The University of New South Wales
School of Computer Science and Engineering

Road To RoboCup 2005

Behaviour Module Design and Implementation
System Integration

Author:
Nobuyuki Morioka 3057705
Bachelor of Engineering in Software Engineering

Supervisor:
Dr William Uther

Assessor:
Prof. Claude Sammut

September 22, 2005

Abstract

This thesis describes the behaviour module design and implementation of 2005 rUNSWift software architecture. With a complete restructure of the behaviour module, we have made several significant improvements in both the individual skill and team strategic levels. In the skill behaviours, the areas in which we had placed strong emphasis were the ball tracking, ball grabbing and grab dribbling. In the strategic level, improvements in both the dynamic role switching system and formations have contributed to better team communication and organisation.

As a result of these behavioural improvements as well as other low level module improvements made by other members in the team, rUNSWift has successfully come third in the world in the international robotics competition, 2005 RoboCup Osaka.

Acknowledgments

I would like to specially thank my supervisor Dr William Uther. Will has been a great supervisor throughout the course of this RoboCup project. His vision, imagination, enthusiasm, seriousness and craziness on the RoboCup were and are truly the factors which I respect. Without his leadership and guidance, we could not have achieved this far and come third in the world.

I would also like to thank Prof. Claude Sammut. Claude has taught me that being "good" is not good enough, being the "best" is the key to succeed in RoboCup. His leadership and guidance were also invaluable to our RoboCup project.

Special thanks to Brad Hall who has spent tireless hours planning out our RoboCup trips and everything. Without him, I think we were not able to stay in the hotel which was one minute walk to Intex Osaka, the venue of the RoboCup competition.

I must personally thank Min Sub Kim for being a friend who answered my RoboCup related questions in the lab.

I must thank the Faculty of Engineering, CSE and NICTA for their RoboCup project funding.
Without their funding, we couldn't have played with the AIBOs.
Without their funding, we couldn't have repaired those many broken AIBOs.
Without their funding, nothing was possible.
Thank you.

I must thank CAS for lending us one of their AIBOs. They have also kindly lent us five AIBOs to run our demonstration on Open Day. Many Thanks.

I thank all the K-17 Level 3 people who had to put up with the noise of the AIBOs' footsteps. Thank you for your understanding and cooperation.

NUBots team, our friend and our rival who we played against a couple of times before the RoboCup competition. Thank you for being friends with us.

I must thank the RoboCup Committee for choosing Osaka as this year's competition venue. Being able to come back to my home town, Osaka for the first time in eight years, especially as a member of the RoboCup team was indeed a miracle and a great privilege.

Many thanks to my wonderful RoboCup team members. Alex has been a great person to work with. His ideas and knowledge on RoboCup was amazing! Thanks for leading the team. Joshie has been an interesting person to work with. The mint must have been the key to our friendship! Thanks for offering the mint which I didn't want! Benny was another interesting person. We had enjoyed talking a lot of non-RoboCup related nonsense topics! Thanks to Andrew for being my great friend and I hope that our friendship will continue even after we graduate! Thanks guys!

Special thanks to Dr Ashesh Mahidadia who had helped me in my early years at university. He had given me many opportunities to learn and work. I will never forget your kindness and support you have shown.

I thank all my relatives who live in Japan. I was happy to see them all come to the RoboCup competition. I must particularly thank my uncle who had come all the way from Tokyo just to see my last two RoboCup matches.

Special thanks to my sister for helping out with my experiments.

I thank my parents for their hard work and support.
Without their hard work, I couldn't have gone to university.
Without their hard work, I couldn't have worked as hard.
Without their hard work, there was no way that I could have appreciated the importance of their support.

Thank you *times* zillion

Contents

I Motivation and Domain Analysis **12**

1 Introduction **13**
1.1 Purpose . . . 14
1.2 Background . . . 14
1.3 Changes in Competition which Affected Behaviour Module . . 15
1.4 2005 rUNSWift Overview . . . 16
1.4.1 Team Members . . . 16
1.4.2 Software Architecture . . . 17
1.4.3 Behaviour Module . . . 18
1.4.4 Formations . . . 19
1.4.5 Roles . . . 20
1.4.6 Skills . . . 20
1.5 Summary . . . 21

II Skill Behaviours **23**

2 Ball Tracking and Finding **24**
2.1 Introduction . . . 25
2.2 Concepts and Ideas . . . 26
2.2.1 Tracking Method . . . 26
2.2.2 Ball Velocity . . . 27
2.2.3 Calculating "Reacquiring" Direction from Camera Image 28
2.2.4 Normal Ball Finding Routine . . . 29
2.2.5 "No Green" Back Off . . . 30
2.2.6 Modification to Head Tracking when Turning . . . 31
2.2.7 Ball Hint Framework . . . 33
2.2.8 Action Hint Framework . . . 33
2.2.9 Using Ball Out Signals . . . 34
2.3 Design and Implementation . . . 34
2.3.1 Top Level . . . 34

2.4 Experimental Results . . . 36
2.4.1 Modification to Head Tracking when Turning . . . 36
2.5 Further Work . . . 37
2.5.1 Backing Off and Fighting in Scrum . . . 37
2.5.2 Ball Interception using Ball Velocity . . . 37

3 Ball Grabbing **39**
3.1 Introduction . . . 40
3.1.1 Speed Control . . . 40
3.1.2 Odometry Calibration . . . 41
3.1.3 Close Ball Inaccuracy . . . 41
3.1.4 Grabbing Moving Ball . . . 41
3.2 Concepts and Ideas . . . 42
3.2.1 Short Step Walk . . . 42
3.2.2 Side Step Adjustment . . . 42
3.2.3 Modification to Walk Command Interface . . . 43
3.2.4 Modification to Head Tracking . . . 45
3.2.5 Modification to Ball Edge Detection Algorithm . . . 45
3.2.6 Ball Distance Hysteresis . . . 45
3.2.7 Grabbing Criteria . . . 46
3.3 Design and Implementation . . . 47
3.3.1 Top Level . . . 47
3.3.2 GrabBall . . . 47
3.3.3 ApproachBall . . . 49
3.4 Experimental Results . . . 51
3.4.1 In Lab . . . 51
3.4.2 In Competition (vs NUBots) . . . 51
3.5 Further Work . . . 52
3.5.1 Removing Hestitation Before Grab . . . 52
3.5.2 Using Head Parameters to Switch On and Off Ball Edge Detection Algorithm . . . 52

4 Grab Dribbling **54**
4.1 Introduction . . . 55
4.2 Concepts and Ideas . . . 56
4.2.1 Grab Dribble Walk Type . . . 56
4.2.2 Grab Dribble Types . . . 56
4.2.3 Lining Up To Best Gap . . . 56
4.2.4 Dodging Around Obstacles . . . 58
4.2.5 Edge Behaviour . . . 60
4.2.6 Time Elapsed Action . . . 63

4.2.7 Ball Releasing 64
4.3 Design and Implementation 64
4.3.1 PerformToTargetGoal 64
4.3.2 PerformToDAD 67
4.3.3 PerformDodge 69
4.3.4 PerformEdge 71
4.3.5 PerformTimeElapsedAction 72
4.4 Experimental Results 74
4.4.1 Lining Up To Best Gap (In Lab) 74
4.4.2 Dodging Around Obstacles 75
4.5 Further Work 76
4.5.1 Obstacle Avoidance using Distance Sensor 76

5 Attacker Action Selection **77**
5.1 Introduction 78
5.2 Concepts and Ideas 78
5.2.1 Decision Tree 78
5.2.2 Field Division 79
5.2.3 Desired Attack Direction 80
5.2.4 Action Selection 81
5.2.5 Contested Ball Situation 82
5.3 Design and Implementation 83
5.3.1 Top Level 83
5.3.2 SelectInDefensiveThird 84
5.3.3 SelectInMiddleThird 87
5.3.4 SelectInOffensiveThird 89
5.3.5 SelectInEdge 93
5.4 Further Work 94
5.4.1 Considering Teammates' Information 94

III High Level Behaviours **95**

6 Role Behaviours **96**
6.1 Introduction 97
6.2 Attacker 97
6.2.1 Back Off Mechanism 98
6.2.2 Own Goal Box Avoidance 98
6.2.3 Active Localisation 99
6.2.4 Action Selection 100
6.2.5 Dodgy Dog 101

6.3 Supporter . . . 102
6.3.1 Positioning . . . 102
6.3.2 Back Off Mechanisms . . . 106
6.3.3 Bird of Prey with Dodgy Dog . . . 110
6.4 Striker . . . 111
6.4.1 Positioning . . . 111
6.5 Defender . . . 113
6.5.1 Positioning . . . 113
6.5.2 Ball Blocking . . . 115
6.6 Experimental Result . . . 115
6.6.1 Attacker Behaviour (In Lab) . . . 115

7 Formation and Strategy Behaviours 118
7.1 Introduction . . . 119
7.2 Supporter / Defender Formation . . . 120
7.2.1 Overview . . . 120
7.2.2 Local Interaction Between Attacker and Supporter . . . 121
7.2.3 Minimum Supporter Back Line . . . 122
7.3 Striker / Defender Formation . . . 123
7.3.1 Overview . . . 123
7.3.2 Field Coverage Effectiveness . . . 124
7.4 Dynamic Role Switching . . . 125
7.4.1 Time to Reach Ball . . . 126
7.4.2 Time to Reach Position . . . 127
7.4.3 Role Assignment Criteria . . . 127
7.5 Wireless Down Strategy . . . 131
7.6 Distributed Ball Finding Strategy . . . 132

8 Conclusion 134
8.1 Until Here 135
8.1.1 RoboCup 2005 Competition Results . . . 135
8.1.2 Competition Remark . . . 135
8.2 From Here ... Road To RoboCup 2006 Bremen . . . 136

List of Figures

1.1 2005 RoboCup 15
1.2 Architecture of the 2005 rUNSWift and its interactions 17
1.3 Decomposition of behaviour module 19
1.4 Structure of skill behaviours 21

2.1 Architecture : Skills focused 25
2.2 Two different point projections used to track the ball 26
2.3 Incorporating ball velocity into ball tracking 27
2.4 Motivation in ball interception using ball velocity 38

3.1 Side stepping stays aligned with the desired attack direction . 43
3.2 Turning misaligns with the desired attack direction 43
3.3 Ball distance hysteresis in ball grabbing behaviour 46

4.1 Comparing dribbling and turning towards the best gap 57
4.2 Obstacles inside the box is checked before dodging is triggered 59
4.3 Dodging direction when the best gap is detected 60
4.4 Dodging direction in the offensive half when the best gap is not detected 60
4.5 Dodging direction in the defensive half when the best gap is not detected 61
4.6 Motivation of edge behaviour 62

5.1 Field division shown graphically 80
5.2 Contested ball detection using obstacles 82

6.1 Architecture : Roles focused 97
6.2 Own goal box avoidance 99
6.3 Motivation of dodgy dog 101
6.4 Supporter positioning 103
6.5 Getting out of the way 108
6.6 Effectiveness of green back off behaviour 110
6.7 Striker positioning 112

6.8 Defender positioning 114
6.9 Attacker behaviour experiment 116

7.1 Architecture : Formations focused 119
7.2 Left : 2003 formation, Right : 2005 formation 121
7.3 Effect of minimum supporter back line 122
7.4 Field coverage effectiveness of striker positioning 124
7.5 Distributed ball finding strategy 133

List of Tables

2.1 Head kinematics pan, tilt and crane calculation 32
2.2 Tracking when turning without modification 36
2.3 Tracking when turning with modification 37

3.1 Attacker behaviour without a goalie for 3 minutes 51
3.2 Grab success rate in the first half 52
3.3 Grab success rate in the second half 52

4.1 Grab turn kick with gap lock 75
4.2 Grab dribble kick with gap lock 75
4.3 Dodging stationary obstacles 76

6.1 Attacker behaviour with a goalie for 3 minutes 117
6.2 Attacker behaviour without a golie for 3 minutes 117

8.1 Competition result : First round robin 135
8.2 Competition result : Second round robin 135
8.3 Competition result : Finals 135

List of Algorithms

2.1 CalculateReacquiringDirection 29
2.2 PerformTrackAndFindBall 35
3.1 PerformGrab 47
3.2 GrabBall 49
3.3 ApproachBall 50
4.1 PerformToTargetGoal 66
4.2 PerformToTargetGoal-Continue 67
4.3 PerformToDAD 68
4.4 PerformToDAD-Continue 69
4.5 PerformDodge 70
4.6 PerformDodge-Continue 71
4.7 PerformEdge 72
4.8 PerformTimeElapsedAction 73
4.9 PerformTimeElapsedAction-Continue 74
5.1 SelectAction 84
5.2 SelectInDefensiveThird 86
5.3 SelectInDefensiveThird-Continue 87
5.4 SelectInMiddleThird 88
5.5 SelectInMiddleThird-Continue 89
5.6 SelectInOffensiveThird-Grabbed 90
5.7 SelectInOffensiveThird-Contested 91
5.8 SelectInOffensiveThird-CornerAndEdge 92
5.9 SelectInOffensiveThird-Other 93
5.10 SelectInEdge 94
6.1 Calculating Supporter Positioning 104
6.2 Calculating Adjusted Supporter Positioning 106
6.3 Calculating striker positioning 112
6.4 Calculating defender positioning 114
7.1 Attacker Role Criteria 128
7.2 Supporter Role Criteria 129
7.3 Striker Role Criteria 131

Part I

Motivation and Domain Analysis

Chapter 1

Introduction

"RoboCup is an international research and education initiative, attempting to foster Artificial Intelligence and Robotics research by providing a standard problem where a wide range of technologies can be integrated and examined, as well as being used for integrated project-oriented education."

- *RoboCup 2005 OSAKA (www.robocup2005.org)*

1.1 Purpose

The purpose of the RoboCup competition is to stimulate research in artificial intelligence and robotics. For a robot to act sensibly and intelligently, it must be capable of at least the following [UNSW United 2000]:

- perceive its environment through its sensors,
- use its sensory inputs to identify objects in the environment,
- use its sensors to locate the robot relative to other objects in the environment,
- plan a set of actions in order to achieve some goal,
- execute the actions, monitoring the progress of the robot with respect to its goal.

The robot's decision making becomes very sophisticated once its environment include other agents.

A competitive game like soccer requires all of the above skills. Thus, RoboCup was created to provide an incentive for researchers to work, not only on the individual problems, but to integrate their solutions in a very concrete task. The benefits of the competition are evident in the progress from one year to the next. The competitive nature of the research has forced participants to evaluate very critically their methods in a realistic task as opposed to experiments in a controlled laboratory. It has also forced researchers to face practical hardware and real time constraints.

1.2 Background

Since 1999, UNSW has been participating in the 4-legged league, one of the five RoboCup Soccer leagues and has a remarkable achievement history. The rUNSWift, formerly known as the UNSW United had become the world champions three times (2000, 2001, 2003), second in the world twice (1999, 2002) and has come third in the world this year, 2005. This year, RoboCup was held at Intex Osaka in Osaka, Japan. Approximately, 400 teams from 35 countries have participated in this competition and has attracted around 200,000 visitors.

Each game of the 4-legged league consists of four Sony AIBO robots on each side where they play 10 minutes halves of hard soccer. Three robots are field players which strategically move the orange ball towards the opponent

half and score as many goals as they can, and one robot plays the role of a goalie which tries to protect the goal from the opponent robots attacking. Two goals are coloured as blue and yellow to let the robots know which goal to kick the ball into. Four coloured beacons are placed at the edge of the field to assist the robots in determining their positions on the field. Field lines are also there to aid the localisation of the robots.

Figure 1.1: 2005 RoboCup

1.3 Changes in Competition which Affected Behaviour Module

The changes introduced in this year's competition which greatly affected the behaviour module was, indeed, the removal of the walls around the field. Because there were no walls around the field, we had to modify our overall attacking strategy to keep the ball inside the field as much as possible. In previous years, the rUNSWift employed a simple strategy to push the ball

up the field using paw kicks where necessary. They have also used the walls effectively to push the ball around the walls. Unfortunately, the simple strategy they had used was no longer applicable to this year's field and a major restructuring in the behaviour module was needed. We will see the major changes which have been made due to the wall removal throughout this thesis.

In addition, the ball out rule was introduced because of the wall removal. During the game, when the ball is kicked out of the field, it will be placed on one of the six throw-in points located on the field. This rule has also affected our overall strategy and formations, and we will see this in detail in chapter 7.

1.4 2005 rUNSWift Overview

The 2005 rUNSWift team has started its journey to 2005 RoboCup Osaka from early January 2005. The team consisted of only 5 final year undergraduate students, Wei Ming Chan, Nobuyuki Morioka, Alex North, Joshua Shammay and Andrew Million Sianty who had began with a little or no understanding in robotics. The team has almost started from scratch, and has re-written the vision module, improved the actuator control module, revised the localisation module and polished the behaviour module. Although the team has only spent half a year preparing, they were successful in becoming 3rd in the world beating CMDash, Carnegie Mellon University of United States. 8 to 0 in the 3rd place play off.

1.4.1 Team Members

We have undertaken this RoboCup project as an undergraduate honours project. Each member has written a thesis and they are all available from our rUNSWift RoboCup website.

We will provide a brief introduction of the team members.

Weiming Chan - He worked on the actuator control. He devised a new style of walking from Skelliptical Walk and made a significant contribution in the overall game play.

Nobuyuki Morioka - He mainly worked on the behaviour module implementation.

Alex North - He mainly worked on the vision module and colour calibration. He also worked on the behaviours, particularly on the dynamic role switching and the positioning of both supporter and defender.

Joshua Shammay - He worked on the obstacle detection and the application of the obstacles.

Andrew Million Sianty - He worked on the technical challenges as well as the localisation module.

1.4.2 Software Architecture

Figure 1.2 shows the software architecture deployed in the 2005 rUNSWift code as well as its interaction with the environment. The robot is provided with OPEN-R framework which allows our code to communicate with the low level sensors and joint controls. The basic overview of the modules inside our architecture is as follows:

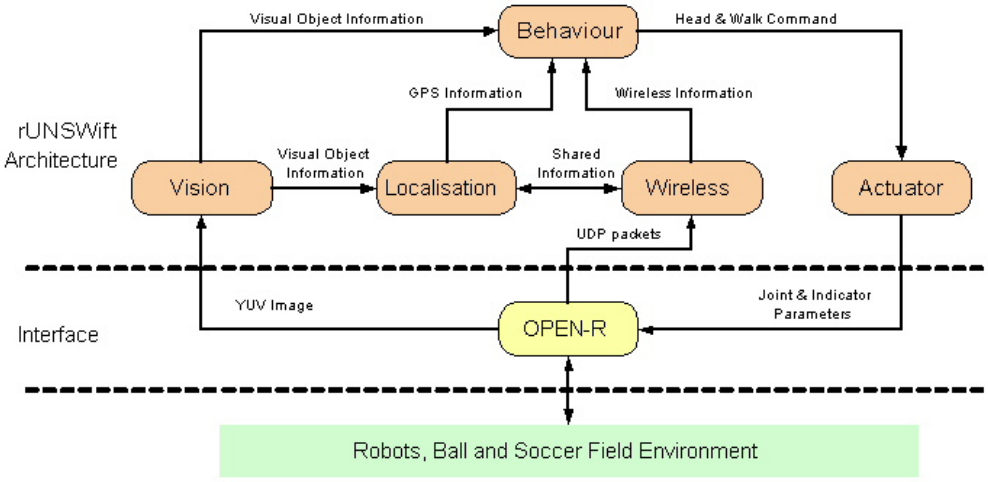

Figure 1.2: Architecture of the 2005 rUNSWift and its interactions

Vision - This module processes YUV images taken from the camera every 1/30th of a second. It uses both edge and symbolic colour transition to recognise objects on the field. Information processed in this module is passed onto the localisation and behaviour modules. For more information regarding this module, please refer to Alex North's thesis [A. North].

Localisation - This module tracks the position of the important objects on the field. They include the robots, the ball and obstacle features. It uses a multi-modal Kalman Filter for self localisation and updates its position using landmarks (i.e. beacons and goals). Field lines are also used to correct the estimated position. The ball is tracked using a single Kalman Filter and the obstacle features are tracked using our own derived method. For more information regarding localisation, please refer to Andrew Sianty's thesis [A. M. Sianty]. For more information regarding obstacles, please refer to Joshua Shammay's thesis [J. Shammay].

Behaviour - It takes in the information processed in both vision and localisation modules to make decisions on how the robot should move intelligently to play soccer. This thesis focuses mainly on this behaviour module.

ActuatorControl - This module receives commands from the behaviour module and moves the joints of the robot. It handles the locomotion for the robot. For more information regarding actuator control module, please refer to Weiming Chan's thesis [W. Chan].

1.4.3 Behaviour Module

This thesis places its focus on the design and implementation of the behaviour module. This module is the heart of the rUNSWift architecture. It combines all the information received from the vision and localisation modules and intelligently decides what to do. The decision is sent to the actuator control module as both head and walk parameters.

Inside the behaviour module, there are a couple of abstraction layers as shown in Figure 1.3. In a broad picture, this module is broken up into two layers, namely high level behaviours and skill behaviours. The high level behaviours are further broken up into formation and role behaviours layers. The formation behaviour is designed for robots to communicate with each other to work as a team and the role behaviour is designed to perform its individual responsibility. The role behaviour is implemented using several skill behaviours. These three layers will later be explained in the next several sections.

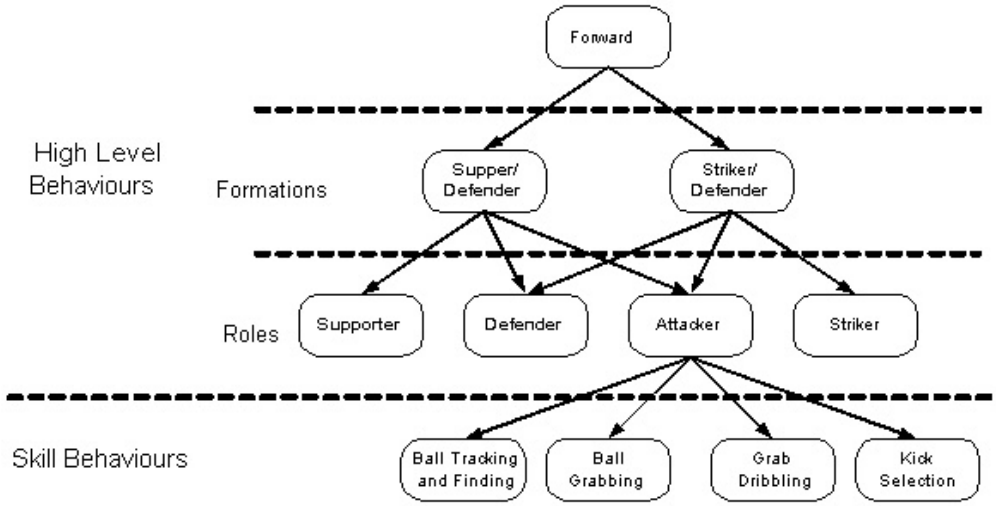

Figure 1.3: Decomposition of behaviour module

1.4.4 Formations

In 2003, rUNSWift had attacker, supporter and winger formations. In 2004, the formation stayed the same. This year, we have created two other formations to accommodate the changes made to the field dimension. Note that our formation strategy does not fix the role assignment to the robots. Depending on the situation, the three field players switch roles between them. In other words, they dynamically switch roles.

