball it completely misses it, or only just clips it. If the ball is released too late, it doesnt́ roll out far enough so and when the left leg swings across to punch the ball it hits the ball with the inner part of the leg rather than the paw, causing the ball to rebound off the right leg and get trapped between the legs or not get propelled very far. Figure 5.10 illustrates this.

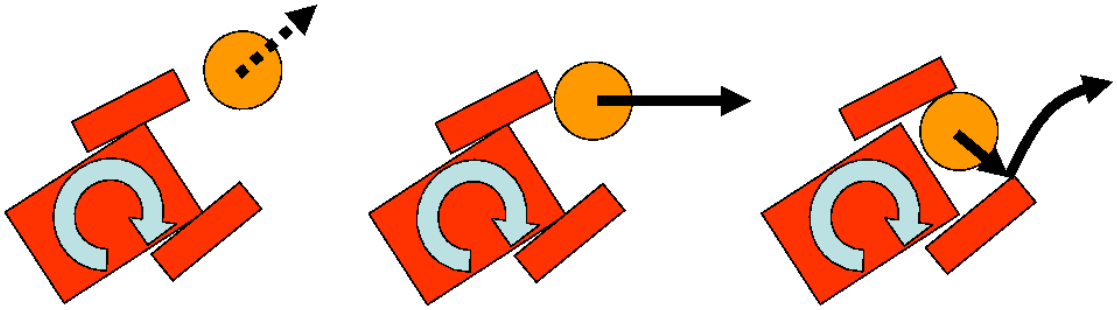


Figure 5.10: First robot released ball too early. The last robot released the ball too late. The middle robot released the ball at just the right time.

In order to perform a 180 degree right turn kick, the legs simply go right around the turn locus once first before executing the steps described above. Figure 5.11 illustrates this. When performing a 180 degree turn kick however, the robot doesnt́ need to keep walking forward until the legs are in the right position to start turning. Since there is so much movement before the critical part of the kick anyway, there is no need to waste time waiting for the legs to be in the right position before it starts turning.

The 90 degree left turn kick is very similar to the 90 degree right turn kick, but timing wise, it isnt́ quite a mirror image of the right turn kick. This is due to asymmetries in the robots body. That is, the battery sits to one side of the robot, making the robot heavier on one side.

In the left 90 degree turn kick, the ball is actually released slightly before the robot starts turning, i.e. whilst it is still waking forward. Again the first step in the turn is one where the legs are opening up. The ball then rolls out to be in the right position when the right leg swings around the closing part of the locus to punch the ball. Again, to perform the 180 degree left turn kick, the robot simply goes around the turn locus twice, kicking the ball on the second revolution around the locus.

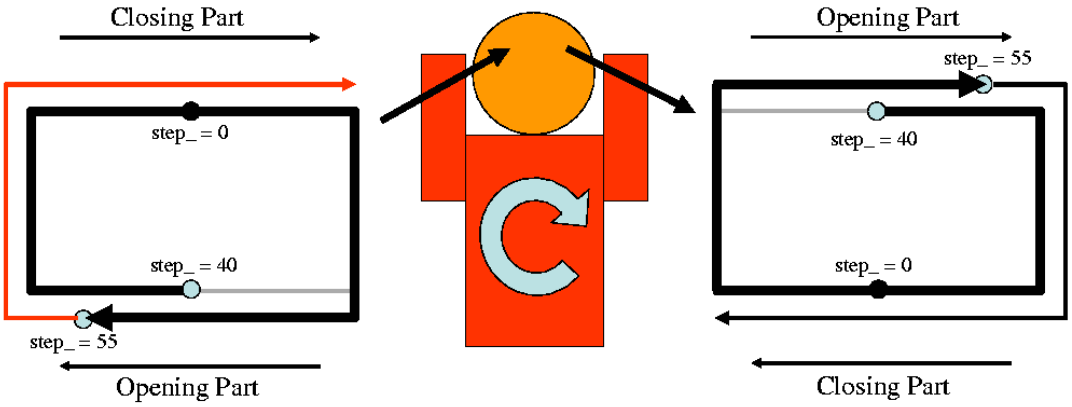


Figure 5.11: 180 degree right turnkick. The blue dots show critical points in the execution of the turn kick. The thick black arrows indicates points where the robot is turning. The thin red arrow indicates the follow through of the kicking leg. The thin black arrow inidicates the follow through of the non kicking leg.

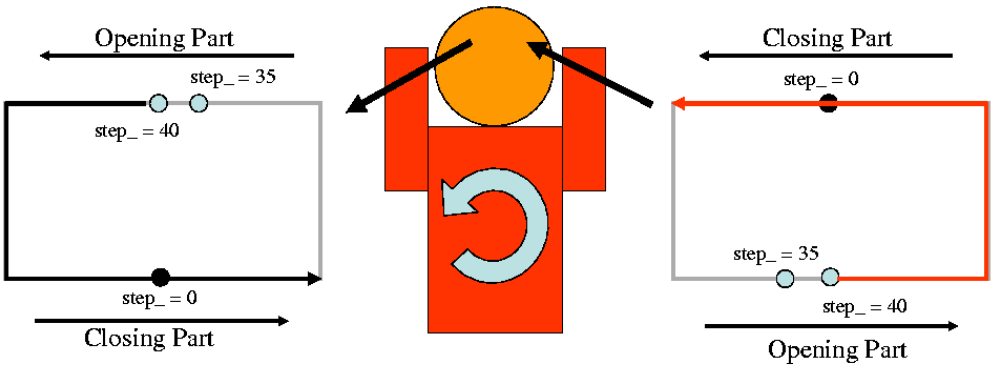


Figure 5.12: 90 degree left turnkick. The blue dots show critical points in the execution of the turn kick. The thick black arrows indicates points where the robot is turning. The thin red arrow indicates the follow through of the kicking leg. The thin black arrow inidicates the follow through of the non kicking leg.

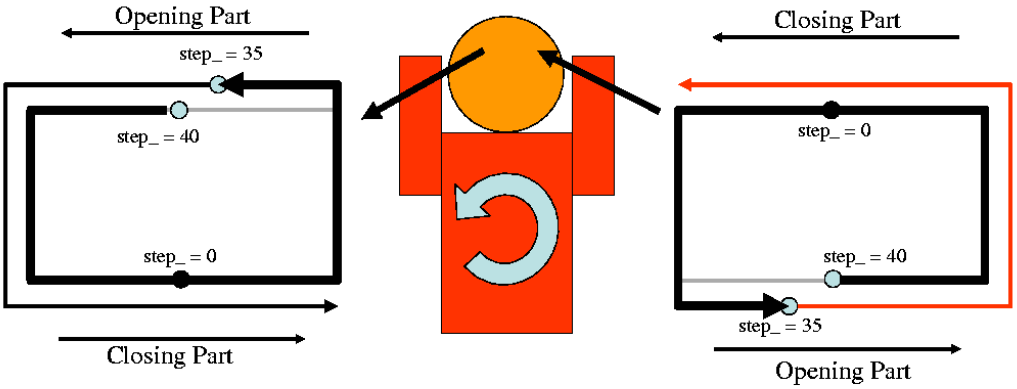


Figure 5.13: 180 degree left turnkick. The blue dots show critical points in the execution of the turn kick. The thick black arrows indicates points where the robot is turning. The thin red arrow indicates the follow through of the kicking leg. The thin black arrow inidicates the follow through of the non kicking leg.

Since the ball can only be released on specific parts of the turn locus, there are only certain angles at which you can kick the ball. These angles are a function of the turn speed. The robot currently is using the maximum turn speed to execute the turn kick. Theoretically you could reduce the turn speed in order to get other angles. This was experimented with briefly, but problems arose because not enough centrifugal force was generated to make the ball roll out. Instead another method was used.

5.9.4 Varying Forward Reach

In the turn kick action described above, the front legs of the robot are actually reaching out more than then do in the normal turning action. By using that same turn kick with reductions in the amount the robot reached forward with its front paws, we were able to make the robot perform 45 degree turn kicks.

With the paws slightly closer to the body, the paw only clips the lower part of the ball as it swings around the closing part of the locus. This causes the ball to be propelled at 45 degrees. Theoretically the same result could have been achieved by releasing the ball slightly earlier so it rolls out slightly further, but there is finer control on the forward reach of the paws than on the timing of the ball release.

Since the ball doesn't get punched as solidly as it does in the 90 degree turn kick, it doesn't travel as far or as fast. This however isn't necessarily bad. It would be useful in evading opponents where you just want to pass the ball diagonally forward and away, where your teammate can take over the attack. The turn kick is way too slow to be used in that situation though, because you need to grab the ball first. Instead, this move is something you would want to be able to do with the paw kick. The other shortcoming of the 45 degree turn kick is that it needs room to execute. The robot needs to allow the ball to roll in front of it where it can clip it with its front paw. If there is any kind of obstruction, such as from the opponent robot it is trying to evade, the kick won't come of at 45 degrees if at all. It is the type of kick that works in the open but not necessarily in the game and thus it wasn't used in our final strategy.

5.9.5 Edge Turn Kick Aka Locate Ball Kick

Since the turn kick relies on rolling the ball out in front of the robot where it can cleanly punch it with its paws, it is not very effective when the ball is up against the walls. In response, a variation of the turn kick was developed to be used when the ball is near or on the walls. It is similar to the old style of turn kick as it involves flinging the ball out from between the robots paws rather than cleanly punching it out. The difference however is that the robot follows through with the kick by spinning into the locate ball routine. The rationale behind this is to force the ball through scrums against the walls, but this will be further elaborated below.

The fling style turn kick is used against the walls as it doesn't require the ball to roll out in front of it to work. In fact, being against the wall actually assists the kick. The robot wedges the ball between its chest and the wall and turns, using the wall like a direction guiding rail for the ball to travel along. There is no specific timing of the legs that needs to be adhered to. The robot simply turns for a fixed amount of time, i.e. enough to do a 90 degree kick and then enters the locate ball routine. The robot however doesn't start from the beginning of the locate ball routine. It jumps straight up to the part of the routine that has the robot spinning quickly on the spot.

During a game of robot soccer, the play frequently ends up on the side edges. Robots are thus often in scrums along the side edges fighting for possession of the ball. If there is an opponent trying to get the ball from the side that you trying to kick it, the turn kick will simply wedge the ball up against the opponent. By following through with the fast part of the locate ball routine, the robot continues to spin into the ball, using all 4 of its legs as paddles spinning into the ball. If the robot is spinning to the right for example, it is usually the right hind leg that is able to the hit the ball so that it can roll up the slanted wall and past the opposition robot. The headswipe, which is a kick where the robot uses its head to hit the ball to the side, is also effective and hitting the ball up and around opponents on the side walls. The headswipe however isn't used in

the current strategy.