Attacker Supporter Defender - In short, Supporter/Defender formation. This is the formation which we had used in the competition. It places one field player in the defensive half all the time and plays as a defender role. Other two field players are ideally always close to the ball and switches role between an attacker and a supporter.

Attacker Striker Defender - In short, Striker/Defender formation. This is yet another working formation which we have prepared before the competition. The only difference between the above formation and this formation is that instead of having a supporter, this formation employs a striker who stays wide to wait for an opportunity. However, we did not use it in the competition, because our Supporter/Defender formation worked quite well in all of the matches we had played in the competition.

For more information on Supporter/Defender formation, please refer to section 7.2.

For more information on Striker/Defender formation, please refer to section 7.3.

1.4.5 Roles

During the game, every player has its own role to perform. The role assigned to each player is dynamically reassigned in every vision frame by our dynamic role switching system. Currently, we have four different roles and each role's behaviour is briefly explained below:

Attacker - The robot attacks the ball and tries to score as many goals as possible.

Supporter - The robot supports the attacker and the ball usually from behind. It stays relatively close to the attacker, so when the attacker loses the ball, the supporter can quickly change its role to become the attacker and reacquire the ball.

Striker - The robot waits in the offensive half of the field, but opposite side of where the ball is. The robot implicitly waits for the ball to pop out to the other side of the field, and as a result, it becomes the attacker to chase after the ball. We thought a striker is quite effective, because the striker position is usually free from the opponents. In other words, the striker had more chance of being free to wait for opportunities.

Defender - The robot stays in the defensive half of the field to avoid being counter-attacked.

1.4.6 Skills

A skill behaviour is a low level behaviour which does a specific task. It can also be defined as a complex procedure which tries to combine one or more actions. Actions also known as atomic actions are short, simple procedures with no feedback. These actions include walk command, head control and indicator control. In addition to this definition, one skill may invoke other skills in a tree fashion.

An example of a skill behaviour is the ball grabbing skill which is diagrammatically shown in Figure 1.4. This skill uses ball tracking and finding skill to approach the ball, and switches to its own set of procedures when the ball distance is close enough to get ready for a grab. The skill itself invokes walk and head control commands to do its own sophisticated behaviour.

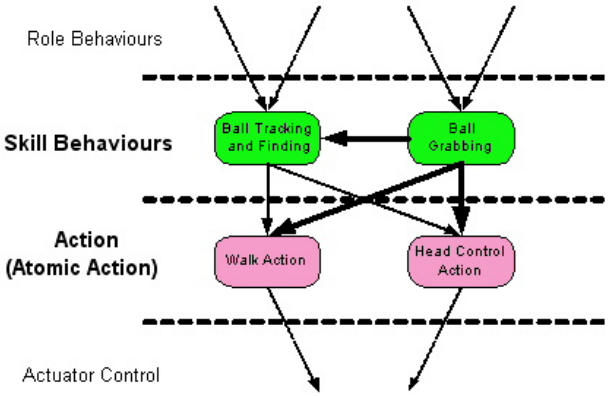

Figure 1.4: Structure of skill behaviours

1.5 Summary

This thesis focuses on the behaviour module design and implementation in the 2005 rUNSWift architecture. In addition to this, the system integration is another important key issue which we will describe during the course of discussion in this thesis. The system integration is basically on how we combine the low level modules like vision, localisation and actuator control together seamlessly to make them work as one functional unit, and we can see that this is what the behaviour module is trying to do. It tries to combine all the low level modules together to make robots play a strategic game of soccer. In order to do this system integration successfully, the key is to understand each module well to exploit the strength and overcome the weakness it has, and be able to connect them together.

To conclude this chapter, we will provide a summary of the structure of the thesis and function of each chapter.

- Chapter 2 looks at how we revised and improved the ball tracking and finding behaviour. This behaviour is the most important low level behaviour.
- Chapter 3 discusses the ideas and concepts behind the ball grabbing behaviour. This behaviour has played a crucial role in developing the strategy which adapted to the field with no walls.
- Chapter 4 extends the ball grabbing behaviour discussed in chapter 3 to allow a robot to dribble with the ball under its chin.

- Chapter 5 describes the attacker action selection. This action selection selects an appropriate action according to the situation that a robot is in.
- Chapter 6 discusses individual role behaviours. These roles include attacker, supporter, striker and defender.
- Chapter 7 looks at the formation behaviours as well as high level strategies of our rUNSWift architecture.
- Chapter 8 concludes this thesis.

Part II

Skill Behaviours

Chapter 2

Ball Tracking and Finding

"Find ball is broken!"

- Dr William Uther (During our regular practice matches)

2.1 Introduction

In this chapter, we will describe the skill behaviours in detail. As Figure 2.1 shows, the skill behaviours are the low level behaviours which make up the foundation of the role behaviours.

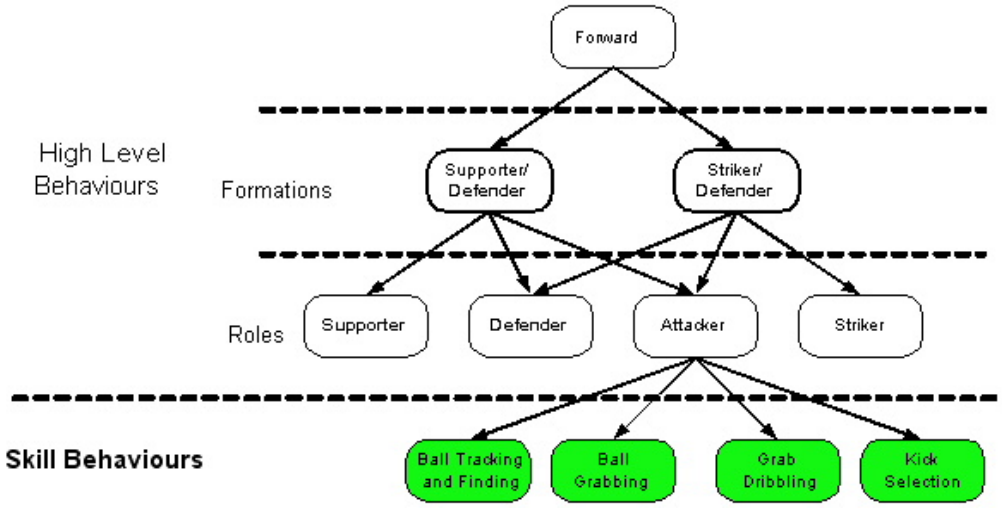

Figure 2.1: Architecture : Skills focused

This chapter describes the ball tracking and finding skill in detail. We must emphasise that this is the most important skill above all. In competitive games like RoboCup, the key is to look at the ball reliably and find it as quickly as possible when it is lost. If one team can find the ball quicker than the other team, then that team has a better chance of getting possession of the ball.

This year, we have revised and improved the ball tracking and finding behaviour with numerous ideas which will be discussed in this chapter. The essence of ball tracking and finding is that a robot should use both of its head and body movement together. We must not think of the head and the body movement as two independent motion controls, rather it is important to think of them as one combined movement.

In this chapter, we will explore the concepts and ideas involved in implementing this ball tracking and finding behaviour. This will be discussed in section 2.2. This will be followed by the detailed implementation of the

behaviour in section 2.3.

2.2 Concepts and Ideas

2.2.1 Tracking Method

This year, we have experimented using two different projection points to track the ball. One is to project top of the ball on the camera image to the ground and use the point projected on the ground as a point to look at. Another is to project top of the actual ball using the ball distance estimation calculated from the radius of the ball and the ball heading calculated from horizontal angle projection.

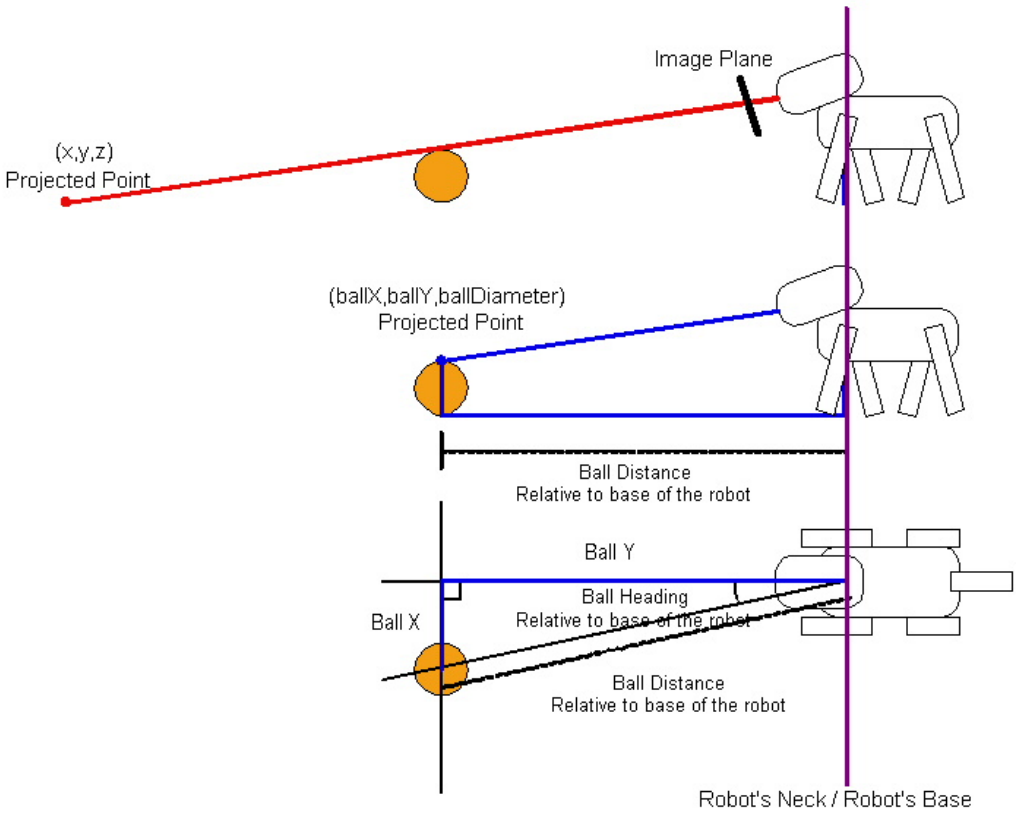

Figure 2.2: Two different point projections used to track the ball

These two different ideas are shown in Figure 2.2. The red line shows the

first point projection method and the blue line shows the the second point projection method using the ball distance and heading.

After running some tracking tests on these two projection points, we concluded that the first projection works much better than the second one. This is especially true when the ball is just in front of the robot, because the distance estimations can sometimes go wrong for the close ball. The robot sees the close ball as a ball with extremely small radius which results in calculating the ball distance to be extremely far. Thus, instead of looking down at the close ball, since the robot thinks the ball is far away, it looks straight ahead and loses the ball from its vision. Whereas, by projecting the top of the ball in the camera image calculates the correct point for the robot to look at.

2.2.2 Ball Velocity

By using the ball velocity, it is possible to predict where the ball would go in the next frame. However, this is only true if the underlying velocity calculation is reliable and accurate.

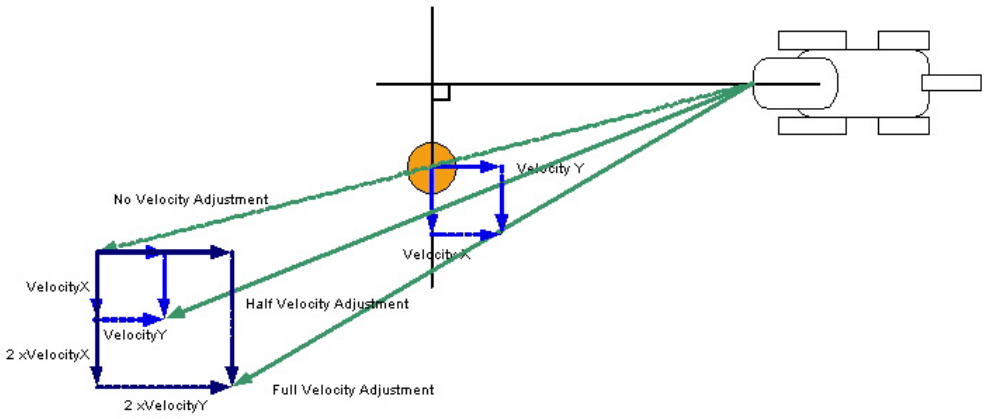

Figure 2.3: Incorporating ball velocity into ball tracking

Figure 2.3 shows how we applied the velocity prediction into our ball tracking behaviour. This figure assumes that we are tracking the ball using the point projection method which we have described in the previous section. We can see that a robot is actually tracking a point on the ground which is twice as far as where the ball is located. This can also be seen in Figure 2.2. If we

were to add x and y components of the ball velocity to the point which the robot is actually looking at, then this would mean that we are adding half of the ball velocity to the ball position, and this is referred to as a half velocity adjustment in Figure 2.3. If we double the ball velocity and add it to the point, then it will mean that we have updated the ball position with a full velocity. This year, we used a half velocity adjustment on the ball position instead of a full velocity adjustment to be conservative on the use of the ball velocity.

2.2.3 Calculating "Reacquiring" Direction from Camera Image

When the robot loses the ball from its vision, it is desirable to know which direction to point its head to reacquire the ball which has been lost. This direction is known as the "Reacquiring" direction and it is calculated from image coordinates of the last visual ball. In this section, we will describe this calculation method.

Basically, the calculation is done by using coordinate transformation and checking two angles which are derived from a series of vectors relative to the camera image. These two angles are ball vertical angle and ball horizontal angle. The angle derivations are as follows.

The ball vertical angle is calculated from a vertical vector relative to the image and the ball vector which is also relative to the image. The vertical vector is calculated from a vector relative to the base of the robot and this is transformed using rotation of tilt, pan and crane. This step is shown in the first several steps of Algorithm 2.1. The ball vector is a vector from the centre of the image to the image coordinate of the ball.

The ball horizontal angle is calculated from a horizontal vector relative to the image and the ball vector which we have explained already. The horizontal vector can be determined from the two horizon points drawn on the image.

After calculating the two angles, we can determine the direction the ball lies on the image. The direction is classified into four major category. They are top left, top right, bottom left and bottom right.

Algorithm 2.1: CalculateReacquiringDirection

```
   Data:
   Result:
 1 verticalRelRobot = (0, 0, 1)
 2 verticalRelRobot.rotate(tilt)
 3 verticalRelRobot.rotate(pan)
 4 verticalRelRobot.rotate(crane)
 5 verticalRelImage = (vt1[0], 0, vt1[1])
 6 verticalRelImage.normalise()
 7 h1x, h1y = getHorizonPoint()
 8 h2x, h2y = getOtherHorizonPoint()
 9 horizontalRelImage = (-(h2y - h1y), 0, (h2x-h1x))
10 horizontalRelImage.normalize()
11 bx = ballImg.x - IMAGE_WIDTH / 2
12 by = ballImg.y - IMAGE_HEIGHT / 2
13 ballRelImage.normalise()
14 ballRelImage = (-by,0,bx)
15 verticalAngle = cos^-1(dot_product(verticalRelRobot, ballRelImage))
16 horizontalAngle = cos^-1(dot_product(horizontalRelImage,
   ballRelImage))
17 if 0 <= verticalAngle <= 90 then
18 |  if 0 <= horizontalAngle <= 90 then
19 |  |  return TOP_RIGHT
20 |  else
21 |  |  return TOP_LEFT
22 |  end
23 else
24 |  if 0 <= horizontalAngle <= 90 then
25 |  |  return BOTTOM_RIGHT
26 |  else
27 |  |  return BOTTOM_LEFT
28 |  end
29 end
```

2.2.4 Normal Ball Finding Routine

If a robot has lost the ball, then it will do a sequence of moves to find the ball. This sequence of moves is referred to as the ball finding routine. This include the following moves:

1. 0 - 3 lost ball vision frames : Look at the last visual ball.
2. 4 - 10 vision frames : Look at the world model ball, or look at the weighted last visual ball, or look down.
3. 11 - 65 vision frames : Scan its head around on the spot, or back off while scanning its head.
4. $>$ 65 vision frames : Look and turn to the wireless ball, if the wireless ball is valid.
5. $>$ 65 vision frames : Spin around to find the ball.

This routine is called every time the robot loses the ball in its vision. However, there are some certain cases when other routines are used for the robot to act appropriately, because the above routine is not optimised for situations which happen occasionally during the game. There are several mechanisms which handle these situations and they include ball hint framework (section 2.2.7), action hint framework (section 2.2.8) and using ball out signals (section 2.2.9).

2.2.5 "No Green" Back Off

"No Green" back off is a behaviour which a robot backs off while it scans around to look for a ball. This is part of the normal ball finding routine which we discussed in the previous section. It is triggered when the following three criteria are satisfied:

1. The robot cannot see the ball.
2. The robot is looking straight forward. The word "straight" is relaxed in this criterion. It is between -45 and 45 degrees.
3. No green features are detected in the camera image.

We must note that this behaviour is the modified version of one of the back off mechanisms used by non-attacker role behaviours. The back off mechanisms will also be discussed in section 6.3.2.

When all the above criteria are satisfied, we can expect the robot to be in the following situations:

1. The robot is off the field and is not facing towards the field.
2. The robot is in a scrum.

3. The robot is in a dark place.

During the course of this RoboCup project, we believed that in a scrum situation, it was logical to back off to get out of the scrum, instead of going aggressively into the scrum to fight for the ball which the robot cannot see. However, not fighting for the ball in the scrum also creates a open space for an opponent to get the ball before us. In section 2.5.1, we will discuss how we can improve our current model to balance out backing off and fighting aggressively in the scrum.

2.2.6 Modification to Head Tracking when Turning

To track the ball, a robot specifies the projected point of the ball explained in section 2.2.1 and this projected point gets converted into pan, tilt and crane values in the head kinematics which belongs to the actuator control module. The conversion is necessary, as the robot uses angle values to set the position of its head.

Table 2.1 shows the calculated pan, tilt and crane values when a robot is asked to look towards a heading shown in the first column and distance of 100cm away. The second column actually shows the point in x and y co-ordinates for convenience. When the heading reaches below -90 degrees or above 90 degrees, we can see that the tilt value suddenly increased up to -80 degrees. This sudden change does not affect the robot's ball tracking significantly when it is standing still. However, when it is moving especially turning on the spot, this sudden tilt angle change makes extremely difficult for the robot to track the ball.

For example, if the ball is caught in the corner of a robot's vision while it is spinning around to look for it, the ball heading would appear large (i.e. greater than 90 or less than -90). According to Table 2.1, the next thing the robot would do to track the ball is to set its tilt angle to -80. However, as it is dropping its head to -80 degrees and turning to face towards the ball, the ball heading decreases and thereby increasing the tilt angle back up again. These immediate drop and increase of the tilt angle cannot be handled well by the mechanical motor of the head, and hence, delays the head motion. Due to this head motion delay, the robot could potentially loses the ball from its vision. Although the world model ball information could give the robot a hint of where to look at, the robot still hesitates a little bit and cannot track the ball smoothly while spinning around to look for it.

Angle	(X-Cord, Y-Cord)	Pan	Tilt	Crane
-110	(-93.9693, -34.202)	-82.3516	-80	-22.8756
-105	(-96.5926, -25.8819)	-82.8111	-80	-18.0584
-100	(-98.4808, -17.3648)	-83.2031	-80	-13.2065
-95	(-99.6195, -8.71557)	-83.537	-80	-8.31943
-90	(-100, 6.12303e-15)	-83.8196	-80	-3.39664
-85	(-99.6195, 8.71557)	-82.1701	-48.25	-2.34241
-80	(-98.4808, 17.3648)	-78.2579	-19.8986	-3.533
-75	(-96.5926, 25.8819)	-73.8021	-10.5361	-3.49519
-70	(-93.9693, 34.202)	-69.1032	-5.26082	-3.48068
-65	(-90.6308, 42.2618)	-64.2943	-1.95755	-3.47373
-60	(-86.6025, 50)	-59.4275	0.272559	-3.46988
60	(86.6025, 50)	59.4275	0.272559	-3.46988
65	(90.6308, 42.2618)	64.2943	-1.95755	-3.47373
70	(93.9693, 34.202)	69.1032	-5.26082	-3.48068
75	(96.5926, 25.8819)	73.8021	-10.5361	-3.49519
80	(98.4808, 17.3648)	78.2579	-19.8986	-3.533
85	(99.6195, 8.71557)	82.1701	-48.25	-2.34241
90	(100, 6.12303e-15)	83.8196	-80	-3.39664
95	(99.6195, -8.71557)	83.537	-80	-8.31943
100	(98.4808, -17.3648)	83.2031	-80	-13.2065
105	(96.5926, -25.8819)	82.8111	-80	-18.0584
110	(93.9693, -34.202)	82.3516	-80	-22.8756

Table 2.1: Head kinematics pan, tilt and crane calculation

To fix this problem, we have modified how a robot should track a ball while it is spinning or turning at a large angle. If we know that the robot is spinning around, then we clip the ball heading to 80 degrees if it is positive and -80 degrees if it is negative. By doing this, the robot will never set its tilt angle to -80, because the ball heading will never be above 80 degrees or below -80 degrees when spinning or turning as also shown in Table 2.1. This clipping also has one more effect and that is the robot implicitly looks at a point which is ahead of where the ball is. Looking ahead while it is spinning avoids the ball running across its vision.

In section 2.4.1, we will analyse the experimental result and see how this modification has improved the ball tracking while spinning and turning.

2.2.7 Ball Hint Framework

Ball hint is a mechanism built inside the ball tracking and finding behaviour which we can explicitly set hint positions at which a robot can look to find the ball rather than looking at the world model ball position which is described in the normal ball finding routine (section 2.2.4). Hints can be specified from outside this ball tracking and finding behaviour and they are only valid for at most 15 vision frames. There are a few criteria which the ball hint is used over the world model ball and they are as follows:

1. A hint has been specified.
2. The hint was specified at most 15 vision frames ago.
3. The robot has lost the ball for more than 3 vision frames, but no more than 10 vision frames.

2.2.8 Action Hint Framework

Our ball finding routine is optimised for when a robot has lost the ball while tracking and chasing after it. Thus, if the robot loses the ball in unoptimised situations, then our finding routine will not be efficient. Because of this reason, we have a mechanism to change the ball search routine and this mechanism is referred to as action hint framework. This can tell the robot to forget the normal finding routine and do a special sequence of actions to handle those unoptimised situations.

There are currently three different action hints which we can set in the ball finding behaviour.

1. Spin Hint - It spins around to look for the ball. This is useful when the ball slips out from a robot's chin while it is grab turning. Since it is turning, by the time it detects it had lost the ball from its chin, it has turned past the ball already and the ball is most likely to be out of the robot's sight. Thus, invoking the normal ball finding routine is inefficient. Since the ball is likely to be around the side or behind the robot, it is quicker for the robot to spin around to reacquire it.
2. Follow Hint - It walks straight forward while it looks at the world model ball position for several vision frames in the first phase and scans its head around in the second phase. This is used to reacquire the ball as soon as it is kicked from the grab dribble. Since the ball is kicked forward, there is no need to stand still and look for the ball. It is better

off walking straight forward and looking for the ball. Furthermore, in the grab dribble, it updates the world model ball position to the likely position which the ball will be kicked to, so that the robot has better chance of reacquiring the ball by looking at that updated position in the first phase of this action hint.

3. Scan Hint - This tells the robot to scan its head around immediately. This hint is useful in the active localisation for an attacker. In the active localisation, the robot looks away from the ball to point its head to one of the beacons on the field while chasing after the ball. After the robot has finished looking at the beacon, it is better to go into scan mode rather than turning its head back to the original position, because while the active localisation is triggering, the local ball position may change.

2.2.9 Using Ball Out Signals

Because this year's field does not have walls around it, the ball could go out of the field. If the ball goes out of the field, it will be placed on one of the six throw-in points in the field. At the same time, all the robots on the field will also be notified with information that the ball has gone out and the team which kicked it out. This notification is done via a wireless message sent from the game controller.

By using this information, it is possible to calculate at which throw-in point the ball will likely be placed and we can let the robots look towards that direction when they cannot see the ball. This can, in fact, be thought of as another action hint which was just explained in the previous section.

2.3 Design and Implementation

2.3.1 Top Level

Algorithm 2.2 shows the top level of the ball tracking and finding behaviour. If the ball is already visible, then a robot can just call trackVisualBall() to track the ball. Otherwise, the robot needs to go and look for the ball which is outlined in the "else" statements. The steps involved in finding the ball is exactly the same as explained in section 2.2.4 with addition of a few extra special cases like find by ball hint, action hint and ball out signal.

Algorithm 2.2: PerformTrackAndFindBall

```
   Data:
   Result:
 1 if ball is visible then
 2 |  trackVisualBall()
 3 else
 4 |  if there is any action hint to do then
 5 |  |  findByActionHint()
 6 |  end
 7 |  else if lostBallFrame < LAST_VISUAL then
 8 |  |  findByLastVisual()
 9 |  else if lostBallFrame < LOST_BALL_HINT then
10 |  |  findByBallHint()
11 |  else if lostBallFrame < LOST_BALL_GPS then
12 |  |  if the ball out signal is received then
13 |  |  |  findByBallOutSignal()
14 |  |  else if last seen ball was right in front of the robot then
15 |  |  |  findByLookingDown()
16 |  |  else if last seen ball was close from the robot then
17 |  |  |  findByWeightedVisualBall()
18 |  |  else
19 |  |  |  findByGps()
20 |  |  end
21 |  else if lostBallFrame < LOST_BALL_SCAN
22 |  AND the robot was not doing findBySpin() just then then
23 |  |  if the robot cannot see any green in front then
24 |  |  |  findByScanBackingOff()
25 |  |  else
26 |  |  |  findByScan()
27 |  |  end
28 |  else if wireless ball is available then
29 |  |  findByWirelessBall()
30 |  else
31 |  |  findBySpin()
32 |  end
33 end
```

2.4 Experimental Results

2.4.1 Modification to Head Tracking when Turning

In this experiment, we have tried to show that our modification improved the ball tracking and finding when a robot is turning around. Two experiments were carried out in the same environment.

Both experiments involve a robot spinning on the spot to look for the ball and check if it can track the ball properly without having to lose the ball from its vision. We will see in these two experiments that as the ball distance gets bigger, it becomes harder for the robot to successfully track the ball, because the rate of change in the ball heading is much higher if the ball distance is large.

The first experiment was done without the modification which we discussed earlier on. The result for the first experiment is shown in Table 2.2. We can see that the robot cannot track the ball properly when the ball is only 130cm away.

Ball Distance (cm)	Success	Failed	Success Rate
50	28	0	100.0%
80	41	13	75.9%
130	11	56	16.4%

Table 2.2: Tracking when turning without modification

The second experiment was tested with the head tracking modification and the result is shown in Table 2.3. We can obviously see the big improvement from the first experiment result, as the robot can still track the ball properly when the ball distance is 130cm. The distance limit which the robot can track the ball successfully is around 230cm. This distance is nearly a double of the limit shown in the first experiment.

We believe 280cm is a rough estimate of the distance limit which the robot can visually detect the ball, because at this distance, the robot could not detect the ball at all while it was spinning around. Thus, being able to track the ball from the spin with more than 50% of success rate at the distance of 250cm is a reasonable achievement.

Ball Distance (cm)	Success	Failed	Success Rate
50	50	1	98.0%
80	31	0	100.0%
‘ 130	50	1	98.0%
180	55	4	93.2%
230	52	21	71.2%
250	38	29	56.7%

Table 2.3: Tracking when turning with modification

2.5 Further Work

2.5.1 Backing Off and Fighting in Scrum

In the section 2.2.5, we have mentioned that in a scrum situation, our robots back off, because if they cannot see the ball in the scrum, then there is no point in fighting for something which might not be in it. However, if the ball is in the scrum, then we will waste an opportunity to fight for it. To make a reasonably good decision between backing off and fighting for the ball, there is an idea which has not yet been experimented. The idea involves using the world model information as well as teammates' both visual and world model ball information sent via a wireless message.

When an attacker is in a scrum and there is at least one teammate who is near the scrum and can see a ball inside it, then the teammate can send the ball position in global coordinates and the attacker can check whether the ball position is near the attacker, itself. If the attacker thinks the ball is close to itself, then we know that the ball is likely to be in the scrum. Thus, the attacker can decide to fight aggressively in the scrum.

2.5.2 Ball Interception using Ball Velocity

Intercepting a moving ball is a difficult task, because accurate ball velocity estimation is required to do this effectively and reliably. The ball interception, as shown in Figure 2.4, is useful in reaching the ball in time. Instead of walking straight at the ball, a robot calculates a point on the field which could be reached by the robot before the ball reaches first.

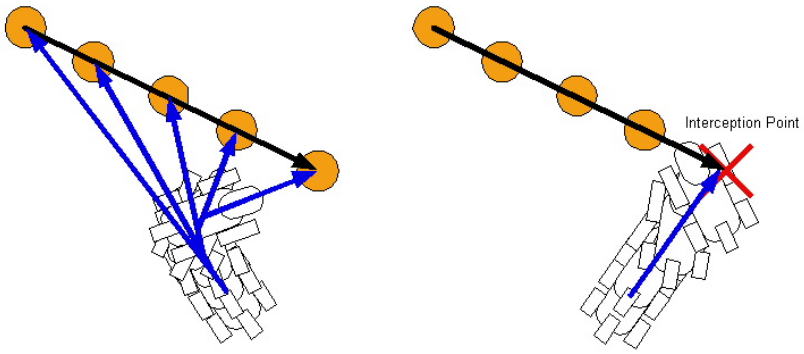

Figure 2.4: Motivation in ball interception using ball velocity

Chapter 3

Ball Grabbing

"Nobu... Can you help me with the Variable Lightening Challenge?"

Andrew Million Sianty (Past midnight, working on his variable lighting challenge. A scary challenge which he was working on!)

3.1 Introduction

In this chapter, we will describe one of our important and crucial skills called ball grabbing. Grab is a skill in which a robot holds a ball using its chin and two front legs. Our attacker's behaviour has used this ball grabbing skill extensively. By having the ball under the robot's chin, kicks can reliably be executed and most importantly the robot is able to have total control and possession of the ball. Thus the robot is unlikely to lose the ball when pushed by other robots compared to not grabbing the ball at all.

In previous years, because there were walls around the field, the ball grabbing behaviour was not a critical factor in the overall rUNSWift's strategy. However, because this year's field does not have any walls around, the grabbing skill plays an important role in keeping the ball inside the field.

It is interesting that there are many different approaches in designing and implementing this skill, and in fact, different teams who had participated in RoboCup 2005 had different approaches in grabbing the ball. Which approach works better depends on the environment that the robot is in. Hence, our ball grabbing skill was optimised for a game environment where the ball tends to be rolling and moving slowly.

In this introduction section, we will describe the problems or issues we have faced when designing and implementing our ball grabbing skill. These problems and issues were resolved with ideas which will be explained in section 3.2.

3.1.1 Speed Control

It is desirable for a robot to grab a ball without slowing down. If the robot slows down to line up for the ball for grabbing, it will risk other robots to get to the ball first. Thus, this leads to losing possession of the ball. This implies the speed of ball grabbing. However, if the robot does not slow down, this will increase the risk of the ball being knocked forward with the robot's chest which results in an unsuccessful grab failure. This implies the reliability of ball grabbing.

The speed and reliability are both important factors in ball grabbing, but these two factors trade off each other. Thus, reasonable balance between these two factors are required to achieve good ball grabbing skill.

3.1.2 Odometry Calibration

Lining up for the ball precisely while walking towards it, is indeed difficult, because odometry calibration is never perfect. When the robot is lining up for the ball, it is required to turn a small amount of usually less than 10 degrees.

Although the calibration is not perfect, this does not mean our odometry calibration is poor. We have, in fact, used both automated and manual odometry calibration in the 2005 rUNSWift architecture to achieve better odometry than what previous years have had.

For more information on odometry calibration, please refer to [W. Chan].

3.1.3 Close Ball Inaccuracy

Nothing is perfect in robotics, because of the non-deterministic environment. In particular, our vision module detects up close balls sometimes incorrectly which results in either smaller or larger balls. Thus, this does well mean distance and heading calculated for the detected up close balls are inaccurate.

Inaccurate distance can mean that the robot may under or over step towards the ball. If the robot under steps, it takes time for the robot to get to the ball. If the robot over steps, it has the risk of knocking the ball out with its chest or front paws. Under stepping affects the speed and over stepping affects reliability of the ball grabbing.

Similar situations can be said for inaccurate heading. Inaccurate heading can mean that the robot may turn either less or more to line up for the ball depending on the error of the ball heading. If the robot turns less, it takes time for the robot to line up. If the robot turns more, it has the risk of knocking the ball out with its front paws, because the ball is too close to the robot. Turning less affects the speed and turning more affects reliability of the ball grabbing.

3.1.4 Grabbing Moving Ball

During the game, the ball is not always still. In fact, it is almost always moving, because it is kicked and softly knocked by the robot. Grabbing a moving ball is more difficult than grabbing a stationary ball, because it has a higher chance of it slipping out from under the robot's chin.

3.2 Concepts and Ideas

3.2.1 Short Step Walk

The actuator control module of our rUNSWift code has been designed in a way so that the robot can only change walk parameters every half step. In other words, during this half step, the robot cannot change the walk parameters. Thus, if the step size is shorter, then, in theory, the robot is more responsive to the walk parameters which are sent from the behaviour module every camera frame.

Short step walk is a feature in the actuator control module which allows the robot to walk in a smaller step size than the complete step size of a walk. In 2005, we have used a walk type called Skelliptical Fast Forward walk which has a half step period of 22. This half step size number implies the number of locus points through which a particular point on the robot's leg travel. Each locus point transition is done in about 8 milliseconds which is called the joint frame. This Skelliptical walk, itself, already has a shorter half step period compared to the Normal walk from 2004 rUNSWift code which had a half step period of 40. Although, the Skelliptical walk is already responsive compared to the Normal walk, we can make use of the short step walk feature to attain much better response.

3.2.2 Side Step Adjustment

During the game, the ball is never stationary. The key in the ball grabbing behaviour is whether we can grab the moving ball reliably or not. This year, the grabbing skill was such an important and critical skill, because by grabbing, we were able to keep the ball inside the field which has no walls around.

The side step adjustment is a new feature introduced in our ball grabbing style this year to increase the reliability of the grab. This side step adjustment is only triggered when a robot is close to the ball and requires a small adjustment to line up with the ball. To correct this small misalignment, we use a side step instead of a turn, because of the following advantages:

1. If the robot is lined up to a desired attack direction, then use of the side step will stay aligned to the direction, because the robot moves perpendicular to the direction. Whereas, if the robot turns to correct this small error, it may end up facing the wrong direction. This is shown in figures 3.1 and 3.2.

2. The side step is well calibrated in walking a small distance compared to turning. Thus, by using the side step, we can actually line up with the ball accurately and faster.

3. Turning near the ball may potentially hit the ball with its front paws.

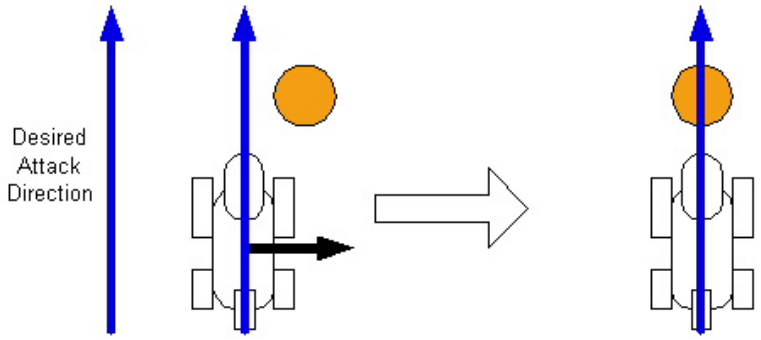

Figure 3.1: Side stepping stays aligned with the desired attack direction

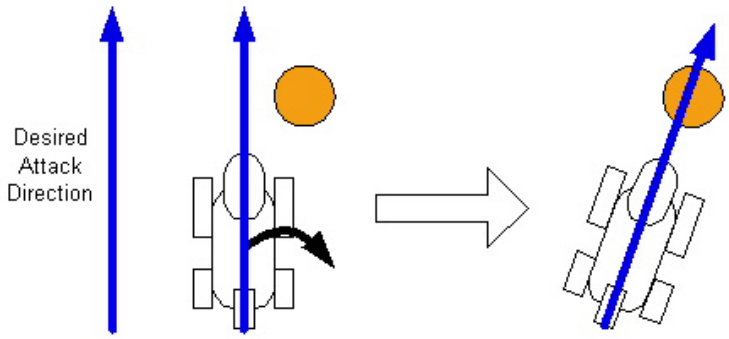

Figure 3.2: Turning misaligns with the desired attack direction

3.2.3 Modification to Walk Command Interface

At every camera frame, the behaviour module sends walk commands to the actuator control module to make the robot's joints move. In previous years,

the behaviour module had been sending step size of forward, side and turn components through a function. In 2005, the behaviour module sends speed and step size of forward, side and turn components, and as a result, it has improved the degree of flexibility in controlling the robot's motion. This was particularly useful for ball grabbing. By being able to specify both speed and step size, the robot can do the following:

1. walk at maximum speed and variable step size
2. walk at variable speed and maximum step size
3. walk at variable speed and variable step size

Below is the function declaration of walk which sends the walk parameters from the behaviour module to the actuator control module every vision frame. This can be thought of as our walk command interface which links the behaviour and actuator control modules. Our walk command interface has the first and second features which are listed above implemented, but not the third feature. We decided not to include this feature in the interface, because we seldom encountered a situation which we wanted the robot to walk at variable speed and variable step size. However, it is still possible to do this by hacking the walk parameters before it gets sent to the actuator control module. The first and second features sufficed most of the situations which we had encountered. The argument cmdType specifies whether the forward, side and turn components use speed (whichstands for 's') or step size (which stands for 'd'). This can be specified separately. The default value of "ssd" implies that forward is set to speed, side is set to speed and turn is set to distance. In this walk command interface, if the command is set to speed, then it will use maximum step size and if it is set to step size, then it will use maximum speed.

def walk(fwd = 0, left = 0, turnCCW = 0, cmdType = "ssd", walkType = SkellipticalWalkWT, minorWalkType = DefaultMWT):

As said earlier, this feature was useful for implementing the ball grabbing skill. One useful example is to specify cmdType value as "ddd". This meant that the robot walks at maximum speed and variable step size for all forward, side and turn components. Therefore, the robot does not need to slow down its speed at all when walking in shorter step size, because we tell the robot to walk at maximum speed. In 2004's walk command interface, there was no way to represent this. Hence, the robot could only walk in shorter step size by slowing down the speed.

3.2.4 Modification to Head Tracking

It takes some time for a robot to move its head over the ball when grabbing it. This time can be shortened by having the head placed forward beforehand. Thus, the robot is only required to slide its head over the ball and open its mouth to lock the ball under its chin. In order to have the head placed forward beforehand, we have modified the head kinematics in the actuator control module. This modification still allows a robot to track a ball with its head leaned forward.

3.2.5 Modification to Ball Edge Detection Algorithm

If the ball is up close, it has been said earlier that the ball is sometimes incorrectly detected which results in either a smaller or larger ball. When this happens the robot does not behave in the way as we expect it to, because the detected ball information is wrong. This situation happens, because our vision module has difficulty in detecting enough ball edges to recognise the correct shape of the ball. The ball edge detection algorithm is written in a way that no false ball edge is detected. The algorithm performs several checks before it verifies the possible ball edges as true ball edges. These checks unfortunately sanity out correct ball edges for the up close ball. As a result, the ball cannot be detected.

To resolve this, we have decided to have a flag which can turn on and off the verification checks in the algorithm, and we only turn the checks off when the ball is up close and the robot is trying to track the ball. Turning off the checks allow more ball edges to be detected on the real ball and since we are only turning them off for the up close ball, we have low risk of detecting false ball edges, because the robot points its head downwards and we seldom detect orange noise in the green field.

Switching the verification checks is only done using the ball distance, however it is interesting to see if the head tilt angle can help in switching the checks. This will be further discussed in section 3.5.2.

3.2.6 Ball Distance Hysteresis

The ball grabbing behaviour itself works in three phases and this is actually shown in Figure 3.3. It shows that there are three different phases in approaching the ball and to determine which phase a robot is in, the ball distance is used.

Since the ball distance estimation fluctuates while the robot is approaching the ball, it is important to have the hysteresis in the ball grabbing behaviour to control the motion smoothly. Otherwise, the robot will flicker between the two phases too rapidly making its grabbing motion unsmooth. In fact, the ball grabbing behaviour is not the only behaviour which uses hysteresis. The ball tracking and finding also uses the hysteresis as well. We must mention here that a well tuned hysteresis will allow us to have a smooth transition between the phases which will result in a seamless motion.

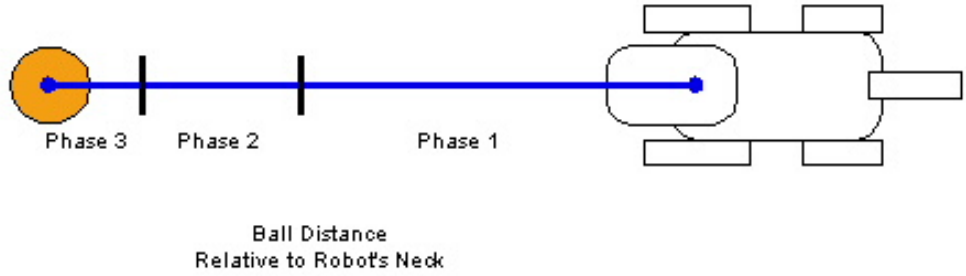

Figure 3.3: Ball distance hysteresis in ball grabbing behaviour

3.2.7 Grabbing Criteria

We have used two different sets of criteria to fire a grab. When one of them is true, then the robot tries to grab a ball.

1.
   - the ball has to be visual
   - the ball distance is within the GRAB_DIST
   - the ball's x value is between -0.5 to 0.5
   - the ball's y value is less than 8
   - the robot has not recently made a big turn
2.
   - the ball has to be visual
   - the ball distance is within the GRAB_DIST
   - the ball heading is between -8 and 8
   - the weighted ball heading is between -8 and 8
   - the difference of the ball heading and the weighted ball heading is between -6 and 6
   - the robot has not recently made a big turn

3.3 Design and Implementation

3.3.1 Top Level

Algorithm 3.1 is the top level function of the ball grabbing behaviour. The grab behaviour does not do anything once, a robot has grabbed the ball. Thus, a behaviour such as the grab dribbling behaviour will takes over after the ball is grabbed.

If a robot can grab a ball from its current position or it has already started grabbing, then call GrabBall(). Whether the robot can grab the ball is determined from the grabbing criteria discussed in section 3.2.7. If the grabbing criteria is not satisfied, then it tries to line up with the ball by calling ApproachBall().

Algorithm 3.1: PerformGrab

Data:
Result: This algorithm is top level of ball grabbing behaviour

```
1  if gIsGrabbed == true then
2  |  continue what we were doing
3  else if gGrabbingCounter > 0
4  OR we can grab a ball from current position then
5  |  r = GrabBall()
6  |  if r == false then
7  |  |  return FAILED
8  |  end
9  else
10 |  ApproachBall()
11 end
12 if gIsGrabbed == true then
13 |  return SUCCESS
14 else
15 |  return EXECUTING
16 end
```

3.3.2 GrabBall

Algorithm 3.2 handles the grab motion. Once it has entered this function, the top level ball grabbing function will keep calling this function until gGrabbingCounter is greater than GRAB_COMPLETE_TIME which is currently

set to 25 vision frames. Thus, for this grab motion to complete, it will take nearly one second. This grab ball function is divided into three sub phases.

1. The first phase is when gGrabbingCounter is less than GRAB_FORWARD_TIME (7 vision frames). It tries to get the head moved forward a little to get ready for th grab.

2. The second phase is when gGrabbingCounter is between GRAB_FORWARD_TIME and GRAB_CHECK_TIME (15 vision frames). This phase tries to move the head forward even more to slide the ball under a robot's chin.

3. The third phase is when gGrabbingCounter is between GRAB_CHECK_TIME and GRAB_COMPLETE_TIME (25 vision frames). This phase tries to check whether a robot can see the ball or not. If the robot can see the ball, then the grab must have failed, because it should be under the robot's chin and it should not be visible, if it was grabbed properly.

After these three phases, it will set gIsGrabbed to true indicating that a robot has successfully grabbed a ball.

Algorithm 3.2: GrabBall

Result:

```
1  increment gGrabbingCounter
2  if gGrabbingCounter < GRAB_FORWARD_TIME then
3  |  set head parameters (pan = 0, tilt = -25, crane = 30)
4  |  close mouth
5  else if gGrabbingCounter < GRAB_CHECK_TIME then
6  |  set head parameters (pan = 0, tilt = -45, crane = 50)
7  |  close mouth
8  else if gGrabbingCounter < GRAB_COMPLETE_TIME then
9  |  set head parameters (pan = 0, tilt = -50, crane = 40)
10 |  open mouth
11 |  if the ball is visible then
12 |  |  return FAILED
13 |  end
14 else
15 |  set head parameters (pan = 0, tilt = -45, crane = 50)
16 |  open mouth
17 |  gIsGrabbed = true
18 |  gGrabbingCounter = 0
19 |  return SUCCESS
20 end
21 return EXECUTING
```

3.3.3 ApproachBall

How we approach the ball is divided in three main phases. These phases are as follows:

1. The first phase is when a robot is still far away from the ball. In this phase, we just call PerformTrackAndFindBall() which belongs to the ball tracking and finding behaviour in section 2.3.1.

2. The second phase is when the ball distance is within CLOSE_DISTANCE, but not within REALLY_CLOSE_DISTANCE. In this phase, the robot walks towards the ball using the head tracking modification discussed in section 3.2.4.

3. The third phase is when the ball distance is less than REALLY_CLOSE_DISTANCE. In this phase, the robot uses the side step adjustment if the line up adjustment is small, otherwise it will turn to line up with the ball.

The first "if" statement in Algorithm 3.3 checks whether the ball distance is less than CLOSE_DISTANCE. If it is, we set the current vision frame to gLastCloseFrame which will be used as a hysteresis for the phase transition between the first and the second phases. There is another global variable gLastReallyCloseFrame which used as a transition hysteresis between the second and the third phases. This distance hysteresis was mentioned in section 3.2.6.