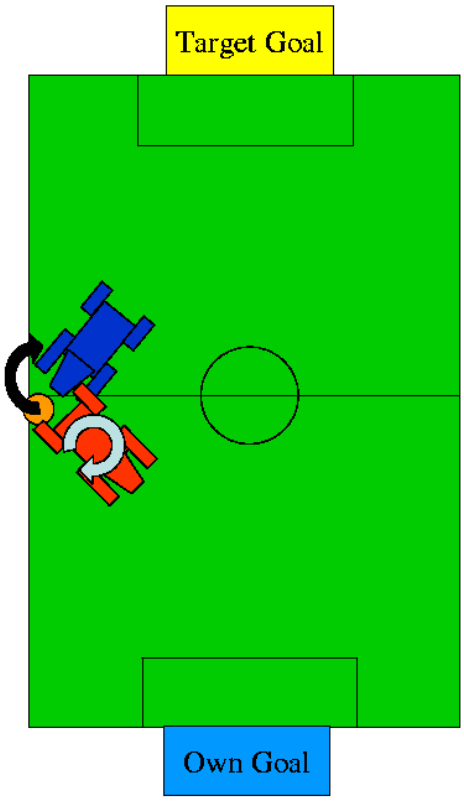


Figure 5.14: Locate ball kick showing back leg kicking the ball

In cases where there is no opposition robot obstructing the edge turn kick, the robot is usually able to see the ball rolling away. Since the robot has found the ball, the locate ball routine ends straight away before it wastes time spinning around in a circle.

5.9.6 Lessons Learnt in the Development of the Turn Kick

When we first started trying to improve the reliability of the turn kick, we tried improving it as a fling. The previous version worked sometimes and didnt others. We knew that to improve the reliability, we first needed to have the ball start in the same position relative to the robot each time. It was hypothesized that if the robot moved forward whilst it turned, the ball would consistently get trapped in the corner between the chest and one of the arms. It would then roll forward along the arm of the robot and get flicked out. However, this didn't

make a difference to the reliability.

It was soon observed that the turn kick was most effective when the ball didn't get flicked out, but rolled forward and got punched out cleanly. We then tried having the robot walk backwards as it turned so that it would move away from the ball where it could punch it cleanly. There was still no changed to the reliability.

It was then hypothesized that the turn kick only worked sometimes because the legs only sometimes started in the correct position, and that the forward/backwards vector had nothing to do with it. After investigating the theory behind the walk loci to justify the hypothesis, experimental calibrations were performed which finally led to noticeable improvements in the reliability of the kick.

One peculiarity found during the calibration is that the reliability of the turn kick reduces when the robot heats up. If you make the robot do consecutive turn kicks, it starts hitting them less cleaning. The effect however is temporary. If you let the robot cool down for a while, the turn kicks become reliable again.

A problem with the turn kick is that it can be stopped if the robot doesn't have room to turn. Being obstructed is a problem with all the kicks, but with the turn kick, obstructions from the back can still inhibit the kick. For the 180 degree turn kicks, not much can be done about this problem. But for the 90 degree turn kicks, a new kick could be developed that didn't require the robot to turn its body. Instead the kick could involve the robot swinging one of its front arms across to hit the ball whilst the body remained pointing forward. Several of the other Robocup teams have such a kick. The University of Pennsylvania team for example has a very powerful kick of this style. A kick of this style however is likely to have an increased recovery time. It would then be left for the strategy to decide when to use each type of kick.

5.10 Visual Opponent Avoidance Kick

5.10.1 Introduction

The Visual Opponent Avoidance Kick (VOAK) was developed for accurate scoring by rUNSWift forwards. In employing VOAK, forwards are able to aim accurately at gaps that exist between opponent robots and rUNSWift's target goal, thereby away from opponent robots. The accuracy to shoot away from opponent robots is achieved by basing decisions such as how much to turn on the bounding boxes of the target goal and obstructing opponent robots that are obtained from the vision module in the current camera frame.

5.10.2 Tradeoff between accuracy and speed

The accuracy that can be achieved by the VOAK is quite remarkable; not only does it avoid opponent robots but also fellow rUNSWift robots if they so happen to be blocking the target goal. However, there is a trade-off between accuracy and speed, which is why in the main attacker strategy, the VOAK is triggered only when the forward is certain that it is in the clear. In the event that opponent robots are nearby, the VOAK becomes ineffective. This is because by the time the forward has found a suitable gap to aim for and has turned to aim at this gap, before the robot has a chance to kick the ball, opponent robots in front of the forward are likely to have moved close enough to the forward to close off any gaps, while opponent robots behind are likely to be pushing against the forward, thus throwing its heading off.

5.10.3 Skill

Prior to performing the VOAK, the forward checks its global heading to determine whether it lies between its heading to the two target goal posts. In the case where its global heading does not lie within these two limits, the forward spins in the direction of the centre of the target until it determines that its heading does lie between these points.

At the start of the VOAK, the forward holds the ball for a sufficient number of frames to ensure that the ball does not come out of its grasp. It then looks up, expecting to see the target goal. In the event that it does see the ball after it has looked up, the forward realises that it has lost possession of the ball, aborts

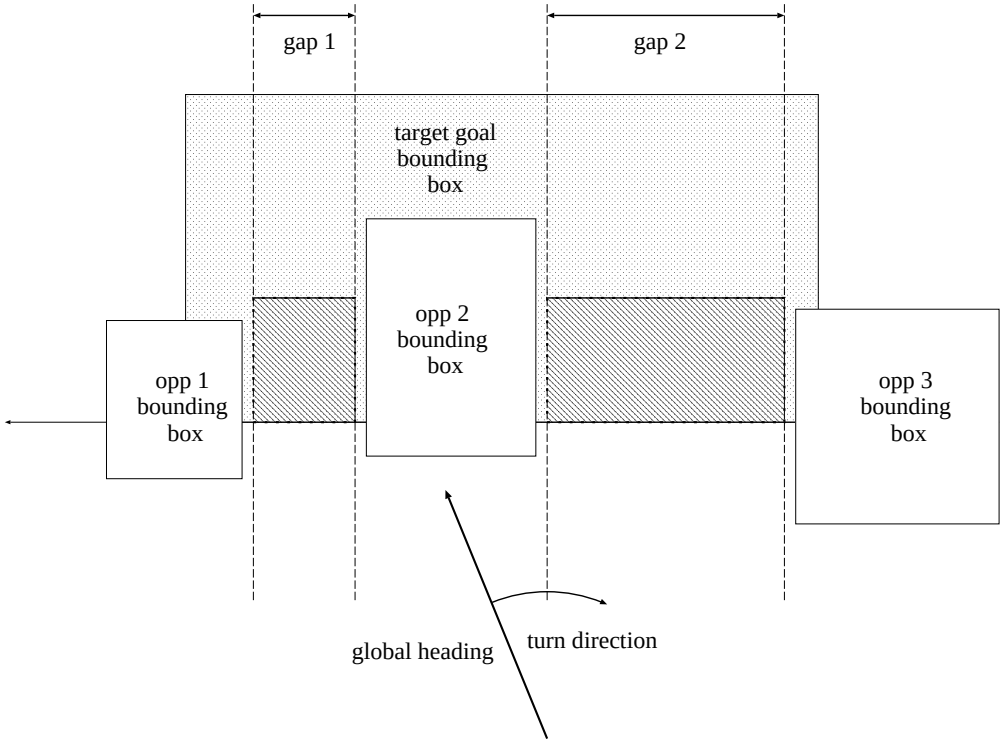


Figure 5.15: The largest gap is selected for shooting.

the VOAK and gives chase.

If the target goal is in its field of view, the forward computes all the gaps that are created by blocking robots standing in front of the target goal. The headings to each gap's left and right boundaries are calculated, based on the bounding boxes of the target goal and the blocking robots obtained from the vision module. The largest gap is then selected to aim at, as depicted in Figure 5.15.

The forward then checks its global heading to determine whether it lies within a safety margin of the selected gap. As shown in Figure 5.16, this safety margin is the angle between its heading to left edge of the gap plus an offset, and its heading to the right edge less an offset. When the forward's heading falls within this safety margin, it performs a forward kick. Otherwise, the forward carries the ball and takes small steps to turn towards the chosen gap.

Because of the three second ball-holding rule, the forward starts a timer

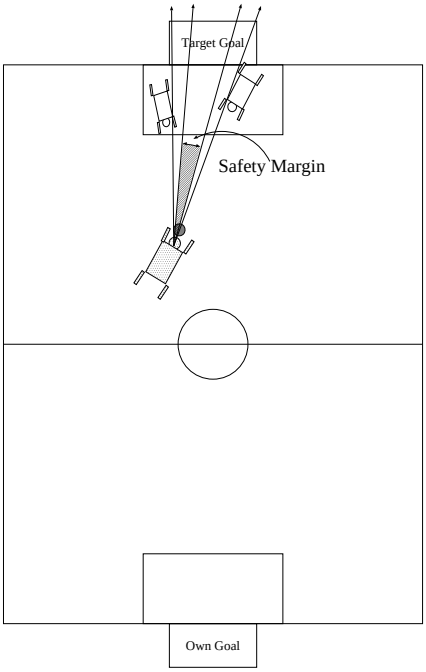


Figure 5.16: Once the forward's global heading is within the safety margin of the selected gap, a front kick is performed.

upon grabbing the ball and when that timer expires, the forward breaks out of the VOAK to execute a forward kick.

In the event that the forward does not see the target goal after it has completed its fast spin and looked up, it then continues to spin with the ball towards the GPS target goal until it either sees the target goal, or the timer expires.

Determining the largest gap can be quite difficult. It is not sufficient to use the width of the gaps calculated from the bounding boxes to determine the most suitable gap for which to aim. This is particularly true when the forward is positioned in the top field corners and the width of the target goal is quite small, because the sizes of the bounding boxes can be quite misleading. Instead, the number of target-goal coloured pixels that are present in the lower half of each gap in the current camera frame is used as an estimate of the size of the gap. Because the ball rolls along the ground and does not fly through the air during the front kick, the number of pixels of target goal colour in the higher region of each gap is irrelevant and hence not taken into account in determining the most suitable gap.

When two or more gaps are found to be of comparable sizes, it is possible for the forward to remain undecided. For example, the forward might first turn towards the left-most gap, but upon turning, a gap to the right might become slightly larger. As a result, the forward stops and starts turning to its right. In order to solve this problem, before the forward starts turning, if the same gap remains the largest for a sufficient large number of frames for the forward to be sure that it is indeed a suitable gap, the forward sticks to that gap and turns towards it, ignoring the other gaps.

5.11 Hover To Ball

Hover to ball is the main function used to move the robot towards the ball in the game. It links together the many skills used when following the ball to form a compound skill that enables the robot to best chase the ball in any situation.