Algorithm 3.3: ApproachBall

```
   Data:
   Result:
 1 ballD = ball distance
 2 ballX = ball's local X position
 3 if ballD < CLOSE_DISTANCE
 4 AND lostBall < LOST_BALL_GPS then
 5 |  gLastCloseFrame = currentVisionFrame
 6 end
 7 if (currentVisionFrame - gLastCloseFrame) < 5 then
 8 |  if ballD < REALLY_CLOSE_DISTANCE then
 9 |  |  gLastReallyCloseFrame = current_camera_frame
10 |  end
11 |  if (currentVisionFrame - gLastReallyCloseFrame) < 5 then
12 |  |  if abs(ballX) < 8 then
13 |  |  |  side step to line up with the ball
14 |  |  else
15 |  |  |  turn only to line up with the ball
16 |  |  end
17 |  else
18 |  |  walk towards the ball
19 |  end
20 |  track the ball with modification
21 else
22 |  PerformTrackAndFindBall()
23 end
```

3.4 Experimental Results

3.4.1 In Lab

The experiments for the ball grabbing behaviour was carried out jointly with the attacker behaviour experiments. Section 6.6.1 describes the experiments we have carried out in detail. For now, let us assume that for our experiment, we have run a behaviour which includes the ball grabbing behaviour and we have counted how many times it succeeded, failed and hesitated in grabbing a ball in 3 minutes. This was of course carried out multiple times as seen on Table 3.1.

Sample	Grab			Grab Success Rate
	Success	Failed	Hesitated	
1	12	1	1	85.7%
2	12	3	4	63.2%
3	13	1	1	86.7%
4	17	5	2	70.8%
5	13	1	2	81.3%
6	12	4	2	66.7%
7	12	3	1	75.0%

Table 3.1: Attacker behaviour without a goalie for 3 minutes

3.4.2 In Competition (vs NUBots)

Tables 3.3 and 3.3 shows the success and fail counts of the grab in the game against NUBots, the semi-final. We can see from the two tables that NUBots have tried to grabbed the ball more than we did. They have tried to grabbed the ball 50 times in the first half and succeeded 28 times in the grab. In second half, the number of the total attempts in the grab was 51 times and succeeded 35 times. The success rate for NUBots is lower compared to our success rate. However, the number of the grab succeeded is much higher than ours. This meant that they had the possession of the ball much more than we did and had more chance of scoring goals. Thus, they won the semi-final.

Team	Success	Failed	Grab Success Rate
rUNSWift	18	11	62.1%
NUBots	30	20	60.0%

Table 3.2: Grab success rate in the first half

Team	Success	Failed	Grab Success Rate
rUNSWift	24	8	75.0%
NUBots	35	16	68.6%

Table 3.3: Grab success rate in the second half

3.5 Further Work

3.5.1 Removing Hestitation Before Grab

Our ball grabbing behaviour sometimes hesitates the robot to grab the ball. As shown in Table 3.1, there are a couple of times when the robot hesitated and did not grab the ball successfully. We have also seen this from the games we had in the competition. The hesitation can delay the whole process of our attack and can potentially shut down the flow of our attack. This is also one of the big factors in losing in possession of the ball. In future, the removing hesitation must be removed to make the ball grabbing skill effective and reliable.

3.5.2 Using Head Parameters to Switch On and Off Ball Edge Detection Algorithm

In section 3.2.5, we have talked about the modification of the ball edge detection algorithm. To trigger the switching between the modified and the original versions of the algorithm, we have mentioned that the ball distance is used.

In this section, we would like to mention one more possible way of triggering this switch. That is by using head parameter values to determine if a robot is looking down or looking straight to track the ball. If the robot is tracking the close distance ball, it would look down and this is the situation when we want to have switch on the modified version as explained in the previous section. Thus, trying both methods to trigger the switch may reliably track

the ball well. This might need to be experimented in future.

Chapter 4

Grab Dribbling

"What my friend is trying to say is ..."

Andrew Million Sianty (He almost always started off with this line, when he wanted to clarify what I said to other people!)

4.1 Introduction

In 2000, grab dribble was used by the UNSW United [UNSW United 2000]. Since then, they were able to grab a ball and then turn to shoot into a goal reliably using the best gap detection. In 2004 RoboCup Portugal, University of Technology, Sydney, UTS Unleashed! had demonstrated a dodge behaviour which a robot avoids an opponent with a ball grabbed under its chin, and we believe that they were the first one to introduce this technique in the 4-legged league [MISC : UTS Unleashed!].

After the year 2000, when the 3 second ball holding rule was introduced, rUNSWift has mostly used a strategy which involves robots running swiftly to the ball first and pushing it up the field with paw kicks to outrun opponents. In fact, this simple strategy worked effectively well, because every year rUNSWift always improved their walk speed to be one of the fastest team in the competition and also, there were walls around the field. The walls meant that they did not have to worry about the ball going out of the field. However, this year, since the walls have been removed from the field, we, rUNSWift felt the necessity in using grabbing and dribbling the ball extensively. Due to the extensive use of grabbing and dribbling the ball, the overall strategy has no doubt changed dramatically and has made a significant contribution to adapt to the field with no walls.

Surprisingly, at the 2005 RoboCup Osaka, NUBots and rUNSWift were the only two teams that had exploited this skill and were successful in having possession of the ball in most of the matches played. The German team who had the successful victory in this year's RoboCup competition had also included this grab dribbling behaviour in the middle of the competition.

Although this skill is useful in handling the ball, it cannot be executed for more than 3 seconds according to the rules. If a robot grabs for more than 3 seconds, it will be penalised and taken out from the field for half a minute. Thus, we have put in a grabbing timer in the code to avoid this penalty. When the grab timer reaches around 2 seconds, a robot will start executing a kick or releasing the ball. This will be described later in section 4.2.6.

This chapter describes the ideas and internal workings behind the grab dribbling behaviour. In section 4.2, we will discuss concepts and ideas behind the grab dribble. Ideas include dribbling to line up to the best gap of the target goal, dodging around obstacles, handling when a robot is near the field edge and ball kicking strategy. Then, it will be followed by the detailed design

and implementation of this grab dribble behaviour. Lastly, we will conclude this chapter with several improvements which can be made to the current grab dribbling skill.

4.2 Concepts and Ideas

4.2.1 Grab Dribble Walk Type

Dribbling with a ball under a robot's chin is reliable only if it has its front feet placed forward. Otherwise, the ball could roll out while walking forward or turning around. Thus, a new walk type had to be created specifically to be used when the robot grabbed the ball, because our normal walk did not have the front feet placed forward. This new walk type walks forward slightly slower than the walk type which we used for non-grab related purposes. This drop in speed is caused by how the front feet are placed. However, with this grab dribble walk type, the robot can turn as fast as the normal walk type.

We may want to put a figure here...

4.2.2 Grab Dribble Types

In grab dribbling behaviour, we have two dribbling styles. One is to dribble towards the target goal via visual detection of the best gap, and the other is to dribble towards the desired attack direction. In short, these two styles are referred to as PerformToTargetGoal and PerformToDAD respectively.

PerformToTargetGoal and PerformToDAD are both described in detail in section 4.3.

4.2.3 Lining Up To Best Gap

Since our rUNSWift philosophy in playing robot soccer has always been "attack the ball with speed", lining up to the best gap of the goal must also be done swiftly without stopping. Our approach was by dribbling towards the gap rather than turning on the spot to line up. The dribbling towards the gap means applying both forward and turn components in the walk, and in some cases, side component is also applied. The basic idea is that when dribbling with the ball, the robot must be moving closer to the gap. Using dribble has several advantages over just turning on the spot:

1. The best gap angle widens as the robot approaches towards the gap. This is shown in 4.1. Assuming that the robot takes Gap 1 as the best gap, the gap angle is much wider compared to the gap angle at the original position. Although Gap 2's angle gets narrower, since the robot is locked into the other gap, we do not have to worry about it.

2. Dribbling forward avoids an opponent coming in from the side or behind implicitly. If a robot chooses to just turn on the spot to line up, then an opponent has enough time to block the robot before it shoots. Whereas, if the robot moves forward while lining up towards the best gap, it has an effect of avoiding the opponent as well.

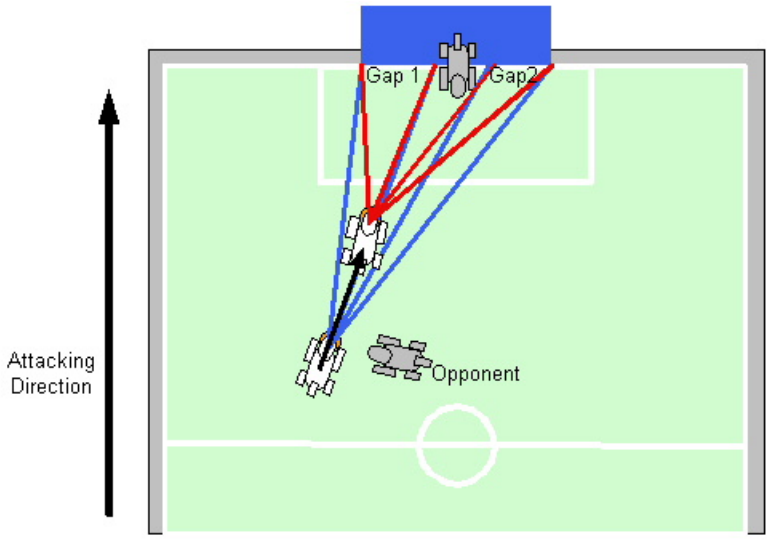

Figure 4.1: Comparing dribbling and turning towards the best gap

Although dribbling the ball forward has implicit effect in avoiding an opponent coming in from the side or behind, it neither avoids or dodges an opponent which is right in front of the robot. In the next section, we will describe how we extended the grab dribbling behaviour to allow the robot to dodge around an opponent which is in front.

Later in the chapter, we will see that lining up to the best gap using the grab dribble instead of turning only gives a far better result when scoring goals. This will be discussed in section 4.4.1.

4.2.4 Dodging Around Obstacles

Dodging is an important feature in the grab dribbling skill, because for a robot to shoot into the best gap or do anything useful, it requires a clear view and not to be surrounded by opponents or obstacles. We use two different type of information to trigger our dodging behaviour. One is by using obstacle information gathered from the vision module and the other is by using two front feet sensors of a robot.

It must be mentioned that there are two different obstacle tracking systems running concurrently inside the vision module. One system tracks the obstacles by using visual information only, thus, information gets refreshed every vision frame. The other one tracks obstacles by caching visual obstacles and decaying them as time goes by. For more information on ideas behind our obstacle tracking systems, please refer to [J. Shammay].

For our dodging behaviour, we used visual obstacle information, not cached obstacles. Because we want to know whether there are any obstacles in front of a robot, we specify a box which has relative coordinates of (-25,20), (-25,100), (25,20) and (25,100) as shown in Figure 4.2 and check if there are enough obstacle features inside it.

Thus, the basic idea is that if the amount of obstacle features are greater than the threshold we specified, then the dodging behaviour is triggered. However, we do not want a robot to start dodging every time there is an obstacle in front of it, because it may be facing the wrong direction. It must be triggered only when a robot is lined up to some desired direction. This check is also included in our dodge trigger criteria.

When a dodge is triggered by the visual obstacles, dodging direction is decided on several factors. Firstly, this is decided on whether a robot can see the best gap of the goal. If it can see the gap then, it will dodge towards the side which the gap angle is bigger as shown in Figure 4.3. To determine which side of the gap has a bigger angle, the absolute value of both minimum and maximum gap headings are compared. The minimum and maximum headings are both local heading relative to the robot's neck. If the absolute minimum heading is greater than the absolute maximum heading, then the left direction is taken. Otherwise, right is chosen. In the case of Figure 4.3, the absolute maximum heading is greater than the absolute minimum heading. Thus, it shows that the robot should dodge towards its right.

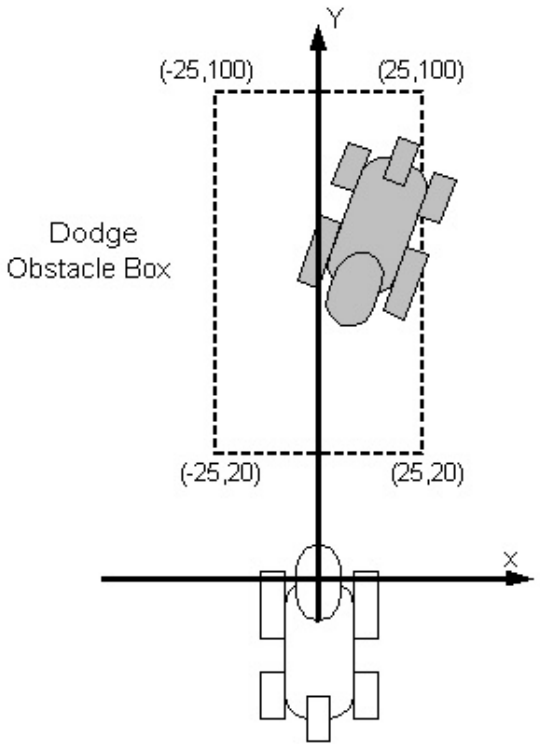

Figure 4.2: Obstacles inside the box is checked before dodging is triggered

If the robot cannot see the gap, then the direction is decided on which half the robot is in. If the robot is in the offensive half, then it will dodge towards the line drawn between the goal and the centre circle as shown in Figure 4.4. If the robot is in the defensive half, then it will dodge away from the line as shown in Figure 4.5.

Because our walking stance does not hit the two front feet sensors, we can reliably use them to trigger the dodging behaviour. The idea of using the front feet sensors was taken from the back off mechanism (section 6.3.2) used by non-attacker field players. If the left front feet sensor detects an obstacle, then the robot should dodge right. If the right front feet sensor detects an obstacle then it dodges to its left. If both sensors get fired, then it moves backward.

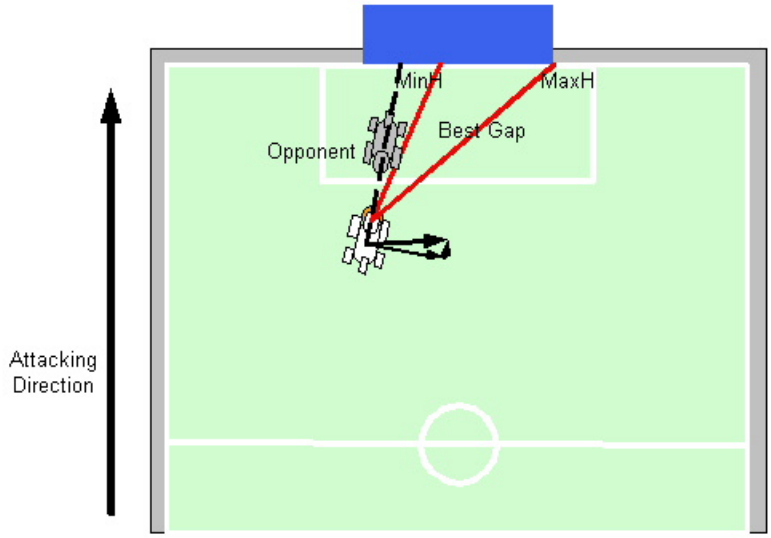

Figure 4.3: Dodging direction when the best gap is detected

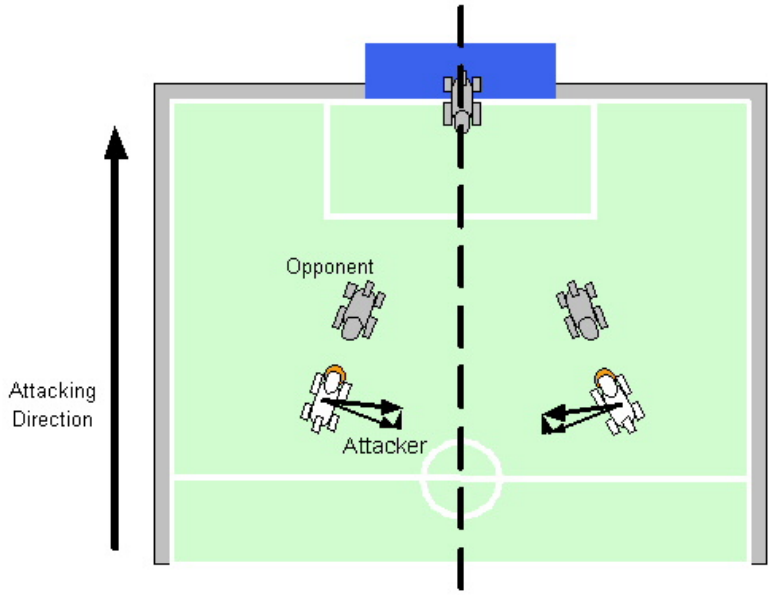

Figure 4.4: Dodging direction in the offensive half when the best gap is not detected

4.2.5 Edge Behaviour

Edge behaviour is a behaviour which a robot moves away from the field edge if it is close to the field edge. This behaviour is performed using backward

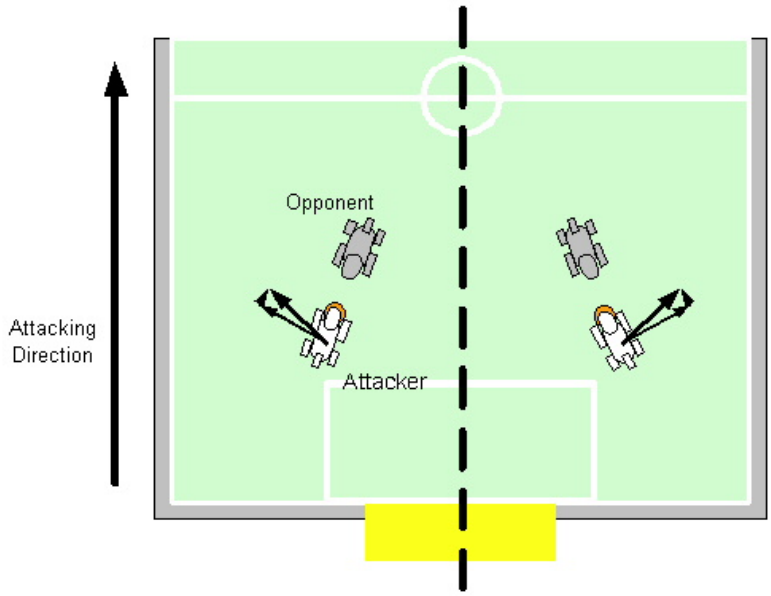

Figure 4.5: Dodging direction in the defensive half when the best gap is not detected

walk, side step walk or diagonal walk, and it does not use any turn components in the walk, because turning with the ball has a chance of the ball crossing out of the field lines if a robot is too close to the field edges. The diagram on the left of Figure 4.6 shows the normal behaviour without the edge behaviour. The robot grab turns in an anti-clockwise direction and this causes the ball to cross over the field line. Whereas, the right diagram of the same figure shows the ball does not go out of the field, because the robot walks backward first and then turns around.

In addition, as our ball grabbing skill is designed in such a way that a robot continues to walk a few steps forward just after it grabs the ball, this edge behaviour is particularly useful in avoiding the situation described above.

As mentioned in the beginning of this section, the edge behaviour is performed using backward walk, side step walk or diagonal walk, and this is chosen by a robot's current position and heading. The criteria used to choose between backward, side step and diagonal walks are described below. However, before we discuss the criteria, we must define what it is meant by "near the field edge". "near the field edge" is defined as when a robot is within 60cm from one of the field edges. There are four field edges known as top,

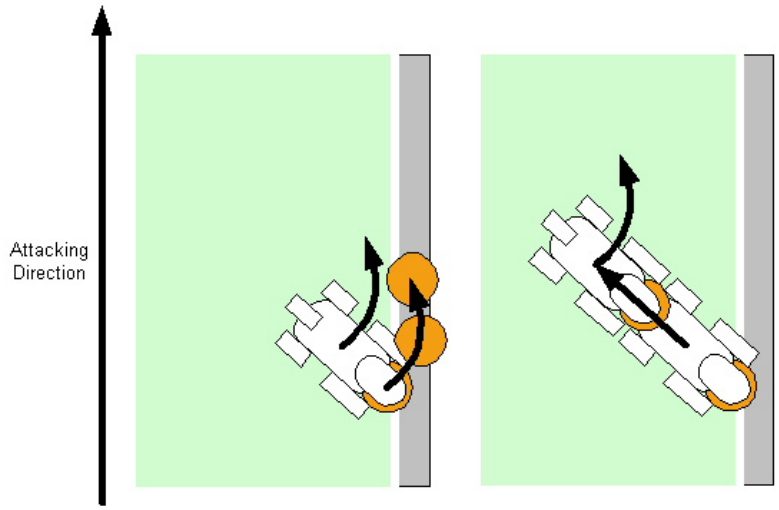

Figure 4.6: Motivation of edge behaviour

left, right and bottom. Top edge is the field edge which has the opponent's goal. Left edge is the edge on the left hand side when observed into the direction of the opponent's goal. Right edge is the other side of left edge, and lastly, bottom edge is the side that has the own goal.

A robot steps backward if ONE of the following criteria is satisfied:

- a robot is near the left edge (L-Edge) and 130 <= global self heading (selfH) <= 230
- a robot is near the right edge (R-Edge)and (0 <= selfH <= 50 or selfH <= 310)
- a robot is near the bottom edge (B-Edge) and 220 <= selfH <= 320
- a robot is near the top edge (T-Edge) and 40 <= selfH <= 140 and target goal is not visible

A robot steps left if ONE of the following criteria is satisfied:

- a robot is near the L-Edge and 80 <= selfH <= 130
- a robot is near the R-Edge and 260 <= selfH <= 310
- a robot is near the B-Edge and 170 <= selfH <= 220

- a robot is near the T-Edge and (0 <= selfH <= 40 or selfH <= 350) and target goal is not visible

A robot steps right if ONE of the following criteria is satisfied:

- a robot is near the L-Edge and 220 <= selfH <= 280
- a robot is near the R-Edge and 50 <= selfH <= 100
- a robot is near the B-Edge and (320 <= selfH <= 360 or selfH <= 10)
- a robot is near the T-Edge and 140 <= selfH <= 190 and target goal is not visible

A robot steps diagonally if ALL of the following criteria are satisfied:

- (a robot is near the L-Edge and 80 <= selfH <= 130) or (a robot is near the R-Edge and 50 <= selfH <= 100)
- robot's y-position < FIELD_LENGTH * 0.8

4.2.6 Time Elapsed Action

Time elapsed action is the behaviour which is executed when the 3 seconds grabbing timer is about to expire. Firstly, the time elapsed action selects an appropriate action to execute. These actions include shooting into the best gap of the goal, kicking up the field, kicking through a gap between obstacles and releasing the ball. The action selection is actually represented in a decision tree and we will see that the idea of this time elapsed action selection is based on the attacker action selection which will be described in chapter 5. The time elapsed action selection is dependant on the following factors:

- Current position of a robot on the field.
- Whether a robot can visually detect its target goal or not.
- Whether a robot can visually detect its own goal or not.
- The best gap heading of the target goal, if a robot can see it.
- The number of visual obstacle features detected in front of a robot.

4.2.7 Ball Releasing

If kicking a ball is not the appropriate action to take, then it is released from a robot's chin after around 3 seconds. The ball releasing is a technique used to roll the ball forward and still allow the robot to continue walking towards it without stopping.

4.3 Design and Implementation

There are several global variables used in the algorithms we are about to describe. These global variables are as follows:

gForceTimeElapsedAction is a boolean variable which indicates whether time elapsed action has been activated or not.

gForceToTargetGoal is a boolean variable which indicates whether PerformToTargetGoal should be invoked from PerformToDAD or not.

gEdgeCounter is an integer variable which counts the duration of the edge behaviour.

gDodgeCounter is an integer variable which counts the duration of the dodge behaviour.

gGapAimCounter is an integer variable which counts total number of frames when a robot has lined up.