It first involves a check to see if the robot is chasing the field down the edge. Since the ball is not as wide as the robot, if the robot attempts to move towards the center of the ball whilst beside an edge the feet continually make contact with the edge greatly slowing the robot's progress. To prevent this from occuring the robot detects when it is very close to the edge, and following the ball at a medium to long distance, and procedes parallel to the edge in order to avoid contact.

The Hover to Ball function also manages the stealth dog and velocity prediction skills. If stealth dog is activated the stealth dog skill takes control of the robot. The velocity prediction skill does not take control of the robot, but rather shifts the heading that the robot desires to take. Both these skills are described further in the paper. If velocity prediction or normal hover mode are required the robot attempts to move towards a particular heading.

The Hover to Ball function is also used to best maintain the speed of the robot in the chase for the ball, whilst using the more stable canter walk and cautious movements when extremely close to the ball to enable a cleaner grab. Given a desired direction the robot calculates the turn factor it should send to the locomotion module. The robot attempts to use the maximum speed possible (offset walk) by considering the desired turn factor and the distance to the ball: -

- If the turn is less than 13.75 and ball distance is greater than 30 cm it is assumed the robot may use the fastest walk possible and take a curved path towards the ball
- If the turn is less than 22 and the ball distance greater than 60 cm it is also assumed the robot can take a curved path and maintain full speed. The turn factor is also set to a maximum of 13.75 to prevent slipping in the walk. The extra distance to the ball should provide time for the robot to complete the turn towards it.
- If the turn factor is large compared to the distance to the ball then the more stable but slower canter walk is used to allow a greater turning factor. The forward and left components are lowered to values hand calibrated such that slipping is avoided. Table 5.1 shows the values the forward and left (absolute) components are limited to. As can be seen, it was found that turning clockwise and anticlockwise produced different results in stability and slipping. This is due to asymmetries in the robots body. (Namely the battery sits on one side of the body).

TurnCCW t	Max Forward	Max Left
$0 \leq t < 10$	5.5	5.5
$10 \leq t < 20$	4.5	4.5
$20 \leq t$	3	2
$0 > t > -10$	5	5
$-10 > t$	1	3

Table 5.1: Hover parameters specifying maximum forward and left components given a turn component

The final stage of the Hover to Ball function slows the robot down when it is very close to the ball to enable it to more easily grab it. This is performed relative to the direction the robot is facing. If it is facing up the field, the distance to the ball must be less than 15cm before the robot starts slowing down. If it is facing down the field (towards its own goal) the ball must be less than 20cm. The difference is due to the fact that a clean grab of the ball is much more important when facing the defensive direction, since fumbling the ball leads to pushing it in a disadvantageous direction. When moving up field, fumbling is not as undesirable, since we are still moving the ball in the right general direction. When the slow down is enabled, the forward value is set to a maximum of 7.5 cms per half step. If the heading to the ball is greater than 15 degrees, this is further lowered to 0.5 to account for the large amount of angular movement that is required in a short space. If the robot moves forwards whilst it turns, it could accidentally knock the ball forwards with its paw. If a large turn is needed the PG value is reduced to 30. This allows the robot send direction changes to the locomotion module more frequently as step completes will occur more frequently. This is advantageous as modifications in the path may be required and will become important in order to grab the ball.

One thing that needs to be noted is that the hover parameters listed above are biased towards moving forwards. The parameters specify maximum forward and left components, based only on the turn value. Ideally the maximum value for one component should be a function of the required movement in the other two directions. For example, if a robot was turning at a rate of 20 degrees, with no forward movement, the left component should not be restricted to a value of 2. In fact, it was experimentally found that even when turning at a rate of 20 degrees, the robot is still able to walk sideways at maximum speed. The hover parameters thus need to be more completely defined in the future. The ball grab also needs to be improved in terms of reliability and speed. It is currently the weakest link in rUNSWift's attack strategy. That is why skills such as the paw kick which avoid grabbing the ball altogether were developed.

5.12 Velocity Prediction

Velocity tracking was introduced into the rUNSWift localisation system in 2003. The purpose of tracking the velocity of the ball is to enable robots to anticipate the ball's future position and react relative to this, rather than assuming the ball is still. An example of this is a scenario of the robot chasing the ball when it is travelling towards the robot at a significant speed. If the robot does not anticipate the future position of the ball it will move directly towards the ball, the ball may then travel quickly past either side of the robot. A more appropriate action the robot could have taken is to anticipate the ball's path past him and move perpendicular to the ball's path in order to stop it.

Velocity prediction is the skill that utilises the ball velocity tracking system in order to act in the most appropriate way in circumstances similar to above. It detects circumstances when a ball's current velocity is above a threshold, and calculates the position it can best intercept it. The skill is implemented in the general ball chase function of the rUNSWift forward (currently doBeckhamHoverToBall()). It consists of conditions when velocity prediction should be used and an algorithm for determining the movement to perform.

5.12.1 Conditions for Velocity Prediction

Velocity prediction should be used cautiously since velocity tracking is not entirely reliable and unnecessarily disturbing the robot's path to the ball could be detrimental to the speed at which control of the ball is obtained. The conditions for the activation of velocity prediction are designed to avoid using erroneous velocity estimations and from performing velocity prediction in undesirable circumstances. The conditions consist of: -

- The ball must be away from the edge of the field. If the ball is travelling near the edge of the field the velocity may not be sustained since the ball may come in contact with the edge. rUNSWift has deliberate behaviour when dealing with the edge of the field, this is performed before velocitty prediction and hence the condition is not directly checked.
- The velocity estimation must be accurate. In order for the velocity estimation to be predictable the robot must be able to see the ball, otherwise it has not been receiving observations and the estimation is ignored. The velocity estimation is significantly unreliable, so the velocity tracking system maintains an innovation vector value for the velocity that is an exponentially decaying average of the filter innovation vector lengths (changes to the velocity due to observations). The innovation vector measure divided by the length of the velocity estimation is checked, if it is long the velocity estimation has recently been greatly effected by observations and is

hence not believed, if it is short the estimation has not been through great change and hence represents a sustained velocity estimation that can be believed.

- Using the velocity must be important relative to the extra complexity applied. The velocity prediction is a complicated algorithm and should only be used when it will make a large difference to the decision of the robot. The speed of the ball (length of the velocity vector) must be greater than 0.3 centimetres per second or the ball can be assumed still. The distance to the ball is also checked, if it is less than 20cm away it is assumed moving straight towards it is the best course of action since the time to obtain the ball is short and velocity is not a large factor

- The current velocity must be in a disadvantageous direction. If the ball is travelling in a desirable direction it is better to not prevent it's movement but rather chase it in the direction it is moving. This case is checked for so robots will not stop a ball's path up the field and be left with possession facing the defensive half. If the ball's velocity is within 60 degrees of the desired kick direction then velocity prediction is not employed.

If velocity prediction is not employed the desired heading of the robot becomes the current heading to the ball. If velocity prediction is employed a direction is calculated as below. The rest of the procedure is to move to the predicted intersection in the most appropriate way.

5.12.2 Velocity Prediction Desired Heading Calculation

The heading desired considering the velocity of the ball is determined through calculation of the point that the robot can best intercept the ball's path. For simplicity the velocity of the ball in the short-term future is considered constant, at the current velocity, and the maximum speed of the robot's movement is also considered constant in all directions, at 0.75 cm per frame. The number of frames for possible intersection t is calculated by solving the possible location of the robot (circle radius t*0.75 centred at the robot's current position) and the position of the ball (line passing through the ball's current position in the direction of it's velocity. Since we must solve for time this becomes solving the intersection of a cone and a line in three dimensional space.

The working is carried out in robot space, hence the robot's current position is assumed $\underline{0}$, the balls position relative to the robot is $(x_B, y_B)^T$ and the velocity vector relative to the robot $(\dot{x_B}, \dot{y_B})^T$. The calculations become: -

$$\begin{pmatrix} 0.75 \times \cos(\theta) \\ 0.75 \times \sin(\theta) \\ t \end{pmatrix} = \begin{pmatrix} x_B + t\dot{x_B} \\ y_B + t\dot{y_B} \\ t \end{pmatrix} \tag{5.1}$$

$$(0.75t \times \cos(\theta))^2 + (0.75t \times \sin(\theta))^2 = (x_B + t\dot{x_B})^2 + (y_B + t\dot{y_B})^2 \tag{5.2}$$

$$(0.75t)^2 = (\dot{x_B}^2 + \dot{y_B}^2)t^2 + 2(x_B\dot{x_B}) + y_B\dot{y_B})t + (x_B^2 + y_B^2) \tag{5.3}$$

Hence the desired number of frames is the solution to the above quadratic, which can be calculated using the quadratic equation or a similar method. This in general has two solutions, though may have one or none. We take the smallest positive solution when it exists, if both solutions are negative either is taken. The case of no solution is produced when there is no intersection between the balls path and the robots possible positions. In this case we take the time when the ball's path is closest to the robots possible positions which is the minimum of the above quadratic and can be calculated by assuming a zero discriminant (tangent to the parabola is zero).

The solution is then restricted to an interval of [0,75], if it is less than zero it is negaitive and hence the solution is set to zero and velocity prediction is essentially not used. If the solution is greater than 75 the predicted time of intersection is greater than three seconds and the assumption that the ball's velocity is constant starts to produce significant errors, hence the solution is set to the maximum 75.

Once the position of intersection is calculated the heading to that point may also be calculated. If the heading is less than a hover parameter threshold the robot moves normally to the position. If the robot is currently at maximum speed (i.e. using Offset walk) the robot turns towards the heading maintaining full speed. If the heading is greater than the threshold and the robot is not at full speed it's desired heading becomes the heading to the ball and it attempts to move to the point by forward and left movement only, this produces a side stepping behaviour that enables the robot to catch, or at least trap, the ball.

5.12.3 Velocity Prediction Conclusion

Velocity prediction performs extremely well when a ball is moving towards it at a significant speed. It's main problems are detecting false positive estimations (estimations of significant velocity that are false), since it does slow the movement of the robot by a reasonable amount. Through the restriction of the skill it was enabled for the competition in 2003, though it's restriction leads to it having less of an effect than desired. When working well velocity prediction does produce a reasonable advantage for quickly obtaining possession of the ball. The velocity tracking in the localisation system and the velocity prediction skill

should be improved in the future and may become an advantage for rUNSWift in future years.

5.13 Stealth Dog

The basis of the rUNSWift game, speed and swift action, is severely hampered through the obstruction of players by opponents in the path of our movement. Although the obstruction rule is enforced, the damage is often done before obstruction is called (the robot still loses vital seconds) and sometimes the opponent in the way is also moving to the ball and hence does not fall under the obstruction rule. Given a method for avoiding such situations, the superior speed of the rUNSWift team would much better allow the ronbots to be first to the ball, and move it in a desirable direction.

The creation of the Stealth Dog (SD) skill was designed to provide this advantage to the rUNSWift team, to enable them to take a path avoiding opponent players, while maintaining top speed and preventing large deviation from their standard course. The properties of the Stealth Dog behaviour are extremely similar to the Bird of Prey, we would like to deviate the course to the ball from a straight line without sudden discrete deviations. We would also like to gain some of the other properties of the BOP, the move should be as locally based as possible, to avoid the complexity of global considerations, and self-localisation errors. The content of Stealth Dog is therefore based upon the BOP action.

5.13.1 Deciding on Activation of Stealth Dog

The first thing to acknowledge when considering when Stealth Dog is required is that if we are in doubt, it is a safer option to not activate SD. Although the evasion of opponent robots is desirable, it is better to impede both our robot and the opponent robot than to deviate far enough from our course to let the opponent player get first to the ball and obtain clean possession. For this reason the conditions to be met for SD are very cautious. They consist of overall conditions, and conditions on selecting opponents we should avoid.

The conditions for commencing stealth dog opponent selection are based around logical considerations of when a looped locus would be undesirable. They simple consist of: -

- Distance to the ball - if the ball is extremely close to the robot then a looped locus will have a large effect on the time it takes to get to the ball. It is also logical that if we are close to the ball, there is no time for our speed to beat an opponent to the ball by following a longer path, hence it is a better option to directly attack the ball and either gain possession or impede our opponent. For the competition SD was only considered if the ball was more than 30cm from the robot's position.
- Possible Edge Clash - if we are near an edge and there is a chance our

curved locus may intersect with that edge it is better to head straight for the ball. It would be detrimental to our chances of getting to the ball if our robot, attempting to avoid an opponent, gets caught on the edge, greatly slowing it's movement. This condition consists of checking if we are within 25cm of an edge, and if the heading to the ball is within a 240 degree interval, centered perpindicular into the edge.

The activation of SD also consists of the determination of which opponents it is desirable to avoid. This is done purely using visual objects to avoid the complexities of the global opponent tracking model and make the action more reactive. The opponent selection algorithm choses up to two opponents who are the closest in heading to the ball and who pass the conditions: -

- The opponent must be closer to us than the ball is. This condition is to avoid attempting to curve around opponents that are the other side of the ball relative to us, and also acknowledges the probability that the path straight to the ball is a better option and will allow the greatest chance of gaining possession
- The opponent must not be too close to the ball. If the opponent is already very close to the ball and has control imply moving straight to the ball is a better course of action than attempting to shift our path around them, if we run into the opponent we will, to some degree, disturb their use of the ball. For the compeition this condition was set to 20cm.
- The opponent is close enough to us. If the opponent is a relatively large distance from us it is better to keep a straight line to the ball and not worry about a possible collision. This also allows us to control the amount the SD shifts the robot's path away from a straight line, as we can balance this value with path variables to create paths closer or further from the opponent. For the competition this condition was set to a maximum of 50cm to the opponent.
- The opponent is not in a direct race to the ball with us. This condition is checked in case a robot is extremely close to us and to the side, i.e. in a "shoulder charge" race for the ball, in this case we should rely on our speed to beat the opposition. If this condition was not checked it would allow opponents to move us from our path simply by their presence near us, whereas we only want to avoid opponents directly in our path to the ball. For the competition this condition was set to ignoring an opponent closer than 20cm to us, and heading of greater than 25 degrees (absolute).

If no opponents were to be avoided SD would not be activated, otherwise the one or two chosen were checked for heading. If the opponents are close together both their headings are passed to the action, otherwise the heading to the closer opponent is passed to the action and the other opponent is ignored. This also leads to an emergent behaviour of weaving through opponents as the SD algorithm switches between them and the smooth motion is sustained.

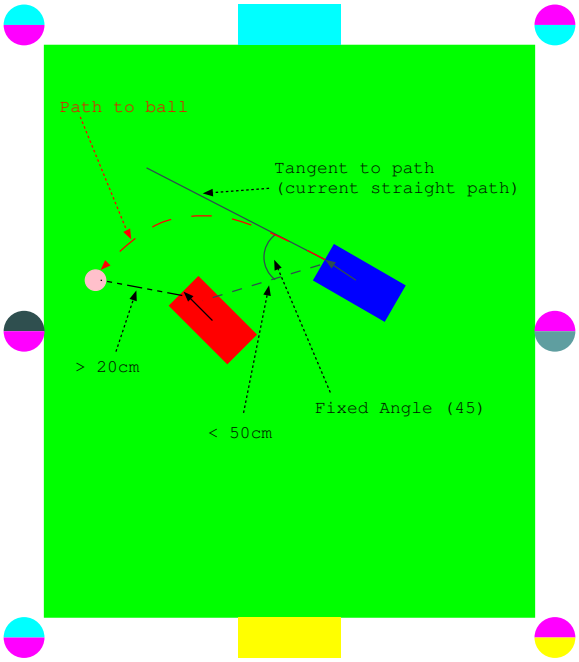


Figure 5.17: Blue Stealth Dog action's locus around a single opponent

5.13.2 Stealth Dog Action

The Stealth Dog action is relatively simple, working on the headings given to it from the condition checking described above. The current code simply takes the average of the two headings to be avoided (the two will be equal if there is only one opponent to be avoided) as the heading to compare with ball heading when determining the side to avoid the opponents. There is inactive code implemented to check if there is room to maneuver between two opponents but it is currently turned off. The SD action then simply avoids the two chosen headings by choosing a desired heading that is a fixed angle away from either of them on the side the ball is relative to their average. The desired heading is then attempted at maximum speed and a regulated turn factor to ensure smooth movement similar to the BOP action. For the competition the fixed angle was set to 45 degrees, though, as mentioned above, this angle can be used in conjunction with the maximum distance to an avoided opponent to create a locus that passes closer or further from the opponent. Fig. 5.17 shows the locus taken to avoid an opponent. It is important to also note that a hysteresis of 5 degrees is used to avoid the SD action from chosing a different side of the opponent to evade on each cycle.

When originally developed there was a fairly obvious problem with the stealth dog action. If an opponent was in the way of the ball, the locus of SD could take the robot around either side of that opponent. If the ball was roughly across the field and the opponent on our defensive side of the ball, our

robot would take a locus curved to the offensive side, which then caused it to make contact with the ball facing the defensive half (Fig. 5.17 shows a positive example of this where the blue dog makes contact with the ball towards the offensive half). This is in contrast to the general rUNSWift policy of approaching the ball facing the offensive half such that it could easily be pushed in an advantageous direction. This was a very large problem with SD since it was usually activated when opponents were close, and hence clear possession was not possible and the ball could not be pushed up field as there was no room for a 180 degree turn kick. In this case it is usually better to directly attack the ball and disturb the opponents use, rather than allow them to force the ball slowly toward our defending goal as we attack from the wrong direction. The action now checks for this cases and avoids SD when it desires a locus that curves toward the offensive goal.

5.13.3 Conclusion

The Stealth Dog action is a very experimental concept and is used very cautiously. In practical situations it has performed incredibly well, producing stunning behaviour that results in clean possession of the ball from seemingly impossible situations. However it is also plagued with inconsistency and sometimes produces negative effects such as allowing opponents clean possession of the ball. These negative effects have been minimised through hand calibration but still occur occasionally. The problems with Stealth Dog include poor visual observations of opponents mixed with a difficult trade-off between deviation from the straight path and collision with the opponent. The concept should definitely be explored further in the future but is an extremely difficult problem to solve.

Chapter 6

Locomotion

6.1 Overview

The locomotion module is responsible for handling the motion of the robot. This can be broadly divided into walking, head movement and kicking routines.

The geometry behind the walk and its parameterization remains largely unchanged, and thus readers should refer to last yearś undergraduate student thesis for details of its implementation. Major additions were made however, with the implementation of a gait optimization algorithm that runs on top of the existing geometry calculations to learn a walk locus shape that optimizes speed. For further details, refer to this paper.

This chapter thus serves as a summary, detailing how the locomotion module works from the users point of view, that is the behaviours module.

6.2 Walk Summary

The walking routine is very flexible and robust, and operates on a set of supplied parameters. These parameters can be modified to produce different stances and walking styles. Essentially the walk works by defining loci for each of the feet to trace out. All of the walking styles used employ the trot gait, in which the diagonally opposite legs are synchronized. This means that as the robot walks, the weight of the body is balanced diagonally across the robot, and shifts with each step taken. As one pair of diagonal legs lift up to move forward, the

other pair must stay on the ground. Once the first pair reaches the ground, the second pair can then lift up to move forward. In order to keep the robot stable, and to make changes in direction smooth, the step cycle is divided into 2 "half steps", and the movement of the robot is restricted so that it is only allowed to change its direction of movement at the end of a half step. Points on the loci that delineate the ends of a half step are termed "stepComplete". You can intuitively see that the step complete points are placed in neutral positions in the middle of the loci were it would be smooth to change the orientation and direction of the loci in order to make the robot walk in another direction.

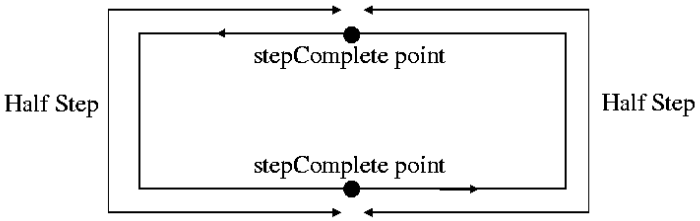


Figure 6.1: Diagram showing half steps and step complete points

There are 4 main styles of walk that the robot uses. Most of these involve changes in the shape of the loci, and result in slight differences in the properties of the walk. The 4 walk styles are:

- Rectangular Walk. This was the first walk style developed and uses simple rectangles as the shape of the loci that the paws must follow. The walk is smooth and allows the robot to walk in any direction, and combine any combination of forward, left and turn vectors.

- Canter Walk. This walk was the first variation developed and in this style, a canter motion is introduced by rocking the robot body in a sinusoidal wave pattern. This improves the speed of the walk considerably whilst maintaining the ability to walk in any direction. The camera shakes around a little bit more than it does in the rectangular walk but is steady enough for our purposes.

- Zoidal Walk. The second variation is the trapezoidal walk, named after the trapezoidal walk locus shape employed. This locus shape was configured manually so that the top of the locus is longer than the bottom. The rationale behind the trapezoidal walk is to avoid dragging the robots hind claws on the ground as it moves forward. The reduction in friction resulted in a faster forward walk. However, the walk is limited to walking forward and turning minimally.

- Offset Walk. Following the improvement in speed from the manual locus change in the zoidal walk, it was decided to have the robot automatically learn what locus shape would then optimize the speed. This resulted in a walk that was again faster than the zoidal walk but also a lot smoother in

that it shook the camera less. It also allows the robot to turn faster than the zoidal walk, but still has restrictions in the other directions.