There are several constants which need to be explained as well:

MIN_TIME_TO_DRIBBLE is currently set to 1900 milliseconds.

MAX_TIME_TO_DRIBBLE is currently set to 3000 milliseconds.

MIN_GAP_AIM is current set to 3 vision frames.

4.3.1 PerformToTargetGoal

Algorithms 4.1 and 4.2 shows the detail execution of the "PerformToTargetGoal" functionality. The first three "if" statements check if one of edge, dodge and time elapsed action behaviour is activating. If one of them is currently running, then it continues to run that behaviour until it finishes. The next two "if" statements check if a robot should stop dribbling and execute either kick or release behaviour. PerformTimeElapsedAction() is called with an argument OFFENSIVE, if the grabbing timer exceeds

MIN_TIME_TO_DRIBBLE (1.9 seconds) as well as a robot has not turned with the ball in the last 15 vision frames. If the grabbing timer exceeds MAX_TIME_TO_DRIBBLE (3 seconds), then the robot releases the ball immediately to avoid the 3 second ball holding penalty.

In the next "if" statement, it checks whether the edge behaviour needs to be executed for the first time or not. This part is where the edge behaviour first gets triggered and it will get executed for several vision frames in the first block of "if" statements explained above.

After these number of checks, the robot will try to line up with the best gap if it can visually detect the target goal. This behaviour calls PerformToTimeElapsedAction() with an argument OFFENSIVE independent of the grabbing timer as well. If the total number of frames which we are lined up with the best gap is greater than MIN_GAP_AIM, then the behaviour decides to trigger the time elapsed action. As an additional note, the total number of lined up frames does not have be consecutive. While the robot is lining up with the best gap, if it needs to dodge an obstacle, the dodge behaviour will get triggered.

If the robot cannot see the target goal, but has seen it in the last 15 vision frames, then it would walk straight hoping it would pick the target. While walking straight, if it needs to dodge, it will trigger the dodge behaviour as well.

If the robot cannot see the target goal and has not seen it recently, then it turns towards the target goal using world model information of the robot, itself.

Algorithm 4.1: PerformToTargetGoal

```
1  if gEdgeCounter > 0 and PerformEdge() == true then
2  |  return true
3  end
4  if gDodgeCounter > 0 and PerformDodge() == true then
5  |  return true
6  end
7  if gForceTimeElapsedAction == true then
8  |  return PerformTimeElapsedAction()
9  end
10 if grabbing timer >= MIN_TIME_TO_DRIBBLE
11 AND the robot has not turned with the ball in the last 15 vision frames
   then
12 |  gForceTimeElapse = true
13 |  return PerformTimeElapsedAction(OFFENSIVE)
14 end
15 if grabbing timer >= MAX_TIME_TO_DRIBBLE then
16 |  gForceTimeElapse = true
17 |  return PerformTimeElapsedAction(RELEASE)
18 end
19 if PerformEdge() == true then
20 |  return true
21 end
```

Algorithm 4.2: PerformToTargetGoal-Continue

```
1  if the target goal is visible then
2  |  if a robot is lined up to the best gap then
3  |  |  increment gGapAimCounter
4  |  end
5  |  if a robot is lined up to the best gap
6  |  AND gGapAimCounter >= MIN_GAP_AIM
7  |  AND step is completed then
8  |  |  gForceTimeElapsedAction = true
9  |  |  return PerformTimeElapsedAction(OFFENSIVE)
10 |  else if PerformDodge() == true then
11 |  |  return true
12 |  else
13 |  |  dribble to line up to the best gap
14 |  end
15 else if the target goal was seen in the last 15 vision frames then
16 |  if PerformDodge() == true then
17 |  |  return true
18 |  end
19 |  dribble straight
20 else
21 |  turn to towards GPS target goal direction
22 end
```

4.3.2 PerformToDAD

PerformToDAD() is similar to PerformToTargetGoal() which was described in section 4.3.1. The first four "if" statements check if one of edge, dodge and time elapsed action, perform to target goal behaviour is activating. If one of them is currently running, then it continues to run that behaviour until it finishes.

Then, if a robot can visually detect the target goal and is in the offensive third, then PerformToDAD() switches to PerformToTargetGoal(), because once the target goal is detected, PerformToTargetGoal() can do the job better. The variable, gForceToTargetGoal is set to true, so that next time we come back to this function, PerformToTargetGoal() will be invoked in the first part of the check.

When the grabbing timer is greater than MAX_TIME_TO_DRIBBLE which is currently set to 3 seconds, the robot will immediately selects time elapsed action with an argument RELEASE to release the ball as quickly as possible.

The edge behaviour is triggered after checking the grabbing timer and if the edge behaviour is not triggered, then the robot will turn towards the DAD specified and start dribbling.

Algorithm 4.3: PerformToDAD

```
1  if gEdgeCounter > 0 and PerformEdge() == true then
2  |  return true
3  end
4  if gDodgeCounter > 0 and PerformDodge() == true then
5  |  return true
6  end
7  if gForceTimeElapsedAction == true then
8  |  return performTimeElapsedAction()
9  end
10 if gForceToTargetGoal == true then
11 |  return performToTargetGoal()
12 end
13 if gForceTimeElapse == false
14 AND the target goal is visible
15 AND selfY ¿ FIELD_LENGTH * 0.6 then
16 |  gForceToTargetGoal = True
17 |  return performToTargetGoal()
18 end
19 if grabbing timer >= MAX_TIME_TO_DRIBBLE then
20 |  gForceTimeElapse = true
21 |  return performTimeElapsedAction(RELEASE)
22 end
23 if PerformEdge() == true then
24 |  return true
25 end
```

Algorithm 4.4: PerformToDAD-Continue

```
1  turnCCW = DAD - a robot's global heading
2  if abs(turnCCW) <= 5 then
3  |  if PerformDodge() == true then
4  |  |  return true
5  |  end
6  |  if grabbing timer >= minTimeToDribble then
7  |  |  if a robot's y position < FIELD_LENGTH * 0.4 then
8  |  |  |  gForceTimeElapse = true
9  |  |  |  performTimeElapsedAction(DEFENSIVE)
10 |  |  else
11 |  |  |  gForceTimeElapse = true
12 |  |  |  performTimeElapsedAction(MIDFIELD)
13 |  |
14 |  end
15 end
16 dribble towards the DAD
```

4.3.3 PerformDodge

The dodge behaviour is already explained in section 4.2.4. The algorithm firstly checks if the dodge counter is greater than 0. If it is greater than 0, this means it has already triggered the dodge. Thus, we keep dodging in the same direction.

Then, we check if the feet sensors detected an obstacle. If it it did, then it will decide an appropriate direction to dodge and start executing. Otherwise, it tries to look for visual obstacle features and if this obstacle feature count is above the threshold, then it will dodge towards the direction which we have explained in section 4.2.4. The dodge behaviour, iteslf will only get executed for 15 vision frames which is half a second.

Algorithm 4.5: PerformDodge

```
1  if gDodgeCounter > 0 then
2  |  decrement gDodgeCounter
3  |  continue dodging in the same direction
4  |  return true
5  else if a robot's feet sensors detect that it is stuck then
6  |  if left and right sensors return a robot is stuck then
7  |  |  dribble backwards
8  |  else if left sensor returns a robot is stuck then
9  |  |  dribble towards right
10 |  else
11 |  |  dribble towards left
12 |  end
13 |  gDodgeCounter = 15
14 |  return true
15
```

Algorithm 4.6: PerformDodge-Continue

```
1  else if visual obstacle features detected are above the threshold then
2      if the target goal is visible and there is a gap in the goal then
3          if the best gap is more towards the left then
4              dribble towards forward left direction
5          else
6              dribble towards forward right direction
7          end
8      else if a robot is in an offensive half then
9          if a robot is on the left hand side of the field then
10             dribble towards forward right direction
11         else
12             dribble towards forward left direction
13         end
14     else
15         if a robot is on the left hand side of the field then
16             dribble towards forward left direction
17         else
18             dribble towards forward right direction
19         end
20     end
21     gDodgeCounter = 15
22     return true
23 end
24 return false
```

4.3.4 PerformEdge

The algorithm of the edge behaviour is rather simple, if we already know the trigger criteria which we have explained in section 4.2.5. For all the trigger criteria, please refer back to section 4.2.5. If the one of the four edge behaviour trigger criteria satisfies, then the robot will continue stepping in the chosen direction for 15 vision frames (half a second). However, the edge behaviour will only be triggered again after 90 vision frames. This means that the edge behaviour can only be executed at most once in one grab.

Algorithm 4.7: PerformEdge

```
1  if gEdgeCounter > 0 then
2  |  gEdgeCounter -= 1
3  |  gLastEdgeFrame = currentVisionFrame
4  |  continue stepping towards the same direction
5  else if currentVisionFrame - gLastEdgeFrame < 90 then
6  |  return false
7  else if step back criteria satisfies then
8  |  gEdgeCounter = 15
9  |  gLastEdgeFrame = currentVisionFrame
10 |  step back
11 else if step left criteria satisfies then
12 |  gEdgeCounter = 15
13 |  gLastEdgeFrame = currentVisionFrame
14 |  step left
15 else if step right criteria satisfies then
16 |  gEdgeCounter = 15
17 |  gLastEdgeFrame = currentVisionFrame
18 |  step right
19 else if step diagonal criteria satisfies then
20 |  gEdgeCounter = 15
21 |  gLastEdgeFrame = currentVisionFrame
22 |  step diagonal
23 else
24 |  return false
25 end
26 return true
```

4.3.5 PerformTimeElapsedAction

The algorithms in this section explains the time elapsed action behaviour which was discussed in section 4.2.6. Firstly, the algorithm checks if an action was selected previously. If it was, then it continues to do the same action until it finishes. After a robot finished executing the action, it will reset the grab status to "not grabbed". So other appropriate behaviour like the ball tracking and finding behaviour (chapter 2) or the ball grabbing behaviour (chapter 3) can take over.

Then, the algorithm checks whether an argument is OFFENSIVE or not.

OFFENSIVE implies that a robot is in the offensive third and want to shoot into the best gap of the target goal visually rather than using the world model and the DAD.

If the argument given is MIDDLE, then a robot must be in the middle third of the field. In this third, a robot should try to avoid its own goal, if it can see it, and it should also try to kick the ball through the gap between obstacles, if there are obstacles in front. Otherwise, it will just release and keep chasing after the ball. If the argument is DEFENSIVE, the decision is very similar to when the argument given is MIDDLE. The only difference is that MIDDLE selects release behaviour and DEFENSIVE selects NUBot's kick for the "otherwise" case.

Algorithm 4.8: PerformTimeElapsedAction

```
1  if an action is chosen already then
2  |  continue doing the same action until it finishes if the selected
   |  action has finished then
3  |  |  reset the grab status to "not grabbed"
4  |  end
5  else if an argument is OFFENSIVE then
6  |  if the target goal is visible then
7  |  |  if a robot is lined up to the best gap of the goal then
8  |  |  |  action = HEAD_TAP
9  |  |  else if the best gap of the goal is on the left hand side (gap
   |  |  heading >= 45) then
10 |  |  |  action = UPENN_RIGHT
11 |  |  else if the best gap of the goal is on the right hand side (gap
   |  |  heading <= -45) then
12 |  |  |  action = UPENN_LEFT
13 |  |  else
14 |  |  |  action = RELEASE
15 |  |  end
16 |  else
17 |  |  return RELEASE
18 |  end
19
```

Algorithm 4.9: PerformTimeElapsedAction-Continue

```
else if an argument is MIDDLE then
    if the own goal is visible then
        action = AVOID_OWN_GOAL
    else if there are obstacles in front of a robot and there is a good
    gap between obstacles to kick the ball through then
        action = GAP_KICK
    else
        action = RELEASE
    end
else if an argument is DEFENSIVE then
    if the own goal is visible then
        action = AVOID_OWN_GOAL
    else if there are obstacles in front of a robot and there is a good
    gap between obstacles to kick the ball through then
        action = GAP_KICK
    else
        action = NUBOT_KICK
    end
else
    action = RELEASE
end
perform "action"
```

4.4 Experimental Results

4.4.1 Lining Up To Best Gap (In Lab)

In this section, some experiments were carried out to see which approach works better in lining up to the best gap of the target goal. There were two experiments and both were done in the lab. The first experiment used the turn only to line up and the second experiment used the dribble to line up to the best gap.

The experiments consisted of one stationary robot with a red uniform which is placed in front of the target yellow goal and the ball placed just outside the goal box. A robot grabs the ball and try to line up using one of the two approaches and shoot when it thinks it lined up. We have enforced the 3 second ball holding rule, so it would not shoot, if it could not line up with

the best gap in 3 seconds. The ball is placed back to the original position after each run.

We have counted how many times each approach had scored, missed shooting and missed lining up and we have run each experiment several times. The results of these two experiments are shown in Tables 4.1 and 4.2.

Table 4.1 shows its success rate was 25.9% which is nearly a half of the success rate shown in Table 4.2. Thus, lining up with the turn only performs much worse than the dribble. The first table also shows that it turned past the goal frequently. Whereas, the second table shows that the dribble did not turned enough before the 3 second timer expired. Lining up with the dribble also showed that it kicked the ball at the centre of the target goal where the stationary robot was standing when missed the kick. If we can avoid kicking into the stationary robot, but kick into the gap next to it, then we may possibly be able to improve the line up using the dribble.

Success	Missed			Turned		Success Rate
	Left	Centre	Right	Less	Far	
14	9	10	6	0	15	25.9%

Table 4.1: Grab turn kick with gap lock

Success	Missed			Turned		Success Rate
	Left	Centre	Right	Less	Far	
29	4	12	0	9	0	53.7%

Table 4.2: Grab dribble kick with gap lock

4.4.2 Dodging Around Obstacles

This experiment was carried out to check if the dodge behaviour can actually dodge opponents. To simply the experiment, we have tried to dodge around a stationary robot with a red uniform on. The ball was placed 40cm and 50cm away from the stationary robot. In each run, a robot tries to grab the ball and grab dribble straight into the stationary robot. If it think it needs to dodge, the dodge behaviour gets triggered. We counted the number of

times it succeeded in dodging, succeeded in triggering the dodge, but locked up with the stationary robot (thus, failing the overall dodge), and failed to trigger the dodge. The results for the experiments are shown in Table 4.3. The table shows that the dodge behaviour works better when the distance between the ball and the stationary robot is 50cm and its success rate is 71.7% which is actually 11% better than the other. This difference may be caused, because our vision module cannot detect the obstacles within 50cm well.

Distance (cm)	Success	Dodge Failed	Trigger Failed	Success Rate
40	32	8	14	59.3%
50	38	10	5	71.7%

Table 4.3: Dodging stationary obstacles

4.5 Further Work

4.5.1 Obstacle Avoidance using Distance Sensor

The NUBots had used the infra-red distance sensor to dodge around opponents and they have shown in the competition that this method works effectively as well. This distance sensor might work well in the close distance obstacles which we had a problem with (as shown in section 4.4.2).

In future, we might include the use of the distance sensor in our current dodge trigger criteria or modify the criteria to only trigger from the readings we get from the distance sensor rather than visual obstacles. To decide which approach is better, we need to have an experiment to observe the outcome.

Chapter 5

Attacker Action Selection

"U Pan Cake"

- 2004 rUNSWift Code (May be that was how some of the 2004 robocuppers were calling Upenn Kick as!?)

5.1 Introduction

This chapter describes how an attacker behaviour chooses actions during games. We refer this decision making mechanism as attacker action selection. Actions mainly include dribbling, kicking, grabbing and getting behind the ball.

Throughout this chapter, we will refer to attacker action selection to action selection for convenience.

5.2 Concepts and Ideas

5.2.1 Decision Tree

Our attacker action selection is represented as one big decision tree which depends on several factors. These factors include:

1. Current position of a ball on the field. This is discussed in detail in section 5.2.2.
2. Current position of the attacker on the field.
3. Distance and heading of the ball relative to the attacker.
4. Whether the attacker has grabbed the ball or not.
5. Whether the ball is contested or not. The contested ball situation is explained in detail in section 5.2.5.

At anytime, the attacker can make a query to the action selection for the desired action to take, and the selection returns the action type, desired attack direction and action activate distance.

Action types include:

Dribble - The robot walks straight into the ball and its chest bumps the ball forward.

Paw kick - It kicks the ball with its paw while walking. This is very efficient and fast, because it does not require the robot to stop to execute the kick.

Upenn kick - A kick which is implemented by the University of Pennsylvania. This kick is also known as side swipe. It can swipe the ball in both left and right directions.

Grab dribble - It grabs and dribbles the ball for at most 3 seconds. Before the 3 seconds expires, it either kicks or releases the ball.

Avoid own goal - It tries to get behind the ball to avoid kicking the ball into the own goal. This is used only in the defensive half of the field.

Desired attack direction, in short DAD, is the global heading of the direction which the attacker should attack the ball. The previous rUNSWift teams have used the term, Desired Kick Direction, in short DKD instead of DAD. DAD is a more appropriate name, because not all action types have kicks. All the action types use this DAD, except for grab dribbling by the visual goal.

Action activate distance implies that the ball distance to be reached before a desired action should be invoked.

5.2.2 Field Division

The action selection is mostly based on the ball position which can broadly be classified into three types as shown in Figure 5.1 by two red lines. The two red lines breaks the field horizontally into three areas known as offensive third, middle third and defensive third. If the ball is in the defensive third, then the attacker should attack the ball defensively. If the ball is in the middle third, then the attacker should try to use dribble to push up the ball aggressively instead of grabbing it, because middle third is the area where players contest for the ball to get possession of the ball. If the ball is in the offensive third, then the attacker should grab the ball and visually aim at the goal to score goals.

The field can be further divided into left and right edges as shown in the same figure by two blue non-straight lines. When near the field edge, the attacker should try to keep the ball inside the field, because if it kicks the ball out, then the ball gets teleported to one of the throw-in points and this will immediately shut down the flow of the attack completely. In addition, the field edge area expands in the offensive area. This indicates that the attacker should be careful not to kick the ball out of the field, because if the attacker kicks it out, then the ball immediately gets placed on the centre line of the field.

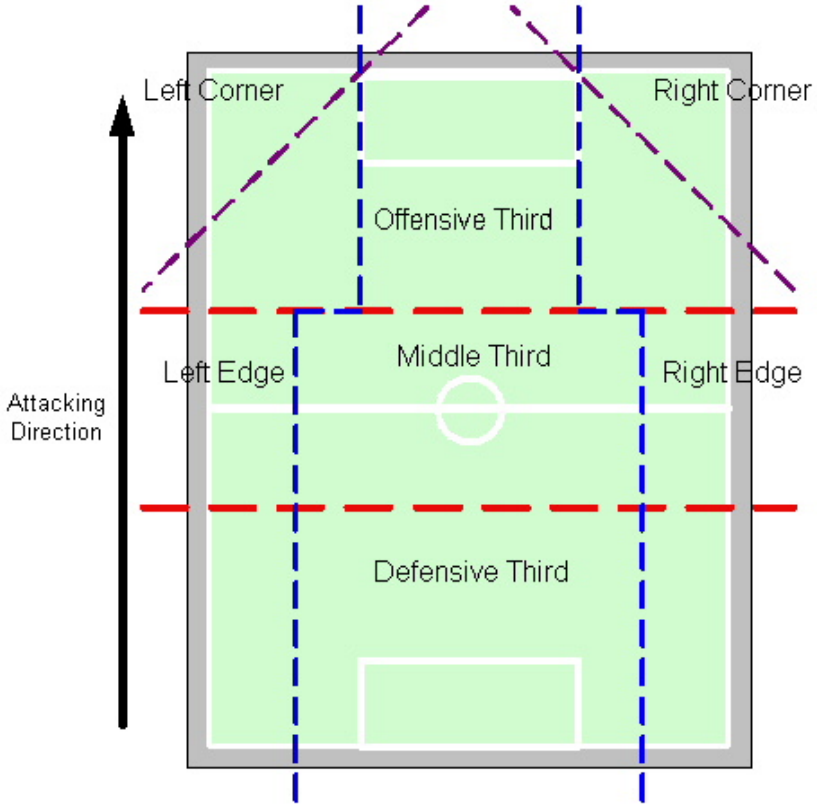

Figure 5.1: Field division shown graphically

Figure 5.1 also shows left and right corners in the offensive third. In both corners, the attacker is expected to cross the ball into the goal box. This assumes that there is a player waiting in the goal box to receive the ball. In Supporter/Defender formation, the supporter backs up wider towards the centre of the field when the ball is in one of the corners. Thus it implicitly waits somewhere around the goal box. For more information regarding the supporter's positioning, please refer to section 6.3.

5.2.3 Desired Attack Direction

The desired attack direction used to be called the desired kick direction. This year, we changed the name to desired attack direction, because not all actions which the attacker performs are kicks. The desired attack direction is referred to as DAD for short.

Since there were walls around the field in the previous years' competitions, it made sense to push the ball using the walls. However, as this year's field has no walls, the desired attack direction must be towards inside of the field. There are couple of different ways in calculating the desired attack direction. In this section, we will describe the calculation of the desired attack direction in a broad picture. The exact calculation of the desired attack direction in each region of the field is shown as an algorithmic style in section 5.3.

1. In either defensive or middle half, we try to attack towards up-field, but more towards inside the field, as the ball could go out of the field.

2. In an offensive half, we try to attack towards the target goal.

3. In the two corners of the offensive half, we try to attack towards the goal box, as if we are crossing the ball into the goal box for a teammate waiting inside.

4. Near the left or right field edge, we try to kick the ball inside the field to keep the ball in.

5.2.4 Action Selection

In this section, we will describe how the desired action is selected in general concepts. In section 5.3, we will discuss how the action is selected in detail by analysing the underlying selection algorithm.

The action selection is usually determined from the desired attack direction (DAD) which we have discussed in the previous section and also the robot's current heading to the ball and heading to the DAD. If a robot is already facing towards the DAD and the ball is right in front of you, either dribble or grab dribble will be selected. If the robot is facing a little off from the DAD, then it will do either Upenn left or right kick depending on which side it is on. If it is facing completely off from the DAD, then it will usually do grab dribble to turn around toward the DAD. However, there is an exception to this selection. If the robot is facing towards the own goal and is in either defensive or middle half, then it will try to get behind the ball explicitly to avoid facing towards the goal.

In general, the actions are selected as explained above. However, an action is chosen slightly different if it is a contested ball situation and this will be described in the next section.

5.2.5 Contested Ball Situation

A situation where the attacker and an opponent contest for a ball is referred to as a contested ball situation. In this situation, the attacker should act differently to situations where the ball is not contested. To begin, we will describe how this situation can be detected, followed by how we should handle the situation.