- In this years strategy, we only used the Offset Walk and the Canter walk. The offset walk was used to chase down the ball. Because of its speed we tried to stay maximize the use of the offset walk, staying within the limits of its turning capacity whenever feasible. If however the dog was required to walk sideways or turn quickly, the canter walk would be used. The canter walk was also used to slowly and steadily walk up to the ball in order to grab it.

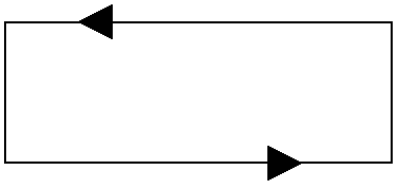


Figure 6.2: Diagram showing the locus shape for the rectangular and canter walk types

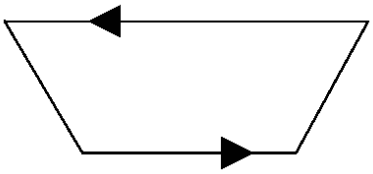


Figure 6.3: Diagram showing the locus shape for the zoidal walk type

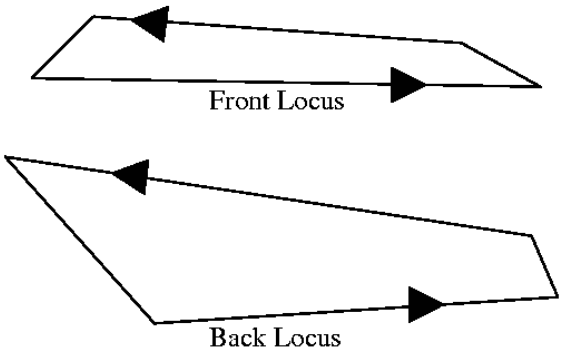


Figure 6.4: Diagram showing the locus shapes for the offset walk. Since the locus for the front and hind legs are different, both are shown.

6.3 Head Movement

The head has three axes of freedom pan, tilt and roll, however we only use pan and tilt. However there are 4 was in which one can specify head movements. These are:

- Relative. In this scheme, you specify a pan and tilt offset from the current head position. This is useful in making the head move in continuous paths such as circles.

- Absolute tilt then pan. In this scheme, you directly specify the tilt and pan angles you want the robot to move its head to. The pan angle is relative to the tilt angle. That is, you can consider the head as first tilting and then panning on the plane perpendicular to the neck axis. This directly translates into angles for the robots motors to move to as the pan motor is in the neck of the robot.

- Absolute pan then tilt. In this scheme, you directly specify the pan first and then the tilt angles that you want the robot to move its head to. This is different to the absolute tilt pan scheme as the tilt angle is now relative to the pan angle. This does not directly translate into motor angles. Motor angles that would make the robot look in an equivalent direction are calculated and used instead.

- XYZ coordinates. In this scheme you specify where you want the robot to look relative to the point on the ground underneath the robots neck. It uses a left handed coordinate system with the x-axis to the right, the z-axis to the front and the y-axis pointing upwards. Again motor angles that would make the robot look in this direction are calculated and used.

6.4 Kicking Actions

While the walking routines are primarily driven by the input parameters with few hard-coded aspects, the kicking actions, in contrast, are simply a playback of hard-coded joint angles. There are only 3 kicking actions of this type that the behaviours module uses. Although other kicking actions exist, they involve some decision making and so are explained in the skills section. These locomotion level kicks are all fairly fast and reliable. As the rUNSWift strategy focuses on speed and effectiveness during game situations, more elaborate kicks were not developed.

6.4.1 Front Kick

The front kick is comprised of three actions, and has a duration of 24 camera frames. To begin, the robot drops to the ground by reaching forwards and slightly inwards with its front legs, and spreading its rear legs, with the knees bent. By reaching forwards with the front legs, the robot is able to lift the ball slightly off the ground and hold it in place, allowing it to adjust its heading to aim the kick in a specific direction. The head aims straight ahead in this step so that it can see where it will be shooting. This step lasts for 6 camera frames.

The robot then raises its front legs over the ball, while lowering its head slightly to hold the ball down. This effectively places the ball in a good position to drop the front legs on, so as to shoot the ball. This step lasts for 12 camera frames.

Finally, the robot drops the front legs onto the ball, while raising its head back up. This shoots the ball out forward a fair distance, and is able to cover approximately half of the length of the new field. The kick is reasonably accurate over short distances, although as the ball slows down near its maximum range, it diverges off the kicked direction due to inconsistencies in the ball and field.

Because of the front kicks ability to hold the ball, it is used in the VOAK skill. The VOAK skill grabs the ball and then aims at gaps in the goal thereby avoiding the goalkeeper. It then executes the front kick to propel the ball forward. If the robot doesnt need to hold the ball to adjust its position, the faster lightning kick is used instead.

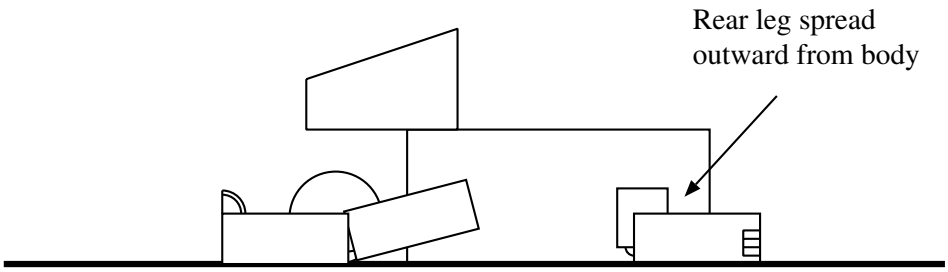


Figure 6.5: The initial step of the front kick.

6.4.2 Lighting Kick

The lightning consists of just two steps, and has a duration of 16 camera frames. It is because of its speed that it came to be known as the lightning kick. The main difference between the lightning kick and the front kick is that it doesnt need to first grab the ball. Instead, the robot simply walks up to the ball until

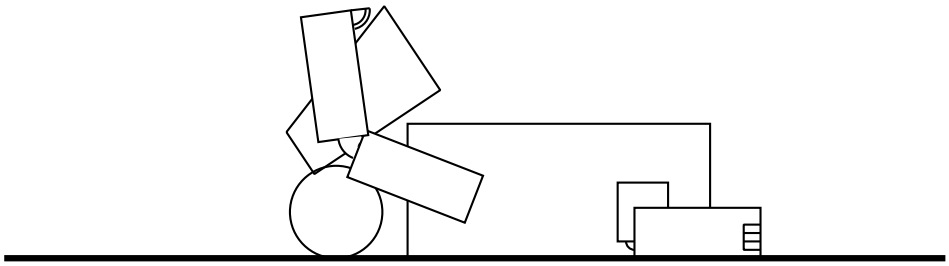


Figure 6.6: The second step of the front kick.

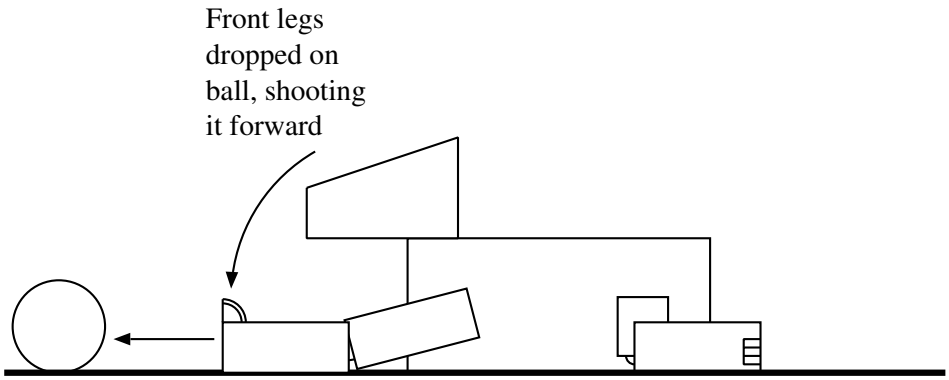


Figure 6.7: The final step of the front kick.

the ball is positioned under its chin. It then raises its arms and drops them onto the ball, thereby bypassing the first step of the front kick.

The first step has the robot looking almost straight down, whilst raising the front legs and spreading the rear legs, with the knees bent. If the ball is positioned correctly as described above, the robot will already be looking down anyways as it would have been tracking the ball. As the front legs move off the ground, the weight of the robot leans on the ball via the robots head, thereby holding it in place. This step takes 10 camera frames.

The second step then brings the front legs down on the ball, shooting it forward. The rear legs are now brought in while keeping the knees bent, shifting the weight forward onto the dropping arms. The head is raised in this step, since it must not trap the ball when it is hit. This step takes 6 camera frames.

The lightning kick is quite strong, also able to kick the ball approximately half the length of the field. Since this kick is powerful and requires only minimal setup time, it is used by the strategy to quickly move the ball upfield towards the target goal. The front kick is only used if the robot needs to adjust its heading before kicking the ball.

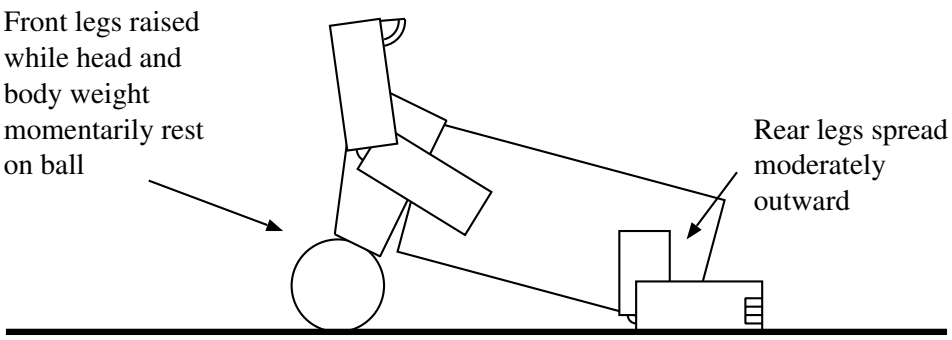


Figure 6.8: The initial step of the lightning kick.

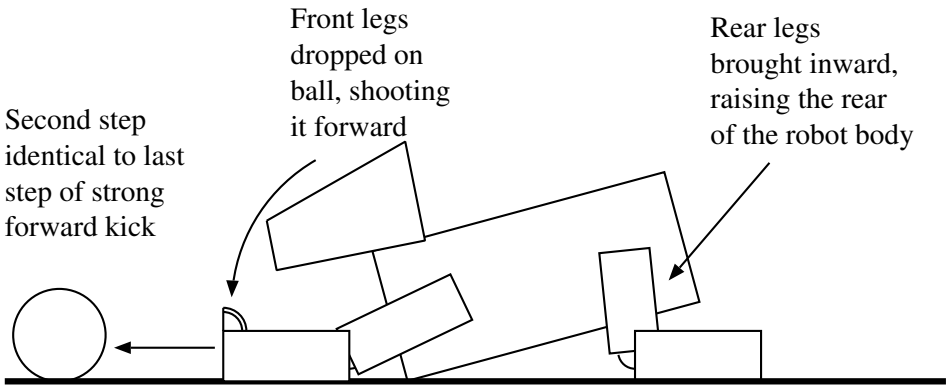


Figure 6.9: The second step of the lightning kick.

6.4.3 Chest Push

The chest push is a short range kick that is able to propel the ball approximately 30 cms. To begin, the robot readies itself by leaning slightly backwards. This is done by moving the shoulder joints of the robot, so that while the paws stay in place, with only the body moving back. It is important to ensure that the robot does not lean too far back, however to prevent it from falling over or otherwise becoming too unbalanced. This first step lasts 6 camera frames.

With the setup prepared, the robot then thrusts its body forward. Again, this is done by only moving the shoulder joints, thus pushing the robot body forward while the paws stay still on the ground. This is the step that actually propels the ball, as the body acts like a ram to push the ball forward. The extent of the forward push is tuned such that the robot pushes the ball with decent body momentum while ensuring that it does not fall forwards in the process. This step also lasts 6 camera frames.

The final step simply brings the robot back to the initial ready position, bringing the robot back into a steady stance to resume walking.

Because of its short range the chest push is used by the strategy to perform short passes to teammates. It was previously used to shoot for goal when the robot is close to the goal and doesnt want to over kick the ball incase it misses. However, new behaviours skill level kicks (namely the dribble and paw kick) have been developed to be used for this purpose and thus the use of the chest push has almost faded out.

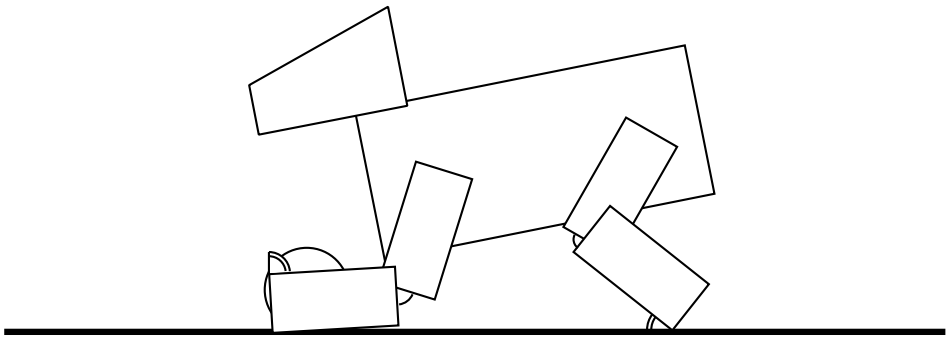


Figure 6.10: The initial step of the chest push.

6.5 Aperios Object Interactions

Now that the locomotion module has been separated out into its own Aperios object, it is no longer nicely synchronized with the behaviours module. This

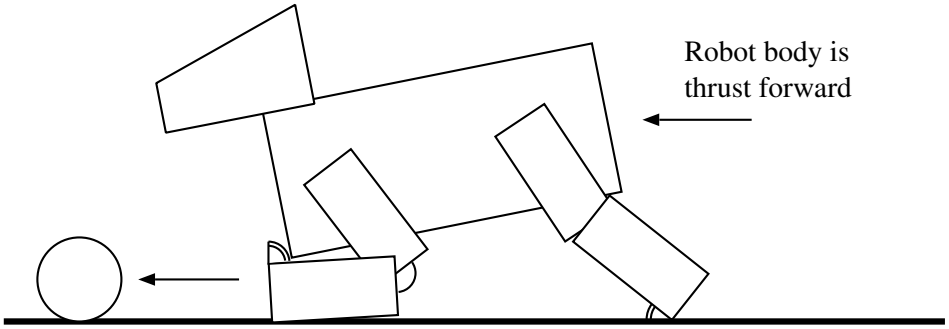


Figure 6.11: The second step of the chest push.

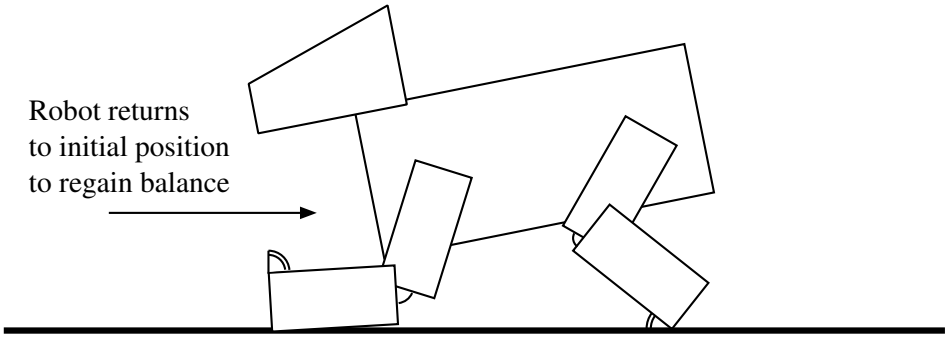


Figure 6.12: The final step of the chest push.

prompted a need for new methods of data exchange between the behaviours module which sends it walk instructions, and the oVirtualRobot object which the locomotion module then sends joint angle instructions to. The solution developed involved having a buffer between the behaviours module and the locomotion module, and a polling system between the oVirtualRobot object and the locomotion module.

The buffer between the locomotion module and the behaviours module is of length 2. It contains the instruction that the locomotion module is currently executing, and the next instruction that the behaviours module wants it to execute. Essentially, the behaviours module continuously writes over the next instruction field with a new instruction, thereby keeping the most recently desired instruction there. When the locomotion module is ready to accept a new instruction, it copies it from the new instruction field into the currently executing field. In cases where the behaviours module doesn't update the next instruction field, the locomotion module will again copy over the old instruction and thus continue doing what it was doing before. If the behaviours module ever misses a camera frame because it took too long processing the last image, the repeat of the last instruction will ensure that the robot keeps moving.

From the walk instruction given to the locomotion module, the locomotion module generates, via locus points, a set of joint angles that the leg motors must move through in order to carry out the instruction. These are sent in small spurts to the oVirtualRobot object which is essentially the programmer's interface to the hardware. Once oVirtualRobot is done executing the last spurt of joint angles it will poll the locomotion motion for more.

Typically instructions are sent from the locomotion module to oVirtualRobot at 125hz, whilst instructions are sent from the behaviours module at 25hz. Because of the polling system from oVirtualRobot to locomotion, it is refered to as a pull interface. The head motions however work on a push interface. As soon as the locomotion module receives a command to move the head from the behaviours module, it converts it to the corresponding head pan and tilt motor angles and pushes them onto oVirtualRobot.

The difference between actions taken for the head and the legs is attributed to a need to smooth out the leg motions to produce a smooth walk. It is less desirable to smooth out the head actions as we do indeed want the head to look in the direction specified and to look there as soon as possible.

Chapter 7

Overhead Camera

7.1 Introduction

The overhead camera is a tool that has been developed with the intention of assisting in various aspects of competition code development. There are several interesting possibilities that could arise from a successful implementation of the tool. One of these possibilities is to allow the robots to learn localisation. The robot is able to obtain its actual position from the overhead camera, so we may be able to teach the robot a better way of associating its position with visual cues. It is also useful for object distance calibrations, and odometry calibrations. However, the camera have been dismounted early in the year and development on the overhead camera have not resumed since, a prototype however, was completed.

7.2 Radial Distortion

A lens that has a sufficiently wide field of view to take in the entire width of the playing field also introduces a substantial amount of radial distortion. In order to remove this radial distortion, it is necessary to determine the radial distortion function that will re-map radii from the center such that straight lines stay straight. Radial distortion is defined in the form of the polynomial:

$$r = p + ap3 + bp5 + ...$$

Where p is the radius in the distorted image, a and b are coefficients of distortion and r is the undistorted radius. In practice only the first two coefficients of distortion are necessary to give sufficiently accurate results. So given the coefficient of distortions, it is then possible to apply the function and undistort every point on the image.

However the lens used were not supplied with the required information, so it was necessary to determine the radial distortion coefficients. The general approach is to obtain a picture with features that are known to be straight so that the degree of lens distortion may be calculated by analyzing the image.

The pattern chosen was a checkerboard pattern, this was a natural choice since there are many known straight lines on a checkerboard, and there are well defined corners that line up with each other, see Figure 7.1.

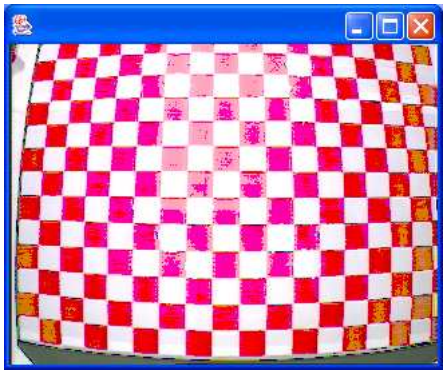


Figure 7.1: Checkerboard pattern

Ideally, the checkerboard should have been black and white, but there was a shortage of black material at the school storage room. So we use pink instead, since that was the other colour available in abundance. Points of high contrast between white and pink is then identified, as they identify areas of the image that represent straight lines, see Figure 7.2.

A 15x15 convolution filter is then applied to each point on the edges to identify candidates for corners, see Table 7.1. Depending on image quality, it is possible for the mask to identify several candidate corners next to each other, in which case they are then merged to form one corner. We chose each side of the mask to be of length 15 because that was the length of each square in the image, and this minimized errors in corner detection. Efficiency is not an issue at this stage because this process does not need to be executed once we have acquired the radial distortion constants. Rather at this stage, we are primarily concerned with accuracy.

These corners provide many collinear points in various directions, horizontal,

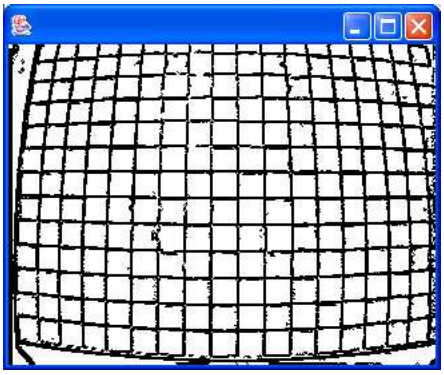


Figure 7.2: Checkerboard after edge detection

0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Table 7.1: Mask for corner detection

vertical and diagonal. It is then possible to calculate how much does each point deviate from their actual position. In Fig 7.4, p1, p2 and p3 are collinear corners on the image, a1 is where p3 should be and d is the deviation.

The position of the point p3 will change depending on the radial distortion function, so it is then possible to work out a radial distortion function such that d is minimized. If we can find a radial distortion function such that d is minimum for all sets of 3 collinear points, then we would have found the radial distortion function that best matches the radial distortion of the lens. However, at this point we chose to take up a tool called flatfish, to reduce development time. Flatfish is a tool developed by CMU for calculating radial distortion coefficients to flatten radial distortion in fish eye lens for their underwater robots [1].