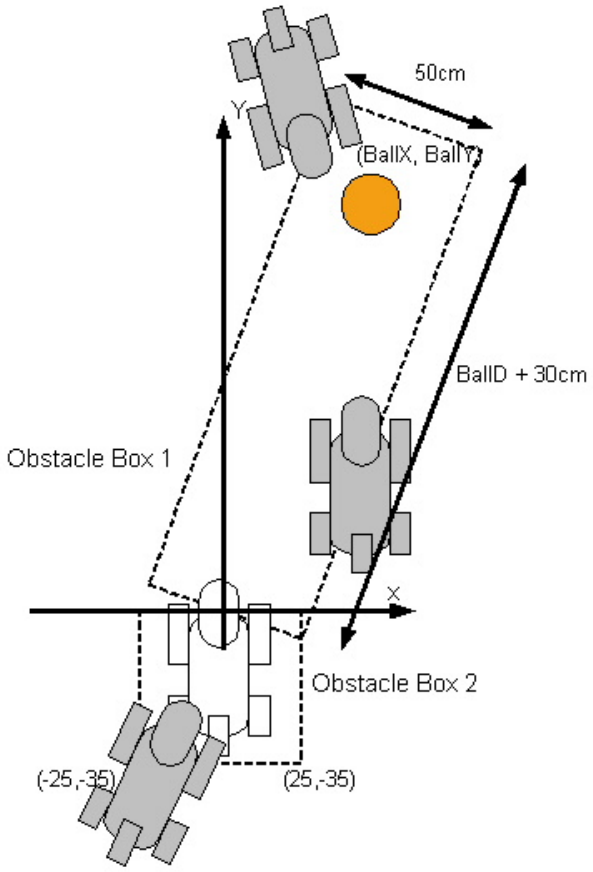

Figure 5.2: Contested ball detection using obstacles

The contested ball situation is detected if one of the three criteria is satisfied:

1. If there are enough GPS obstacle features between the ball and the robot, and also behind the robot. GPS obstacles are tracked by caching obstacle features detected by the vision module and decaying them

as time goes by. This is shown in Figure 5.2. For more information regarding obstacles, please refer to [J. Shammay].

2. If both left and right rear rotator joints' max step PWM values exceed a threshold. Currently, the threshold is set to 800.

3. If the front feet sensors of a robot detects its paws hitting an obstacle.

Our basic approach to handle the contested ball situation is to push the ball up the field as quickly as possible by using the Upenn kick and the dribble. The ball grabbing behaviour is not used unless it is necessary and situations where the grabbing is required are as follows:

- the ball is near the left field edge and the robot is not facing in the DAD.
- the ball is near the right field edge and the robot is not facing in the DAD.
- the robot is facing downwards (towards its own goal) and it is in either the defensive or middle third of the field.

5.3 Design and Implementation

5.3.1 Top Level

Algorithm 5.1 is the top level function of AAS. This top level function is rather simple compared to other underlying functions.

- If the ball lies in the defensive third, SelectInDefensiveThird() is invoked.
- If the ball lies in the middle third, SelectInMiddleThird() is invoked.
- If the ball lies in the offensive third, SelectInOffensiveThird() is invoked.

Algorithm 5.1: SelectAction

Data:
Result:

```
1 if the ball's y position < FIELD_LENGTH * 0.4 then
2 |   return SelectInDefensiveThird()
3 else if the ball's y position < FIELD_LENGTH * 0.6 then
4 |   return SelectInMiddleThird()
5 else
6 |   return SelectInOffensiveThird()
7 end
```

5.3.2 SelectInDefensiveThird

In the defensive third, it is important to consider the own goal, because we must avoid own goal as much as possible. Thus, after checking if a robot has grabbed a ball or not, we have several criteria and if one of these criteria satisfies, then we say that the robot is facing towards its own goal:

1. If the robot's heading to the ball roughly faces towards the own goal.
2. If the robot's heading to the ball is between the headings of left and right edges of the own goal box.
3. If the robot can visually detect its own goal.

If the robot is facing towards its own, it selects an action called AVOID_OWN_GOAL. This action forces the robot to get behind the ball as much as possible, so that the robot will not be lined up with the own goal.

If the robot is not facing towards its own goal, the function now checks if the ball is in the contested situation or not. How the contested ball situation is detected is documented in section 5.2.5. If the ball is, indeed, contested, then the robot tries to use either Upenn kick or dribble to move the ball up the field quickly. However, grab is still chosen as an action if the robot is facing down field direction or is near the field edge and not facing towards the target goal, because we do not want to Upenn kick or dribble in some direction which is not up field direction.

If the ball is not in the contested situation, we then check if the ball lies near either left or right field edge. If it is, then it will invoke SelectInEdge() function to handle the field edge case.

If none of the checks apply, then the function will select Upenn kick as long as the robot's heading is within the appropriate kick angle range. If the robot's heading is not within the range, then it would try to grab the ball, unless it is already facing directly towards the DAD, then it will do dribble instead.

The following is a list of angles used in Algorithms 5.2 and 5.3 to select an action.

selfH2BallH = normaliseAngle_0_360(selfH + ballH)

ballH2OGoalH = getHeadingBetween(ballX, ballY, OWN_GOAL_X, OWN_GOAL_Y)

ballH2MinOGoalBoxH =
getHeadingBetween(ballX, ballY, MIN_GOALBOX_EDGE_X, OWN_GOAL_Y)

ballH2MaxOGoalBoxH =
getHeadingBetween(ballX, ballY, MAX_GOALBOX_EDGE_X, OWN_GOAL_Y)

selfH2OGoalH = normaliseAngle_180(selfH2BallH - ballH2OGoalH)

ballH2CenterH =
getHeadingBetween(ballX, ballY, FIELD_WIDTH / 2.0, ballY + 200)

selfH2CenterH = normaliseAngle_180(selfH2BallH - ballH2CenterH)

Algorithm 5.2: SelectInDefensiveThird

```
1  if a robot has grabbed the ball already then
2  |  return (GRAB_DRIBBLE_DAD, ballH2CentreH, AUTO)
3  else if abs(selfH2OGoalH) < 60
4  OR ballH2MinOGoalBoxH < selfH2BallH < ballH2MaxOGoalBoxH
5  OR the own goal is visible then
6  |  return (AVOID_OWN_GOAL, ballH2CentreH, AUTO)
7  else if the ball is contested then
8  |  if 45 <= abs(selfH2TGoalH) < 60 then
9  |  |  if selfH2TGoalH < 0 then
10 |  |  |  return (UPENN_RIGHT, ballH2TGoalH, AUTO)
11 |  |  else
12 |  |  |  return (UPENN_LEFT, ballH2TGoalH, AUTO)
13 |  |  end
14 |  else if (the ball is near the left edge AND abs(selfH2TGoalH) >
   |  10)
15 |  OR (the ball is near the right edge AND abs(selfH2TGoalH) > 10)
16 |  OR selfH >= 180 then
17 |  |  return (GRAB_DRIBBLE, ballH2TGoalH, AUTO)
18 |  else
19 |  |  return (DRIBBLE, ballH2TGoalH, AUTO)
20 |  end
21
```

Algorithm 5.3: SelectInDefensiveThird-Continue

```
else if the ball is near the left or right edge then
    return SelectInEdge()
else
    if 45 <= abs(selfH2CentreH) < 60 then
        if selfH2CenterH < 0 then
            return (UPENN_RIGHT, ballH2CenterH, AUTO)
        else
            return (UPENN_LEFT, ballH2CenterH, AUTO)
        end
    else if abs(selfH2CentreH) < 90 then
        if abs(selfH2CenterH) < 20 then
            return (DRIBBLE, ballH2CenterH, AUTO)
        else
            return (GRAB_DRIBBLE, ballH2CenterH, AUTO)
        end
    else
        return (GRAB_DRIBBLE, ballH2CenterH, AUTO)
    end
end
```

5.3.3 SelectInMiddleThird

The structure in SelectInMiddleThird() is exactly the same as the structure in SelectInDefensiveThird(). The only difference is the criteria that determines whether a robot is facing towards its own goal or not is weaker than the defensive third ones. Also, some of assignments to the variables slightly differ from the ones in SelectInDefensiveThird().

The following is a list of angles used in Algorithms 5.4 and 5.5 to select an action.

selfH2BallH = normaliseAngle_0_360(selfH + ballH)

ballH2OGoalH = getHeadingBetween(ballX, ballY, OWN_GOAL_X, OWN_GOAL_Y)

selfH2OGoalH = normaliseAngle_180(selfH2BallH - ballH2OGoalH)

ballH2TGoalH = getHeadingBetween(ballX, ballY, TARGET_GOAL_X,

TARGET_GOAL_Y - 50)

selfH2TGoalH = normaliseAngle_180(selfH2BallH - ballH2TGoalH)

ballH2CenterH =
getHeadingBetween(ballX, ballY, FIELD_WIDTH / 2.0, ballY + 200)

selfH2CenterH = normaliseAngle_180(selfH2BallH - ballH2CenterH)

Algorithm 5.4: SelectInMiddleThird

```
1  if a robot has grabbed the ball already then
2  |  if the ball is near the left edge or the right edge then
3  |  |  return (GRAB_DRIBBLE_DAD, ballH2CentreH, AUTO)
4  |  else
5  |  |  return (GRAB_DRIBBLE_DAD, ballH2TGoalH, AUTO)
6  |  end
7  else if abs(selfH2OGoalH) < 40
8  OR the own goal is visible then
9  |  return (AVOID_OWN_GOAL, ballH2TGoalH, AUTO)
10 else if the ball is contested then
11 |  if 45 <= abs(selfH2TGoalH) < 60 then
12 |  |  if selfH2TGoalH < 0 then
13 |  |  |  return (UPENN_RIGHT, ballH2TGoalH, AUTO)
14 |  |  else
15 |  |  |  return (UPENN_LEFT, ballH2TGoalH, AUTO)
16 |  |  end
17 |  else if (the ball is near the left edge AND abs(selfH2TGoalH) >
   |  10)
18 |  OR (the ball is near the right edge AND abs(selfH2TGoalH) > 10)
19 |  OR selfH >= 180 then
20 |  |  return (GRAB_DRIBBLE, ballH2TGoalH, AUTO)
21 |  else
22 |  |  return (DRIBBLE, ballH2TGoalH, AUTO)
23 |  end
24
```

Algorithm 5.5: SelectInMiddleThird-Continue

```
else if the ball is near the left or right edge then
    return SelectInEdge()
else
    if 45 <= abs(selfH2GoalH) < 60 then
        if selfH2GoalH < 0 then
            return (UPENN_RIGHT, ballH2CenterH, AUTO)
        else
            return (UPENN_LEFT, ballH2CenterH, AUTO)
        end
    else if abs(selfH2GoalH) < 90 then
        if abs(selfH2GoalH) < 10 then
            return (DRIBBLE, ballH2TGoalH, AUTO)
        else
            return (GRAB_DRIBBLE_DAD, ballH2TGoalH, AUTO)
        end
    else
        return (GRAB_DRIBBLE_DAD, ballH2TGoalH, AUTO)
    end
end
```

5.3.4 SelectInOffensiveThird

The following is a list of angles used in Algorithms 5.6 to 5.9 to select an action.

tGoalY = TARGET_GOAL_Y + 20

selfH2BallH = normaliseAngle_0_360(selfH + ballH)

ballH2TGoalH = getHeadingBetween(ballX, ballY, TARGET_GOAL_X, tGoalY)

selfH2TGoalH = normaliseAngle_180(selfH2BallH - ballH2TGoalH)

ballH2CrossingH = getHeadingBetween(ballX, ballY, TARGET_GOAL_X, tGoalY - 40)

selfH2CrossingH = normaliseAngle_180(selfH2BallH - ballH2CrossingH)

ballH2CenterH =
getHeadingBetween(ballX, ballY, FIELD_WIDTH / 2.0, ballY + 200)

selfH2CenterH = normaliseAngle_180(selfH2BallH - ballH2CenterH)

The first part of SelectInOffensiveThird() checks if a robot has grabbed a ball already. If it is, then there are three different action selections:

1. if the ball is in the top left or top right corner, then do grab dribble towards the goal box,
2. if the ball is near the left or right field, then do grab dribble towards the selfH2CenterH direction.
3. Otherwise, do grab dribble using visual detection of the target goal.

Algorithm 5.6: SelectInOffensiveThird-Grabbed

```
1 if a robot has grabbed the ball then
2 |   if the ball is in the top left or top right corner then
3 |   |   return (GRAB_DRIBBLE_DAD, ballH2CrossingH, AUTO)
4 |   else if the ball is near the left or right edge then
5 |   |   return (GRAB_DRIBBLE_DAD, ballH2CenterH, AUTO)
6 |   else
7 |   |   return (GRAB_DRIBBLE_TARGET_GOAL, ballH2TGoalH,
  |   |   AUTO)
8 |   end
9
```

Algorithm 5.7: SelectInOffensiveThird-Contested

```
else if the ball is contested then
    dad = ballH2TGoal
    if the ball is in the top left or top right corner then
        dad = ballH2CrossingH
    end
    selfH2dad = normaliseAngle_180(selfH2BallH - dad)
    if 45 <= abs(selfH2dad) < 60 then
        if selfH2dad < 0 then
            return (UPENN_RIGHT, ballH2TGoalH, AUTO)
        else
            return (UPENN_LEFT, ballH2TGoalH, AUTO)
        end
    else if the ball is in the top left or top right corner
    OR the ball is near the left or right edge
    OR selfH >= 180 then
        return (GRAB_DRIBBLE_DAD, ballH2TGoalH, AUTO)
    else
        return (DRIBBLE, ballH2TGoalH, AUTO)
    end

```

The second part of the function will check if the ball is in the contested situation. If the ball is known to be contested, then a decision which is similar to the ones in SelectInDefensiveThird() (section 5.3.2) and SelectInMiddleThird() (section 5.3.3 will be made here. The only difference is the DAD changes if the robot is in the top left or top right corner.

Algorithm 5.8: SelectInOffensiveThird-CornerAndEdge

```
1  else if the ball is in the top left or top right corner then
2  |   if abs(selfH2CrossingH) < 10 then
3  |   |   return (DRIBBLE, ballH2CrossingH, AUTO)
4  |   else if 45 <= abs(selfH2CrossngH) < 60 then
5  |   |   if selfH2CrossingH < 0 then
6  |   |   |   return (UPENN_RIGHT, ballH2CrossingH, AUTO)
7  |   |   else
8  |   |   |   return (UPENN_LEFT, ballH2CrossingH, AUTO)
9  |   |   end
10 |   else
11 |   |   return (GRAB_DRIBBLE_DAD, ballH2CrossingH, AUTO)
12 |   end
13 else if the ball is near the left or right edge then
14 |   return (GRAB_DRIBBLE_DAD, ballH2TGoalH, AUTO)
15
```

The third part of the function will check if the ball is in the top right or top left corner and decides an appropriate action. It also checks whether the ball is near the left or right edge or not. If it is then, it selects grab dribbling with the heading to the target goal as the DAD.

Algorithm 5.9: SelectInOffensiveThird-Other

```
else if the robot has not grabbed the ball
AND ballY > FIELD_LENGTH - 30
AND ballX > FIELD_WIDTH / 2 - GOAL_WIDTH / 2 + 15
AND ballX < FIELD_WIDTH / 2 + GOAL_WIDTH / 2 - 15
AND abs(selfH2TGoalH) < 45 then
    return (DRIBBLE, ballH2TGoalH, AUTO)
else if ballY > (FIELD_LENGTH - GOALBOX_DEPTH) AND ballX
> (FIELD_WIDTH / 2 - GOAL_WIDTH / 2 - 50) AND ballX <
(FIELD_WIDTH / 2 + GOAL_WIDTH / 2 + 50) then
    return (GRAB_DRIBBLE_DAD, ballH2TGoalH, AUTO)
else if ballY < FIELD_LENGTH * 0.65
AND abs(selfH2TGoalH) < 15 then
    return (DRIBBLE, ballH2TGoalH, AUTO)
else
    return (GRAB_DRIBBLE_TARGET_GOAL, ballH2TGoalH,
    AUTO)
end
```

The last part of the function check if the ball is close to the goal or not. Depending on how close to the target goal, it will select either dribble or grab dribble. If the robot is a little far from the target goal and is roughly lined up to the target goal direction, then it will select dribble. Otherwise, it will choose to do grab dribble using visual target goal detection.

5.3.5 SelectInEdge

The following is a list of angles used in Algorithm 5.10 to select an action.

selfH2BallH = normaliseAngle_0_360(selfH + ballH)

ballH2TGoalH = getHeadingBetween(ballX, ballY, TARGET_GOAL_X, TARGET_GOAL_Y)

Algorithm 5.10: SelectInEdge

```
1 if 45 <= abs(selfH2GoalH) < 60 then
2 |   if selfH2GoalH < 0 then
3 |   |   return (UPENN_RIGHT, ballH2TGoalH, AUTO)
4 |   else
5 |   |   return (UPENN_LEFT, ballH2TGoalH, AUTO)
6 |   end
7 else
8 |   return (GRAB_DRIBBLE_DAD, ballH2TGoalH, AUTO)
9 end
```

5.4 Further Work

5.4.1 Considering Teammates' Information

At the moment, our AAS does not consider teammates' information. The teammates' information especially their global positions on the field might be an interesting dimension to include in the AAS. By considering where each teammate is located, we can set the DAD to the heading towards a particular teammate, and thereby, we implicitly introduce the ball passing to the teammate. If the field gets bigger and the robots spread out more in the near future, this ball passing concept might well be worth implementing.

Part III

High Level Behaviours

Chapter 6

Role Behaviours

"Where's Benny?"

- Everyone except Wei Ming (aka. Benny) (We said this when he did not come into the lab by 4pm. After all, he was the only night shifter in the team! =))

6.1 Introduction

This chapter will discuss the role behaviours. The role behaviours include attacker, supporter, striker and defender. Each role can be briefly described as:

Attacker - It mainly chases after the ball.

Supporter - It closely backs up the attacker and the ball.

Striker - It can be thought of as a long distance supporter.

Defender - It hangs back in the defensive half.

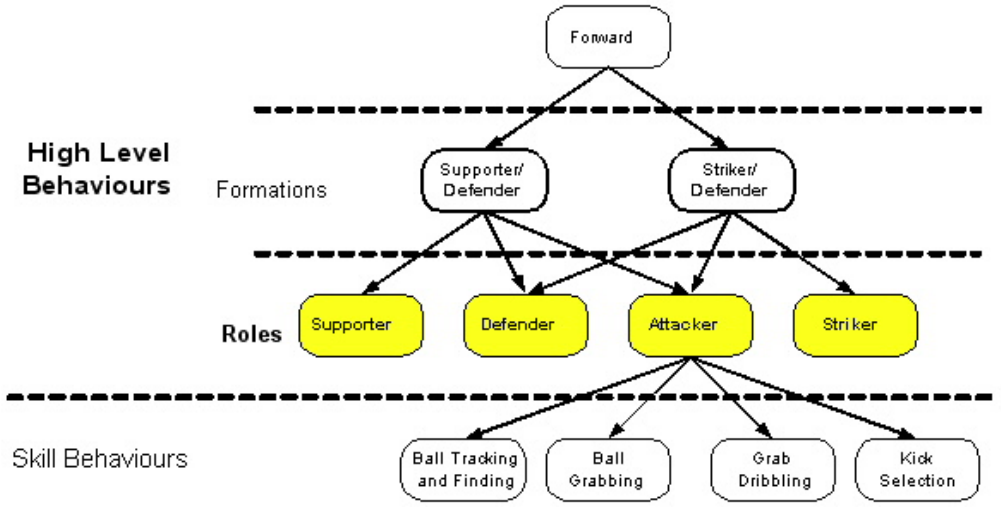

Figure 6.1: Architecture : Roles focused

This chapter describes each role in detail.

6.2 Attacker

When a robot gets assigned to the attacker role, its main responsibility is to chase after the ball and attack it. In addition to this responsibility, it includes sub-behaviours like own goal box avoidance, active localisation and dodgy dog. In this section, we will describe each of these sub-behaviours which make up the attacker role behaviour.

6.2.1 Back Off Mechanism

This behaviour is usually triggered when a robot thinks it is stuck in an obstacle and it will be explained in detail in section 6.3.2. In this section, we will outline how this back off mechanism gets triggered.

We are already aware that the attacker's responsibility is to get to the ball first and attack it, and because of this, the attacker should not back off when the ball is in close distance. Thus, the trigger criteria for the attacker had to be made much stronger than the criteria used for a supporter.

The back off trigger criteria for the attacker are as follows:

- the attacker detects it is stuck.
- visual ball distance is greater than 200cm,
- GPS ball distance is greater than 200cm,
- the attacker must not have grabbed the ball,
- the attacker must not have tried to grab the ball in the last 5 vision frames,

In section 6.3.2, we will describe how a robot detects it is stuck in something. For now, we will assume that it can detect whether it is stuck or not.

6.2.2 Own Goal Box Avoidance

This behaviour is included in the attacker behaviour to avoid it entering the goal box and being taken out from the field for an illegal defender penalty. Other behaviour roles do not need this behaviour, because they position themselves to implicitly avoid the goal box. Since the attacker always chases after the ball, we explicitly need this avoidance behaviour in the attacker behaviour.

Basically, this behaviour is triggered when the attacker thinks that the ball is in the goal box and also it must not have grabbed the ball. There are three different cases in which we handle this behaviour and they are:

1. If the attacker is far away from its own goal, then it just walks towards the ball. This implies that we do not need to trigger any special behaviours yet. This is shown in case 1 of Figure 6.2.

2. If the attacker is next to the goal box, then it walks to the corner of it. This is shown in case 2 of the same figure.

3. If the attacker is in front of the goal box, then it moves to the side of the ball. This, in fact, allows the attacker to track the ball as well as do active localisation. In case 3 of the same figure, both robots are positioned in such a way that it is possible for them to look at the landmarks.

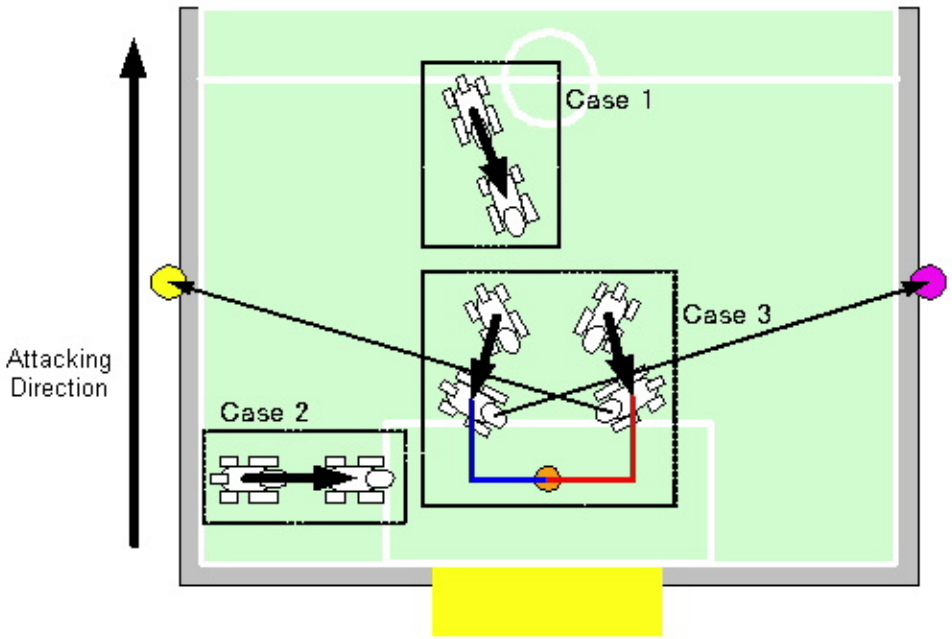

Figure 6.2: Own goal box avoidance

Active Localisation while avoiding the goal box is essential, as a robot must know its position accurately not to go inside the goal box. The positioning of the goal box avoidance also gives a clear view for the goalie to kick the ball either straight out or to the side of the goal box.