[1] Hans Moravec, CMU *Robust Navigation by Probabilistic Volumetric Sensing, DARPA MARS program research progress*

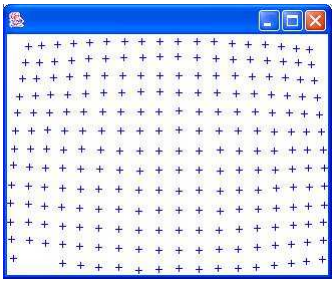


Figure 7.3: Checkerboard after corner detection

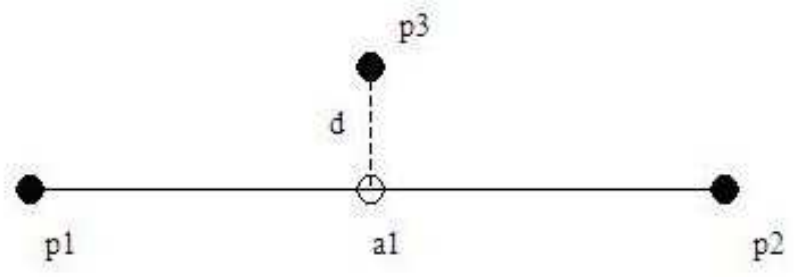


Figure 7.4: Determine error introduced by radial distortion between known collinear points

Flatfish uses a grid of circular spots for calibration, so we sprayed 130 black spots on a plastic sheet, which was then laid on the playing field. Since the overhead camera needs to distort images real time, it is too slow to apply the radial distortion function to each point of the image, so flatfish was modified to also produce a table that maps each pixel of the image to their respective undistorted coordinates, see Figure 7.5.

7.3 Object Recognition

A pink spot and a blue spot 10cm in diameter are attached to the back of the robot on a piece of black rectangular cardboard, see Figure 7.6. Red and blue were chosen because they were one of few colours that could be identified reliably by the camera under the bright competition lighting. The lighter colours such as yellow and pink were over saturated and became white. Green could not have been used for fear of being too similar to field green. Two spots were necessary so that orientation of the robot could also be identified as well as location. Position of the robot is the midpoint between red and blue spots, and its orientation is the angle of blue spot from red sport plus 90 degrees.

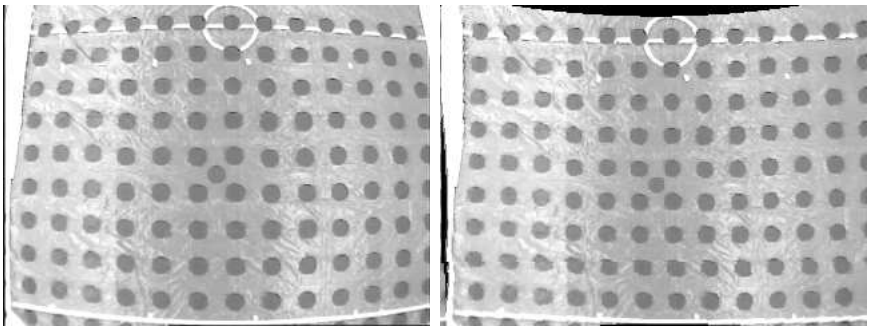


Figure 7.5: Before and after radial distortion removal

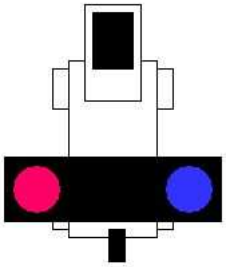


Figure 7.6: Spots on the back of robot

Object recognition is based on colour segmentation with simple analysis of surrounding pixels and simple pattern matching. Colours are calibrated using the same method used for competition, by classifying training data manually and applying the nearest neighbor algorithm for generalizations (refer to cahpter 2, Vision). However, because the spots occupy little more than 5 or 6 pixels in diameter, missing one or two pixels at each edge of the spot would make a huge difference in the location as well as orientation of the robot, hence the competition calibration method was found to be unsatisfactory. Due to fringing effects, pixels that lie on or lie close to the intersection of two colours differ substantially from their actual colour, and the nearest neighbor method of generalization fail to classify them correctly.

These pixels on the fringe of a spot usually deviate a fair amount from the colour of pixels at the center region of the spot. If we label these colours, we run the risk of having random noises on the field classified as spots. However to human eyes, the two still resemble the same colour if they are adjacent, such as in the case of the spot where the fringe colour surrounds the spot. However if these pixels are isolated then we no longer see them as being the same colour and thus we correctly see it only when it is part of a meaningful object. We are assisted largely by edge detection that is built into the hardware of our eyes, however it also show that we see colours in relative rather than absolute. Using a similar but much simplified idea, perhaps we can correctly classify the fringe

0	0	1	1	1	0	0
0	1	1	1	1	1	0
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
0	1	1	1	1	1	0
0	0	1	1	1	0	0

Table 7.2: Mask for spot detection

pixels of a spot if classify by also using information from the surrounding pixels. The first approach we experimented with was in taking the weighted average of surrounding pixels to assist in pulling a fringe pixel into the colour region of a spot colour pixel if it was adjacent to a spot, and to leave it out otherwise. The same method was also used with the median rather that the mean. We found mean to offer better performance because the mean of very dark and light colours produce random combinations that are not helpful to determining the real colour of a pixel. We also tried to tally the number of classified pixels in the surrounding area and to turn an unclassified pixel into the highest tallied colour. We found tallying to offer the best results in identifying fringe pixels correctly, and the result was such that reliable object detection can take place. However all three methods were very slow because it is costly to access all surrounding pixels for each non classified pixel in the pixture. There is only one robot on the field, in other words two coloured spots so the majority of pixels would be unclassified.

To reduce the number of pixels we need to check, we found a range for each of the spot fringes, and we only check the surrounding pixels if the pixel fell within the range of a spot fringe colour. This reduces the number of pixels that we need to apply surroundings check drastically, however in checking the surrounding itself, there is alot of overhead because we need to access 8 extra pixels for each pixel that we wish to check. Instead, we found that we can obtain very reasonale results by checking just one other adjacent pixel. This enables us to pick up close to all fringe colours, however it is less reliable in eliminating noise. But the noise introduced is easier to pick up because they would usually form a thin strip as a result of our method, so it is possible to filter these out in the object recognition routine.

After colour segmentation, coloured pixels are merged to form horizontal runs, and these runs are again merged to form blobs. To prevent random noises or portions of the robot from being identified as spots or ball, it is necessary to check whether the spots are circular in shape. For each spot coloured blob that is approximately spot width and height, a 7x7 mask of a spot is applied (see Table 7.2) and the blob is considered to be a spot if the difference between the mask and blob is less than 65%.

Although the mask offers reliable detection of spots, there are sometimes performance degradations. An alternative approach is to only check features of a spot. There are two checks used, the first is whether the vertical and horizontal central axis are approximately equal sized and that they both span the width/height of the blob. The second is to check that the density of the blob is over a certain threshold. Density can be calculated during blob formation relatively cheaply, and checking only axis reduces the number of pixel accesses from 49 to 14. In addition, radial distortion removal, run length encoding, blob formation and object recognition are all performed over a single pass of the image. The result was able to run at full frame rate, see Figure 7.7. We have obtained a profile of its performance, and ignoring the time in waiting for camera frames, in other words if the camera can always keep up with the program, then the program is able to run at 90 frames per second under our current set up.

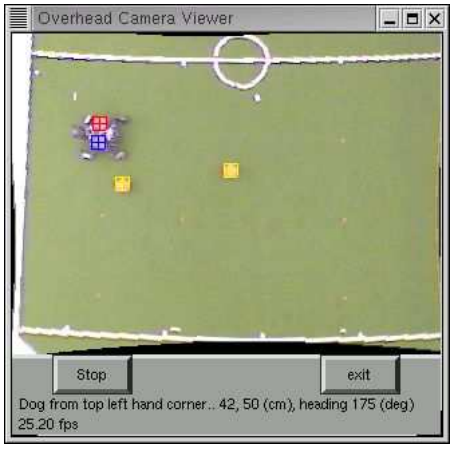


Figure 7.7: Overhead Camera

Location of objects on the field is determined by first finding the distance represented by each pixel. This is done by placing a 1.5 meter ruler at various locations of the field and divide its length by the number of pixels occupied, the various results are then averaged to work out an approximation of pixel to distance mapping. The initial attempt have object distances calculated by measuring their distance in pixels from the center of the image, then translating the pixel distance to real world distance and hence calculating their real world coordinates. However, because radial distortion is not fully removed, various distortions are still visible around the image, and these would affect the distance occupied by different pixels around the image. To minimize this error, in addition to the center pixel, we work out the real world coordinates of other pixels around the image so that given any object, it can choose the known pixel closest to it and workout its distance from that known pixel, hence reducing the error introduced by pixel to real world distance transformation. In practice, 4 additional points at the center of each of the four quadrants of the image were used.

The results of the overhead camera is found in Table 7.3.

Calculated (from top left corner)			Actual (from top left corner)			Error	
X (cm)	**Y (cm)**	**Heading (deg)**	**X (cm)**	**Y (cm)**	**Heading (deg)**	**Position (cm)**	**Heading (deg)**
47	49	100	50	50	105	3.16	5
48	99	204	50	100	200	1.73	4
102	101	75	100	100	80	1.73	5
146	100	314	150	100	315	4.00	1
202	100	233	200	100	235	2.00	3
201	151	104	200	150	110	1.41	6
201	49	179	200	50	180	1.41	1
47	153	265	50	150	270	4.24	5

Table 7.3: Comparison between location of actual robot and its calculated position as perceived by the overhead camera

7.4 Future Improvements

A single camera cannot capture the full playing area on the field. The current setup can catch only half of the field. To see the entire field, it will be necessary to install another camera. This will introduce extra complexity such as in synchronization between the two cameras, and the stitching together of images in real time.

The program will not recognise multiple robots. This is necessary if we are to play out a full game. In achieving this we will need better pattern recognition algorithms so that we can correctly identify robots from a myriad of spot patterns. In addition, the program does not currently handle occluded balls.

We need to track velocity of ball and robot.

Radial distortion removal is not ideal. It is still possible to see curvatures in white boundaries around the field. It may be necessary to switch to methods other than flatfish, as that was developed with fish eye lens in mind.

Chapter 8

Results

8.1 Australian Open 2003

The Australian Open 2003 was held on Friday the 2nd of May at the University of New South Wales. The competition involved four teams from Australia, rUNSWift UNSW/NICTA team, Newcastle University, University of Technology Sydney and Griffith University. The competition is set up as a friendly meet for the Australian teams in order to help them prepare for the wolrd competition held later in the year. The home ground, and success of past teams, placed a large amount of pressure on the rUNSWift 2003 team to perform well.