6.2.3 Active Localisation

Because the attacker needs to track the ball all the time, it can often get mislocalised, and for the attacker to act sensibly while in possession of the ball, it is desired to know its accurate position on the field. This mislocalisation can be resolved by actively looking at the landmarks provided around

the field time to time. This behaviour is known as active localisation.

This year, we have used the following criteria to trigger the active localisation:

- The attacker has visually detected the ball for the past 3 vision frames.
- The attacker has not actively localised recently.
- The ball is still far away and straight ahead.
- The ball is not moving fast.
- There is not too many obstacle features between the attacker and the ball. This implies that there should not be anything in the way of the attacker.
- There is a beacon which the attacker can look at from its position and heading.

6.2.4 Action Selection

Action selection (aka. Attacker Action Selection (AAS)) was discussed in detail as an individual skill behaviour in chapter 5. In this section, we will try to expand the idea by describing how we integrated this skill into the attacker role behaviour. Since the attacker needs to know which action is suitable, it must make a query to the AAS and it acts accordingly to the action selected by the AAS. Because the attacker should be responsive in what it should do, it queries the AAS almost every vision frame to reselect the desired action with some exceptional conditions. In these exceptional conditions, the action will not be reselected until the attacker finishes the selected action or one of the exceptional conditions holds true. These exceptional conditions are as follows:

- If the ball is close to the attacker.
- If the selected action requires a grab and the ball is close to the attacker.
- If the attacker has already grabbed the ball.
- If the attacker has started kicking.

6.2.5 Dodgy Dog

In 2003, a behaviour called stealth dog was implemented for a robot to avoid opponents while chasing after a ball[rUNSWift 2003]. This was a very effective behaviour in 2003, because without it, the rUNSWift's speed and swift action could have been severely hampered through the obstruction of opponents which are in front of the player. This year, we have developed another version of stealth dog called dodgy dog. Although these two behaviours ultimately have the same goal which is to avoid opponents, they fundamentally work differently. The former uses the robot detection and adds a constant turn amount to its walk. Whereas, the latter uses the obstacle detection to find the best gap between the detected obstacles and depending on the best gap heading, it adds a variable turn amount to the walk.

Without dodgy dog, it is difficult to use our full speed walk, because it can easily get caught up with other robots' legs, and this in turn, slows down the game play. Figure 6.3 shows the motivation behind the dodgy dog. If the robot was to walk straight to the ball, it would catch up with the opponent and get its leg locked. However, if the robot has the dodgy dog behaviour, it is possible to dodge around the opponent by walking in a curved path as shown.

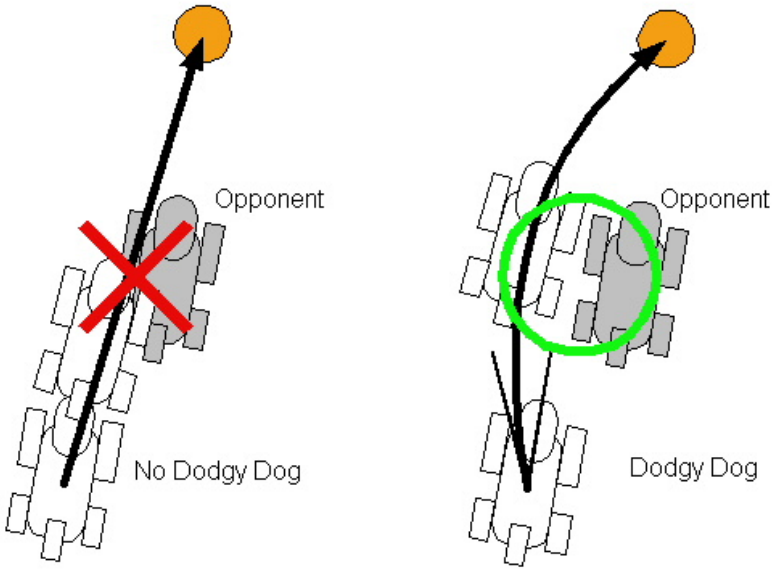

Figure 6.3: Motivation of dodgy dog

Details of how the dodgy dog behaviour is designed and implemented are described in [J. Shammay]. In this section, we will discuss how we integrated this dodgy dog behaviour into the attacker role behaviour. The trigger of the dodgy dog within the attacker role is the main focus of this section.

The dodgy dog constitutes to one of our rUNSWift skill sets and it can be executed on its own just like any other skill. It has its own trigger criteria and it checks for any obstacles in between a robot and a destination point. We usually specify the ball position for the destination point when the attacker is chasing after the ball. Whereas, the bird of prey behaviour which will be discussed in section 6.3.3 invokes dodgy dog with the point where a robot is going as the destination point. Thus, because an obstacle checking function is included in the dodgy dog itself, we do not have to put the criteria in the attacker behaviour. In fact, the criteria we put in the attacker are the ones specifically imply to its context and situation, and these criteria are defined as follows:

1. The attacker has not grabbed the ball.
2. The attacker has not started its grab motion.
3. The ball distance is outside the grab activation distance.
4. The specified forward speed must be greater than the specified side speed in the walk components.
5. The specified forward speed must be greater than half of the maximum forward speed.

6.3 Supporter

The supporter behaviour basically involves getting the robot to move to its supporting position and invokes the back off behaviour to avoid obstructing the teammate who is the attacker. The supporter role is somewhat less complex compared to the attacker, because it only needs to be at the designated position. In this supporter section, we will mainly describe the supporter positioning and the back off mechanisms included in the supporter.

6.3.1 Positioning

The positioning of the supporter is roughly outlined in Figure 6.4.

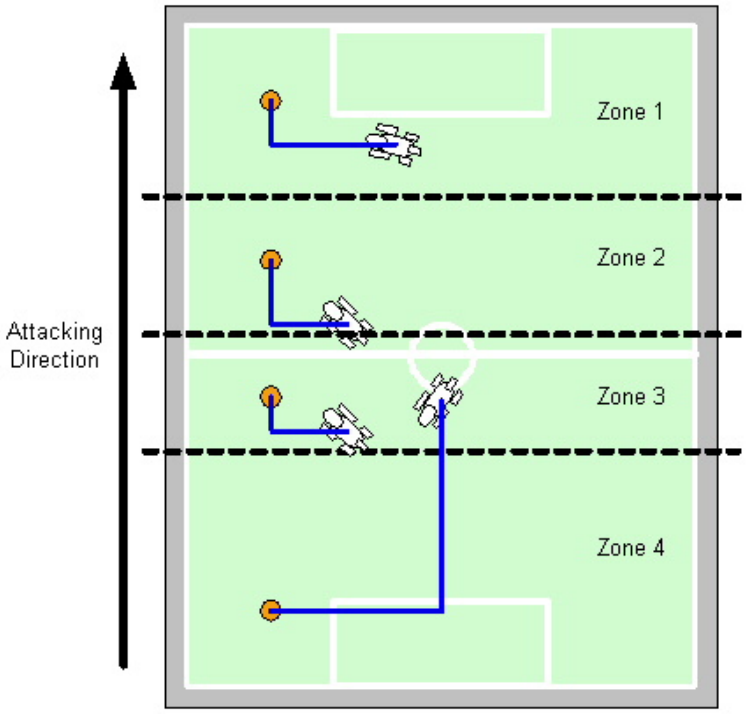

Figure 6.4: Supporter positioning

As we can see, the supporter positioning is broadly divided into four different zones. A zone in which the supporter belongs is determined solely by the ball's position in every vision frame. The zones are divided as follows:

Zone 1 - ball's y-position is greater than FIELD_LENGTH * 0.75.

Zone 2 - ball's y-position is between FIELD_LENGTH * 0.5 + 15 and FIELD_LENGTH * 0.75.

Zone 3 - ball's y-position is between FIELD_LENGTH * 0.22 + 80 and FIELD_LENGTH * 0.5 + 15.

Zone 4 - ball's y-position is less than FIELD_LENGTH * 0.22 + 80.

In each zone, the supporter position is calculated differently, and this calculation is done in two steps and is shown in Algorithms 6.1 and 6.2. Algorithm 6.1 returns a tuple of three values. These values constitute to x and y global coordinates of the base supporter position and the offset value which is to be added or subtracted from the x coordinate of the base supporter position to let the supporter stay on either left or right side of the ball depending on

the different situations.

Algorithm 6.1: Calculating Supporter Positioning

```
   Data:
   Result:
 1 offsetY = 80
 2 offsetX = 40
 3 ballX = ball's x-position
 4 ballY = ball's y-position
 5 if a supporter is in Zone 1 then
 6 |   supX = min(max(ballX, offsetX * 1.5),FIELD_WIDTH - offsetX *
   |   1.5)
 7 |   supY = ballY - (offsetY * 0.8)
 8 |   return (supX, supY, offsetX * 1.5)
 9 else if a supporter is in Zone 2 then
10 |   supX = min(max(ballX, offsetX),FIELD_WIDTH - offsetX)
11 |   supY = ballY - offsetY
12 |   return (supX, supY, offsetX)
13 else if a supporter is in Zone 3 then
14 |   supX = min(max(ballX, offsetX), FIELD_WIDTH - offsetX)
15 |   supY = max(ballY - offsetY * 0.5, 100)
16 |   return (supX, supY, 2*offsetX)
17 else
18 |   supX = FIELD_WIDTH * 0.5
19 |   supY = FIELD_LENGTH * 0.45
20 |   return (supX, supY, 0)
21 end
```

Algorithm 6.2 uses the three values calculated in the previous algorithm to decide on which side of the ball the supporter should back up and returns x and y coordinates of the adjusted supporter position.

Firstly, it checks if the the base supporter x coordinate is near the field edge or not. If it is, then the side which is towards the inside of the field is chosen. For example, if the base x coordinate is near the left field edge, then it should support on the right hand side of the ball. Thus, we add the offset x value to the base supporter x coordinate to get the adjusted x coordinate.

Then, it checks if the zone in which the supporter belongs is 1. If it is,

then it also chooses the side which is towards inside of the field. Since offset x in zone 1 is wider than usual, it positions itself near the goal box when the ball is in the corner. This positioning allows the supporter to implicitly wait for the ball to be crossed from the corner by the attacker.

Otherwise, we decide the appropriate side from the teammate who is currently assigned to the attacker role. If it is grabbing, then choose the side which is towards inside of the field. If it is not, then choose the opposite side on which the attacker is relative to the ball.

Algorithm 6.2: Calculating Adjusted Supporter Positioning

```
   Data:
   Result:
 1 supX, supY, offsetX = GetSupporterPos()
 2 if a supporter cannot see the ball then
 3 |  supY = supY - 20 offsetX = offsetX * 1.2
 4 end
 5 if supX is near the field edge then
 6 |  if supX < FIELD_WIDTH * 0.5 then
 7 |  |  supX = supX - offsetX
 8 |  else
 9 |  |  supX = supX + offsetX
10 |  end
11 else if a supporter is in Zone 1 then
12 |  if supX < FIELD_WIDTH * 0.5 then
13 |  |  supX = supX - offsetX
14 |  else
15 |  |  supX = supX + offsetX
16 |  end
17 else
18 |  if there is the attacker mate who has grabbed a ball then
19 |  |  if supX < FIELD_WIDTH * 0.5 then
20 |  |  |  supX = supX - offsetX
21 |  |  else
22 |  |  |  supX = supX + offsetX
23 |  |  end
24 |  else if the attacker mate is on the left hand side of the ball then
25 |  |  supX = supX + offsetX
26 |  else
27 |  |  supX = supX - offsetX
28 |  end
29 end
30 return (supX, supY)
```

6.3.2 Back Off Mechanisms

In 2001, the UNSW United introduced their first back off mechanism in the architecture using the visual detection of a teammate [UNSW United 2001]. Since then, back off has been a useful behaviour for non-attacker players

to not interfere with the attacker, and has led to a better local interaction and cooperation between the attacker and a supporter. In 2002, not only the visual back off was used, but also back off called world model rotational back off which used the world model teammate information was developed [rUNSWift 2002].

This year, because we have rewritten the vision module from scratch and an obstacle detection was ineffective for close distance obstacles, we had to devise a new approach in getting the supporter to back off to improve the local interaction. Currently, the supporter role employs three different back off mechanisms and they are as follows:

1. Back off using the two front feet sensors.
2. Back off using the world model teammate information and ball position.
3. Back off using the visual green feature count.

In this section, we will explain each back off mechanism in detail. Other sections which discuss back off mechanisms should consult this section for deeper understanding of this behaviour.

Back Off Using Two Front Feet Sensors
This year, we have experimented with the front feet sensors and found out that these sensors were actually quite useful, because our walk does not interfere with the sensors at all. Thus, the sensors only get fired when they hit something in front of the robot which is most likely be an obstacle during the game.

When this back off is fired, we have three different ways of reacting. They are as follows:

1. Both left and right sensors fired - step back
2. Left sensor fired - step back right
3. Right sensor fired - step back left

All of these reaction behaviours will be executed for at least 8 vision frames which is enough for our robots to take one whole step.

As an additional note, the behaviour used in the ready state of the game invokes this back off behaviour to avoid the leg locked situations with other robots.

Back Off Using World Model Teammate Information and Ball
As we have mentioned earlier, in 2002, rUNSWift had implemented a back off behaviour which used the world model information [rUNSWift 2002]. The back off behaviour which is about to be explained in this section is similar in a sense that it uses the world model information as well.

This back off behaviour is called "get out of the way" back off. It considers source and destination points and determines if a robot is in the way of these two points as shown in Figure 6.5. We say that a robot is in the way if it is within a certain perpendicular distance away from the line. The threshold distance can also be adjustable in this behaviour to cater for different end points. The same figure shows that there are two instances of "get out of the way" running. One is between the ball and the target goal and the other is between the ball and the attacker. In addition to this, the supporter in the figure tries to back off down the field and this makes it easier for it to go back to its supporter position.

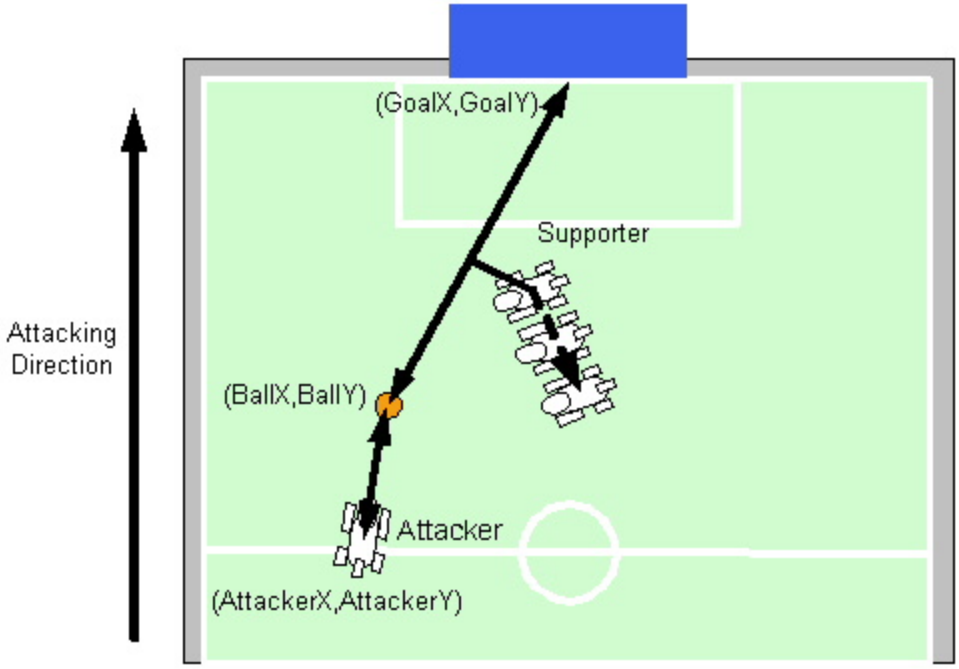

Figure 6.5: Getting out of the way

Back Off Using Visual Green Feature Count

The visual green feature is one type of the feature points detected in the camera image. The green represents the colour of the field and if any of the green features are found in the image, then this implies that we can see the field. If there are no green features in the image, then there are several possibilities.

- A robot is facing directly towards another robot.
- A robot is facing directly towards either a goal or a beacon.
- A robot is facing directly outside the field and is near the field edge.
- A robot is somewhere off the field. It may well be penalised and placed somewhere else.
- A robot is looking down.

The first three possibilities are things we should consider handling as back off behaviour. We do not want a non-attacker role charging into another robot as it will be taken out of the field for a player pushing penalty. A robot should not also run directly into a goal or a beacon, unless it can visually detect a ball in the target goal or next to the beacon. If a robot is facing outside the field, then it should do something intelligent to face inside the field. Since we are only looking at the green feature count, it is difficult or almost impossible to differentiate between the first three possible cases listed above. Thus, we have implemented a general back off behaviour which works in all of the three cases and is called the "non green" back off. The "no green" back off basically works as follows:

1. Step backwards for first 20 vision frames.
2. Turn around.

This behaviour exits when we see the field again. This idea is also shown in Figure 6.6. The picture on the left shows a robot backing off as it is too close to the robot in front and sees no green features. In this case, before it starts turning, it detects enough green features again and exits the back off process. The picture in the centre shows a robot standing in the blue goal and triggers back off. Since it is facing towards the goal, even if it moves backward, it may still not see any green features. However, because it starts turning after moving backwards, it will detect green features and the back off process will terminate. The similar scenario can be said for the picture on the right which shows a robot looking out to the side and does "no green

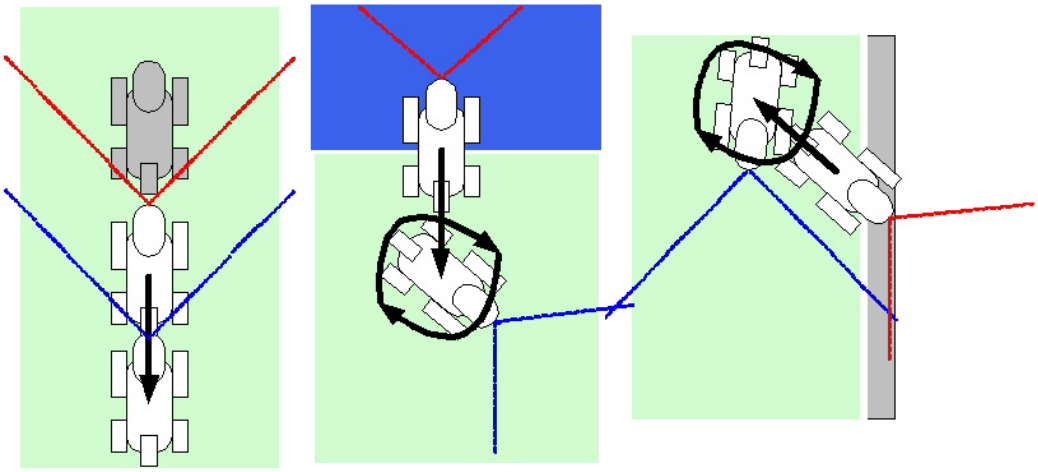

Figure 6.6: Effectiveness of green back off behaviour

back" off to face more into the field.

Because this back off behaviour is only triggered by checking the green feature count, it may well also be invoked in the last two possible cases as listed above. To distinguish from the above three cases, we have the following checks in place. If a robot is taken off the field for penalty, then it can know that it is penalised via game controller signal. Hence, it is possible to add a check to see if a robot is penalised or not before triggering the back off. Whether a robot is looking down or not can be determined from the current head parameter values. Thus, we can add another check which handles the situation for a robot which is looking down.

Lastly, in our implementation of the "no green" back off, instead of using the green feature count in the current image, we used the green feature count which is actually averaged over the last 30 frames and kept executing the back off until the average became above the threshold. This average acts like a hysteresis for not terminating the back off when incorrect green features are detected.

6.3.3 Bird of Prey with Dodgy Dog

The Bird of Prey (BOP) is a behaviour which was first introduced by the 2003 rUNSWift team to "counter the lack of a defensively assigned player in the game" [rUNSWift 2003]. 2003's BOP used the ball position as the des-

tination point but this year, we modified this to accept any positions on the field. This modification has been done by Alex North in the 2005 rUNSWift team. We have used this for both supporter and defender to get back to their respective positions.

Former BOP used stealth dog to dodge around obstacles. This year, since we have the dodgy dog, we replaced the stealth dog with the dodgy dog.

6.4 Striker

A striker role is included in the Striker/Defender formation and because of its positioning, the formation is more spread out compared to the Supporter/Defender. The striker's main responsibility is to stay in the offensive half to wait for the counter attack opportunity. To wait for such an opportunity, it mainly positions itself on the opposite side of where the ball is. The striker's behaviour is similar to the supporter in a sense that it runs to its position and waits for an opportunity.

6.4.1 Positioning

Figure 6.7 shows the broad outline of how striker positioning works. As we can see from the figure, the striker is not designed to come back in the defensive half for defensive support. It wanders around in the offensive half until it switches to some other role. As similar to the supporter positioning, the striker positioning is divided into three zones. These zones are divided as follows:

Zone 1 - ball's y-position is greater than FIELD_LENGTH * 0.6.

Zone 2 - ball's y-position is between FIELD_LENGTH * 0.6 and FIELD_LENGTH / 2.

Zone 3 - ball's y-position is less than FIELD_LENGTH / 2.

Algorithm 6.3 describes how to calculate the striker positions on the field. It returns a tuple of two values which constitute to x and y global coordinates of

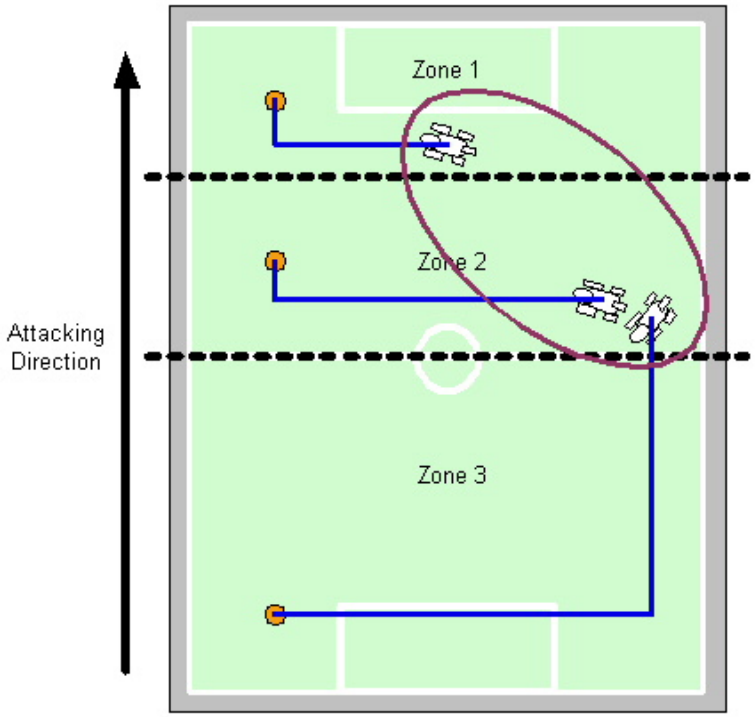

Figure 6.7: Striker positioning

the striker position.