The competition involved each team playing each other team once throughout the course of the day, leading to each team playing three games in the round robin stage. The results of the round robin were: -

	rUNSWift	**Newcastle**	**UTS**	**Griffith**	**Points**
rUNSWift	-	4-0	10-0	7-0	6
Newcastle	0-4	-	5-0	5-0	4
UTS	0-10	0-5	-	2-1	2
Griffith	0-7	0-5	1-2	-	0

The round robin games for rUNSWift were extremely important since they were the first true competition the team had. The first game against the experienced Newcastle University was the most crucial since it was against an opposition that was well prepared, returning from a third place in the world competition 2002. The game brought both highs and lows, the 4-0 victory was

very successful considering the calibre of the competition. Despite this there were several robot crashes throughout the game and great frustration in the scoring department as many opportunities for goals were missed. The tean learnt a great amount from this first competitive game, particularly in the area of high level behaviour.

The round robin games against the new teams, UTS and Griffith, were fairly successful and highlighted the positive points in the rUNSWift offensive play. The game against UTS saw the opposition have a lot of trouble with robot crashes and localisation, particularly for the goal keeper, leading to a 10-0 victory. The game against Griffith was very similar, with the speed and skill of the rUNSWift team outplaying the oppoisition to a 7-0 victory. Both games still saw rUNSWift frustation in offensive situations and probably should have involved larger scores.

After the round robin stage the bottom two teams, UTS and Griffith, played off for third position, in which UTS won 4-1. Then the two top teams, rUNSWift and Newcastle, played in the final.

The final saw a large crowd gather in the small laboratory on level 3 K-17. This produced very different light conditions relative to the round robin matches, and large problems for the rUNSWift vision system. The adaption to new lighting conditions was shown to be substantially poor compared to the opposition, and gave the rUNSWift team many issues to think about after the competition. Despite these problems rUNSWift overcame Newcastle 5-3, in a close game, to become 2003 Australian champions.

8.2 RoboCup 2003

The world RoboCup competition in 2003 was held in Padua, Italy, from the 1st to the 11th of July. The first three days of this period involved the construction of fields and setup time for teams, the 4th day involved practice games, and the final two days was dedicated to the symposium. This left five consecutive days in which the competition took place, involving three days of pool matches, a day of quarter finals and challenges, and a day of semi finals and the final.

The rUNSWift team arrived in Padua on the 1st, minus one memeber due to passport problems. The team was in high spirits, anticipating a week of intense compeition with the other teams from around the world

The format of the competition was much the same as last year's RoboCup, a more compact format than years before. Unlike last year's tornament, and despite ordering the closure of the venue between 3-8am, the venue actually

stayed open 24 hours for team development. This lead to many late nights and early mornings for all teams involved, as last minute efforts to gain an advantage over the next day's opposition was attempted.

8.2.1 Team Setup and Practice Matches, July 2nd-4th

Having arrived in Italy on the night of the 1st, the rUNSWift team arrived at the competition venue early in the morning of the 2nd. The first task at the competition was to set up our system and make sure everything was working as we expected. This involved the setup of our table, the checking of equipment and the modification of wireless networking settings.

The rest of the team setup period involved the calibration of vision, movement and behaviours. The calibration of vision was a particularly intense task, as the lights used on the fields were very different to those in the lab. In particular a strobing effect was detected in the camera frames, caused by the strobing of the light in syncronisation to the elctrical power. Many pictures were taken and an appropriate colour calibration was developed over the several days.

The surface of the field was also found to be very different to that of the field in the lab. It had a "velvet" feel to it and was a lot more slippery than the lab surface. All the walking movements, such as the straight walk and the omni direction parameters, had to be hand calibrated in order to prevent the robot slipping on the surface. Due to this, many of the behaviours had to be also analysed and modified. The walking learner was also applied and continued throughout the main rounds of the tornament, improving the rUNSWift speed in every game.

The practice games involved two half games played by each team throughout the third setup day. rUNSWift drew UPennalizers from the University of Pennsilvania and LRP from France as it's opposition. UPenn was beaten 4-0 in ten minutes in a game that was highlighted by wireless communication difficulties for both teams. LRP was then beaten 6-0 as rUNSWift displayed it's superior speed and control. The practice matches were a great success for the rUNSWift team, apart from wireless communication problems that all teams were experienceing the robots were playing well.

In the setup period meetings were held by competition organisers and team leaders to organise the competition draw and to clarify rules. The competition involved 4 groups each of six teams. rUNSWift, due to being ranked second from last year's tornament, was chosen as the top seed in a group that also contained SPQR, Metrobots, UW Huskies, Jolly Pochies and Dynamo Uppsula.

8.2.2 Round Robin 1, July 5th

The first round of the competition saw the top seed of each group draw a bye, hence rUNSWift simply spent the first round watching other teams, analysing possible threats.

The first serious game for the rUNSWift team was against Dynamo Uppsula in the afternoon of the first competition day. Dynamo Uppsula were a new team this year from Sweden. Despite winning 11-0 the rUNSWift team still showed signs of problems both in vision calibration and behaviours. A long night was spent analysing the two areas and improving them.

8.2.3 Round Robin 2, July 6th

The second day of serious competition saw rUNSWift meet Metrobots in a morning game. Metrobots are a team formed from three universities, University of Columbia, City University of New York and Rutgers University. The rUNSWift team performed exceptionally well in the first half, gaining an 8-0 lead. This was largely due to the team taking it's scoring opportunities. It was a very pleasing half after a long night fixing problems. The second half was not quite as successful, leading eventually to a 13-0 overall win. Unfortunately, circumstances where the team capitalized with a goal in the first half, were not used in the second. Problems in scoring still had to be looked into.

The afternoon saw rUNSWift meet Jollie Pochies, a team from Kyushu university Japan. The speed and skill of the rUNSWift game was too much for the opposition as the eventual score was 14-0. This brought the tally up to 27 goals from two games, an extremely successful day. Despite this scoring frenzy the team missed many opportunities and several small problems surfaced for the members to analyse that night.

8.2.4 Round Robin 3, July 7th

The third and last day of round robin matches first saw rUNSWift meet SPQR from University of Roma, the only other team in the group to have not been defeated. In a fairly consistent game the score eventual came to 12-0 rUNSWift's way. The opposition was the toughest serious opposition to this point, and hence the result was successful.

The last pool game was rUNSWift against UW Huskies, from the University of Washington. Throughout the pool games UW showed amazing defensive ca-

pabilities including arguably the best goalie in the competition. The rUNSWift team was slowed down by a swarming strategy that was created with the purpose of slowing the rUNSWift robots down. Throughout the game the rUNSWift goal keeper had it's first touch of the competition as defence lapsed, although the keeper made no mistakes. The eventual score was 8-0, a relatively fine result, though the ability of the UW team to slow the rUNSWift robots down was noted.

The day also saw many other amazing results. The top seeds for the competition, CMU, were beaten by Wright Eagle, a team from China. This result saw Wright Eagle take top place in the group and CMU second place, placing them on our side of the draw. The other significant result for the day was Newcastle's 16-0 win over team Sweden, beating rUNSWift's best result for the competition and equalling the all time record set by rUNSWift2002. University of Newcastle ended up with the highest goal tally for the group stage edging out rUNSWift by only a couple of goals. The teams through to the quarter finals were: -

- Wright Eagles
- CMU
- Nubots
- Portus
- rUNSWift
- SPQR
- German Team
- UPennalisers

8.2.5 Quarter Finals and Challenges, July 8th

The day of the quarter finals also featured the challenges for 2003. The day started with the black and white ball challenge in which a ball designed like a regular soccer ball was to be detected and shot into the goal by the robot. No team in the competition managed to score a goal, so ranking depended on the time it took to make contact with the ball. Unfortunately, despite getting it closest to the goal, rUNSWift only managed a ranking of 5th.

The second challenge for the day was the localisation with no beacons challenge. The hot favourite German team made several mistakes thorughout this challenge, allowing rUNSWift to take first place with an impressive performance.

The afternoon saw the quarter final between rUNSWift and Portus from Lisbon, Portugal. A result of 12-0 was an extremely good score, though the game was far from that. The performance of the rUNSWift team was plagued with problems. Many behavioural problems were seen and the colour calibration became suspect, as it was suspected that the exceptionally large crowd altered lighting conditions. Many problems faced the team less than a day before the semi finals and final were played. This forced the team to work hard that night, missing the RoboCup banquet as a result. However the night did see the rUNSWift team have dinner with their traditional opposition CMU, 14 hours before playing the semi-final.

The afternoon also involved the obstacle avoidance challenge where the robot is to walk from one end of the field to the other without touching any of the robots standing on the field. Circumstances saw an obstacle configuration that was less than desirable for the rUNSWift team as the simplicity did not provide advantage for rUNSWift's learning algorithms to have an advantage. The German Team performed exceptionally well in the challenge, taking it and the general challenge competition out, leaving rUNSWift mid-table in the obstacle avoidance and second in overall rankings.

The day was not without exceptional drama with competition rules enforced in the quarter final between CMU and the German Team. After starting their robots as the wrong colour CMU quickly went one goal down, the German Team however allowed them to start again with no penalty. The eventual score of 2-1 to CMU was overalled by the competition committee and a penalty shootout took place. Aster many penalty shots CMU were the eventual victors in a ery close game.

8.2.6 Semi-final and Final, July 9th

The semi-final against CMU was a well anticipated game as the traditional rivals squared up once again. There was little doubt left in the result of 6-1 to rUNSWift, the careful deliberation of the CMU team no match for the rUNSWift speed and aggressive play. The one goal scored by CMU was while all three forwards for rUNSWift were penalised for the illegal defender rule, the goal keeper not able to hold on. The result overall was excellent and a sweet taste after the loss to CMU in the previous year.

The other semi-final saw Nubots meet UPennalisers in a game that went down to the wire. In a stunning result UPennalisers defeated Nubots in a penalty shootout after a 2-2 result. In a play off for third the Nubots overcame CMU 4-1.

The final against UPennalisers was the climax to a long session of work for the rUNSWift team. UPennalisers had improved dramatically throughout

the competition developing a particularly lethal kick to push the ball past the rUNSWift robots. The game began with an amazing goal by UPennalisers from their own half, curving around the rUNSWift goal keeper. The dampened spirits of the rUNSWift team recovered however in a fantastic half of soccer producing great play from both teams, with rUNSWift ahead at halftime 4-2. The second half saw UPennalisers come out determined to close the gap. An early goal from UPenn applied the pressure they desired to play an offensive game, setting up a terrific end to the competition. Despite UPenn's efforts an amazing display of defence shown by the rUNSWift robots prevented any further score in the game, making rUNSWift the eventual winners 4-3.

List of Figures