Algorithm 6.3: Calculating striker positioning

Data:
Result:
offsetX = 60 offsetY = 80
ballX = ball's x-position
ballY = ball's y-position
strX, strY = 0, 0
if *a striker is in Zone 1* **then**
    strX = FIELD_WIDTH / 2
    strY = min(ballY - offsetY / 2, FIELD_LENGTH - offsetY)
else if *a striker is in Zone 2* **then**
    if *ballX > FIELD_WIDTH / 2* **then**
        strX = offsetX
    else
        strX = FIELD_WIDTH - offsetX
    end
    strY = min(ballY - offsetY / 2, FIELD_LENGTH / 2)
else
    if *ballX > FIELD_WIDTH / 2* **then**
        strX = offsetX
    else
        strX = FIELD_WIDTH - offsetX
    end
    strY = FIELD_LENGTH / 2 + offsetY
end
return (strX, strY)

6.5 Defender

A defender role is somewhat complement to a striker role. Its main responsibility is to hang back in the defensive half to handle the ball out situations, which lead to possible counter attacks by opponents. The defender not only hangs back in the defensive half, but also blocks the ball when it is coming towards the defender in fast velocity.

6.5.1 Positioning

Figure 6.7 shows the broad outline of how defender positioning works. As we can see from the figure, the defender never moves out of the defensive half. As similar to the supporter positioning, the defender positioning is divided into three zones. These zones are divided as follows:

Zone 1 - ball's y-position is greater than FIELD_LENGTH * 0.6.

Zone 2 - ball's y-position is between FIELD_LENGTH * 0.6 and FIELD_LENGTH * 0.22.

Zone 3 - ball's y-position is less than FIELD_LENGTH * 0.22.

Algorithm 6.4 describes how to calculate the defender positions on the field. It returns a tuple of two values which constitute to x and y global coordinates of the defender position.

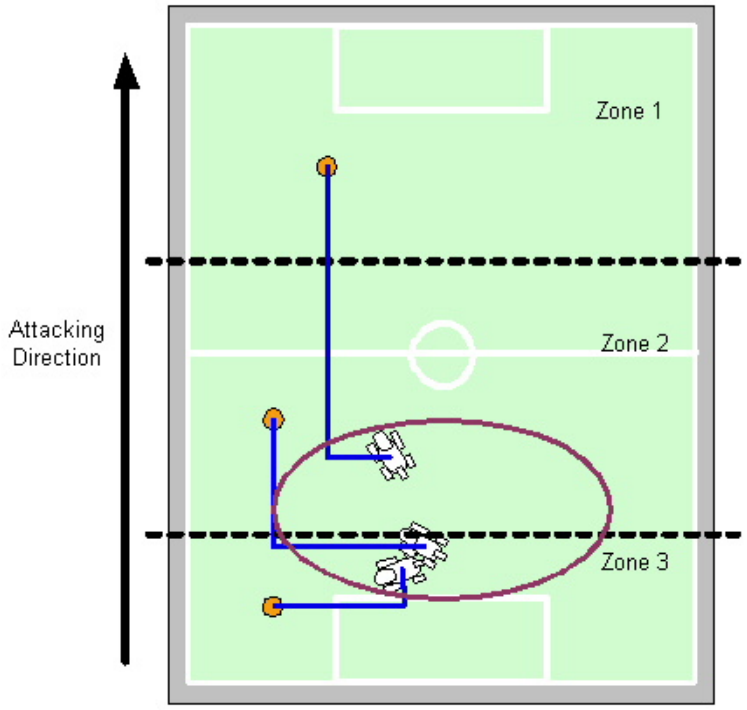

Figure 6.8: Defender positioning

Algorithm 6.4: Calculating defender positioning

Data:
Result:
offsetX = 70
ballX = ball's x-position
ballY = ball's y-position
defX, defY = 0, 0
if *a defender is in Zone 1* **then**
    defX = FIELD_WIDTH / 2 + (ballX - FIELD_WIDTH / 2) / 2
    defY = FIELD_LENGTH * 0.32
else if *a defender is in Zone 2* **then**
    defX = (ballX + FIELD_WIDTH / 2) / 2
    defY = max(GOALBOX_DEPTH + 27, ballY * 0.4)
else
    if *ballX > FIELD_WIDTH / 2* **then**
        defX = min(ballX - offsetX, FIELD_WIDTH / 2 + 80)
    else
        defX = max(ballX + offsetX, FIELD_WIDTH / 2 - 80)
    end
    defY = max(ballY, GOALBOX_DEPTH + 20)
end
return (defX, defY)

6.5.2 Ball Blocking

The defender has a blocking behaviour included in its behaviour. This blocking behaviour is exactly the same as the one included in the goalie behaviour, but the block trigger criteria for the defender is much tighter than the goalie's.. The blocking trigger criteria for the defender are as follows:

- the defender is in its defending position,
- it is not facing down the field,
- it is not visually detecting its own goal,
- it can see the ball,
- the block can be reached in time,
- the ball distance is between 30cm and 80cm, and
- the ball velocity is between 2cm / frame to 10cm / frame,

6.6 Experimental Result

6.6.1 Attacker Behaviour (In Lab)

The experiments described in this section were carried out to test the effectiveness of our attacker behaviour. We have prepared two experiments and both experiments involved the attacker running around the field for 3 minutes and score as many goals as possible in the yellow target goal. When the ball went out of the field, it was placed in one of the throw-in points shown in Figure 6.9. We chose the closer throw-in point whenever the ball went out. If the goal was scored, the ball was placed near the centre of the field, but a little closer to the target goal shown as "Start Point" in the same figure. Since there was only one attacker running around on the field in the experiments, we ran the experiment at a smaller scale to avoid the attacker spending too much time chasing after the ball and taking it up the field instead of actually scoring the goal.

The first experiment was done with a goalie behaviour in front of the target goal and the second experiment was done without a goalie. The purpose of these two experiments was to evaluate if all the underlying behaviours are

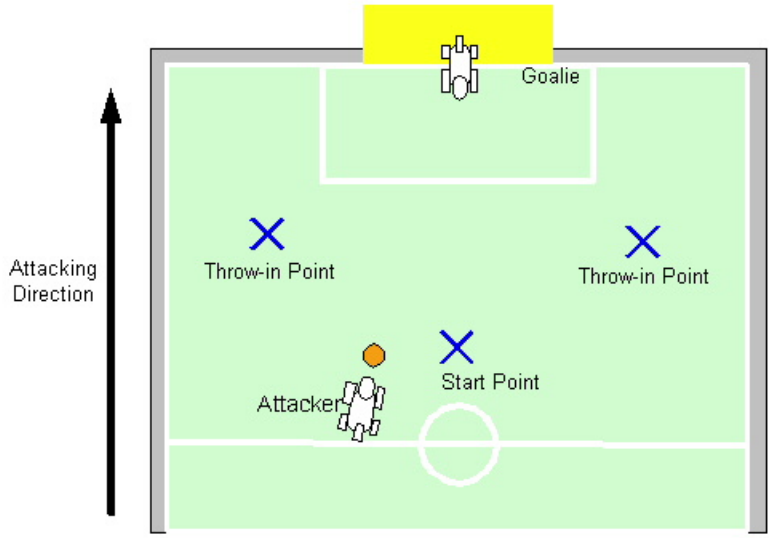

Figure 6.9: Attacker behaviour experiment

working together to make sensible moves in different situations, and counting number of goals scored was a reasonable benchmark in assessing the effectiveness of the attacker behaviour.

Table 6.1 shows the result for this experiment which tried 5 times. We can see that the goal count varied from 0 to 7 goals. This fluctuation of the goal counts may well be because of the presence of the goalie which tries to block the attacker's shots.

As said earlier, the second experiment is exactly the same as the first experiment, but carried out without the goalie behaviour running in front of the target goal. The purpose of this experiment was to check if the goal count fluctuation observed in the first experiment was due to the goalie's presence. Table 6.2 shows the result of the experiment which was carried out 3 times. It shows that the attacker can score more goals when there is no goalie in front of the target goal.

In both tables, we can see that there are extra columns assessing the ball grabbing behaviour. The evaluation of the ball grabbing behaviour is actually documented in section 3.4, as it is more appropriate to discuss this in the ball grabbing chapter rather than in this chapter. The purpose of this chapter was to discuss the ideas behind the overall role behaviours.

Sample	Goal Count	Grab			Grab Success Rate
		Success	Failed	Hesitated	
1	5	12	1	1	85.7%
2	0	12	3	4	63.2%
3	7	13	1	1	86.7%
4	3	17	5	2	70.8%
5	6	16	0	1	94.1%

Table 6.1: Attacker behaviour with a goalie for 3 minutes

Sample	Goal Count	Grab			Grab Success Rate
		Success	Failed	Hesitated	
1	9	13	1	2	81.3%
2	10	12	4	2	66.7%
3	6	12	3	1	75.0%

Table 6.2: Attacker behaviour without a golie for 3 minutes

Chapter 7

Formation and Strategy Behaviours

"You want a mint?"

- Joshua Shammay (I finally had his mint, after he asked me about a hundred times.)

7.1 Introduction

In this chapter, we will discuss the highest level of the behaviour hierarchy called formations behaviours. The formation behaviour controls the team organisation, so that the robots can play soccer intelligently. Currently, rUNSWift has two distinct formation behaviours. The first one is a rather aggressive strategy which includes attacker, supporter and defender roles. The other formation is a more spread out strategy which includes attacker, striker and defender roles. We can switch between formations at runtime both programmatic and using a wireless signal. Thus, we can test out different formations without remaking the sticks and rebooting the robots.

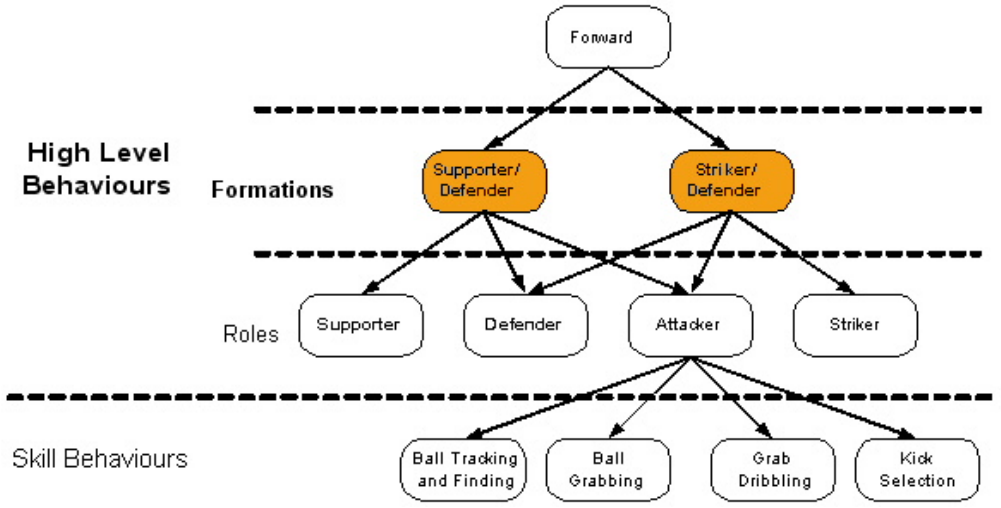

Figure 7.1: Architecture : Formations focused

This chapter begins by describing the concepts and ideas behind the two formation behaviours which we have implemented. Then, in section 7.4, we will describe rUNSWift's dynamic role switching, the method by which roles are assigned to robots. This is followed by a detailed explanation of criteria used for assigning different roles.

7.2 Supporter / Defender Formation

7.2.1 Overview

This formation behaviour is based on the 2003 rUNSWift team's formation [rUNSWift 2003]. The major change we have made to the formation is the replacement of a winger role to a defender role. To avoid any confusions about role names used in previous years, a clarification must be made. In previous years, what we now refer to a winger was called a striker. This year, we used the term, striker to imply a different role.

The concept of the defender role is simple. It is to always stay in the defensive half of the field to avoid counter attacks by opponents. This role is useful, if the attacker kicks the ball out in its offensive half, because even if the ball teleports to the middle of the field, we still have one field player in between the ball and the own goal. With the 2003 rUNSWift team's formation, all the three field players would be far up in the offensive half and this leaves no field player behind the ball. Although the bird of prey behaviour was intended to mitigate possible counter attack situations, due to this year's field expansion, it was not feasible for the robots to come back quickly to handle the ball out situations. Thus, we have decided to have one field player staying back all the time. This is shown graphically in Figure 7.2. In the ball out situation shown in the figure, the ball would be placed in the middle of the field as shown by the arrow. The formation on the right is still safe, because it has one player as a defender role staying back in its defensive half. Whereas, the formation on the left shows that all the field players are in front of the ball and the goalie is the only robot to defend from the counter attacks.

The attacker and the supporter roles continue to follow the same philosophy which the 2003 rUNSWift team had. The attacker always gets to the ball first and the supporter closely backs up the attacker to cover any ball handling mistakes that the attacker makes. However, because there were no walls around the field this year, we had to remove all the behaviours that relied on the existence of the walls. One of these behaviours were in both the attacker and the supporter which positioned themselves to move in a defensive circular path in the bottom corners of the field. In 2003, although this was indeed effective in pushing the ball up the field using the wall, it was not applicable to this year's field, because the ball could go out.

Since the supporter places itself close to the attacker, good local cooperation between these two roles is required to avoid any obstruction to each

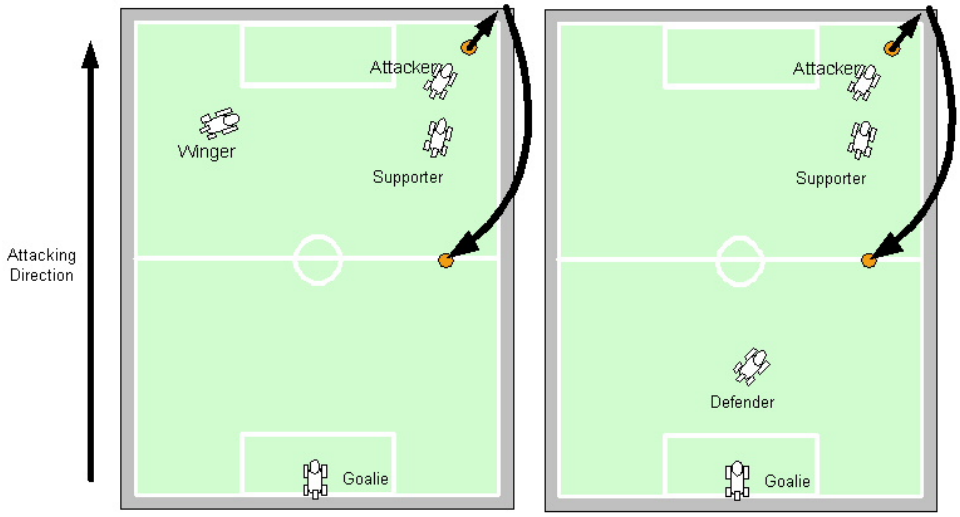

Figure 7.2: Left : 2003 formation, Right : 2005 formation

other. The supporter employs the back off mechanisms to improve the local interaction. The back off mechanisms were described in section 6.3.2.

7.2.2 Local Interaction Between Attacker and Supporter

This year, the local interaction between the attacker and the supporter was heavily relied on the effectiveness of the dynamic role switching and the back off mechanisms which the supporter role had employed. Most of the time, our dynamic role switching is capable in determining the right attacker and supporter. However, in some unexpected cases, it assigns incorrect roles and this usually leads to confusions among the attacker and the supporter and brings an obstruction to each other. To mitigate or resolve this kind of situation, the back off mechanisms are integrated inside the supporter role to not interfere with the attacker. In addition to this, we must also mention that the defender also employs the same back off mechanisms which the supporter has.

For more information on the back off mechanisms, please refer to section 6.3.2.

For more information on the dynamic role switching, please refer to section 7.4.

7.2.3 Minimum Supporter Back Line

Almost all the time, a supporter always tries to back up the attacker and the ball, so that the supporter can quickly switch to the attacker role to get hold of the ball when the attacker loses possession. However, when the ball is in one of the bottom corners of the field, a supporter positioning changes a little. The 2003 rUNSWift team had a good defensive support positioning where two robots, presumably the attacker and the supporter, move in a defensive circular path to push the ball up the field along the field edge. This is shown in the left diagram of Figure 7.3. This positioning was very effective in 2003, because robots could use the walls to push the ball along the field edge. However, since we had no walls around the field this year, the ball could easily get kicked out if we have employed this defensive positioning system.

Instead, we came up with an idea called minimum supporter back line as shown in the right diagram of Figure 7.3. This line indicates the furthest back line a supporter can move back to when the ball is in the defensive third of the field.

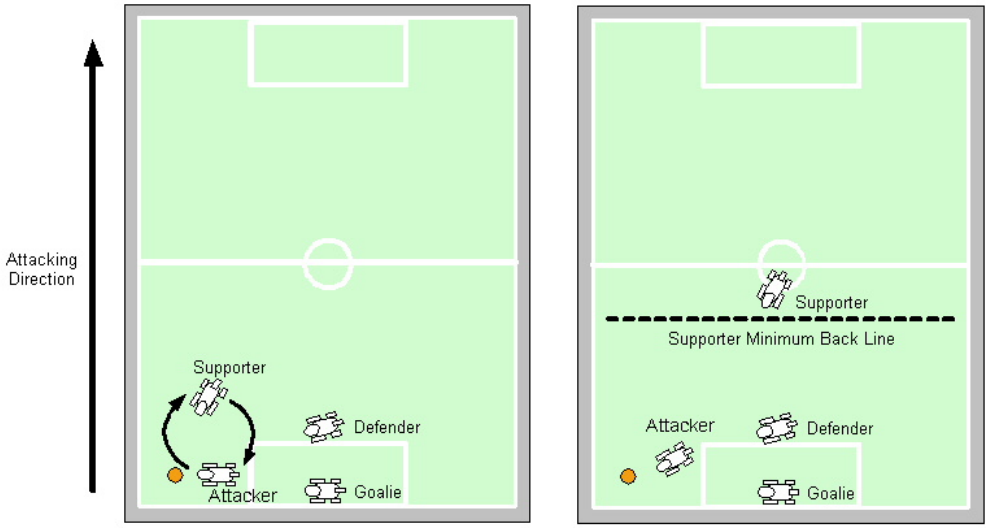

Figure 7.3: Effect of minimum supporter back line

The purposes of a supporter staying close to the centre of the field are:

1. To avoid the supporter and the defender getting too close to each other. Figure 7.3 on the left shows the supporter and defender getting bunched

up. If they come close together, then they may start flickering roles between the supporter and defender which unstabilises the role assignment. Thus, by placing the supporter near the centre of the field, we implicitly avoid this role confusion problem.

2. To let the supporter counter attack when the ball is kicked or popped out to the centre of the field. When the ball gets kicked out to the centre of the field, the supporter can quickly become the attacker and this creates an opportunity for us to attack. This is also useful when an opponent kicks out the ball into our defensive half, as the ball gets placed near the centre. Thus, the supporter can get to the ball quickly from its position.

7.3 Striker / Defender Formation

7.3.1 Overview

This formation was initially designed to check whether the small incremental changes or improvements we made to our behaviour were not converging into a local minima, instead of converging into a global optimum. Because we have always tried to improve by playing against ourselves, we tended to modify the behaviour to play better against teams who play in similar styles to us, but not others who play differently. In any competitive games like soccer, we do not play against ourselves all the time to improve. To improve, we play against different teams, and this allows us to learn what kind of techniques work better on those teams and we can learn their techniques as well. Thus, this new formation was implemented to simulate or clone a team which plays differently from our default formation, Supporter/Defender formation. Although the only difference was the formation behaviour and the underlying skills and low level modules were kept the same, it was still a good indication of whether our behaviour could play against teams who had different styles.

This Striker/Defender formation has a striker role instead of the supporter role in Supporter/Defender. A striker stays on the opposite side of where the ball is. Thus, its role responsibility is rather passive if we compare against the supporter which aggressively supports the attacker closely from behind. This "spread out" type of formation is commonly used by many other teams and playing against this Striker/Defender formation tests whether our Supporter/Defender formation can play well.

Since the formation is more spread out than the Supporter/Defender formation, the dynamic role switching and the back off mechanisms do not have to be well tuned to run this formation. Thus, they are less of an issue when using a spread out strategy like the Striker/Defender formation. Whereas, in the Supporter/Defender formation, the dynamic role switching and the back off mechanisms are very crucial components in getting the local interaction and cooperation between the attacker and a supporter correct.

7.3.2 Field Coverage Effectiveness

In theory, the striker role can give a wider coverage of the field, due to its positioning. As we can see from Figure 7.4, the coverage shown on the left is much better than the one on the right. However, this wider coverage is only effective when the ball moves to the side in which the striker stays.

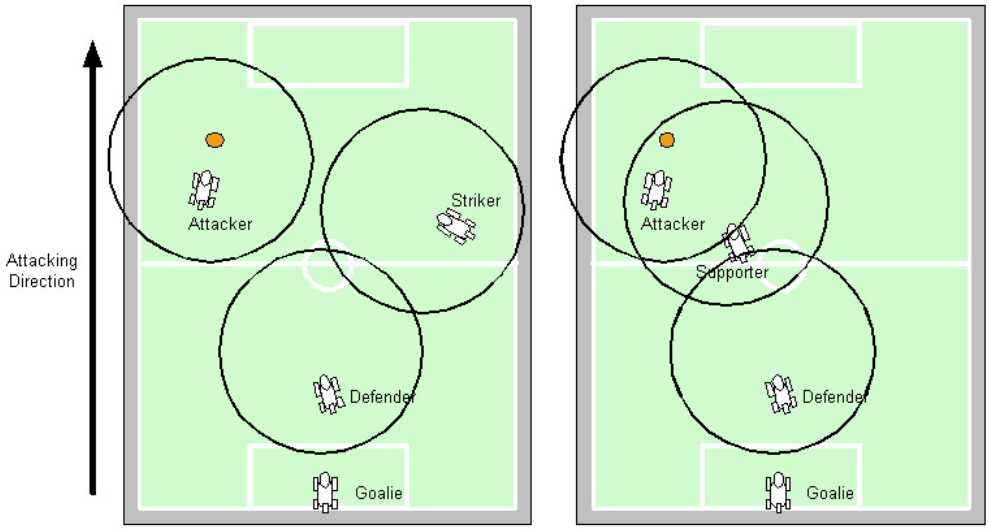

Figure 7.4: Field coverage effectiveness of striker positioning

If the ball is more likely to stay on the attacker's side, the supporter role is preferred rather than the striker role, because the coverage of where the ball is in is focused by both attacker and supporter. This can be seen on the right formation of the same figure. The two region coverage circles of the attacker and the supporter are overlapped to show the better coverage of a specific region of the field.

7.4 Dynamic Role Switching

Dynamic role switching have been continuously revised and improved since the beginning of the 4 legged league by numerous teams. In 2000, the UNSW United [UNSW United 2000] (now known as rUNSWift) had an implicit role switching mechanism which switched roles between the attacker and the supporter using visual robot detection. In 2003, the rUNSWift team [rUNSWift 2003] introduced switching mechanism which employed both visual and wireless information. They had used both vision and wireless information to assign roles to individual robots. They argue in their thesis that finer control of switching roles was achieved by using vision rather than wireless, because vision could refresh its information faster than the wireless. This finer control was particularly useful for local interactions between the attacker role and the supporter role.

This year, we have moved away from using vision information and used wireless as the fundamental source of information in deciding the role assignment. One of the main reasons why we did not use vision information this year was because of the poor detection of close obstacles which are within 50cm range of the robot. Because our vision module used shadow information for detecting obstacles, the robot usually detected its own shadow seen on the image as obstacles. Thus, detecting nearby obstacles was unreliable and the role switching mechanism used by the 2003 rUNSWift team could not be adapted to this year's code.

In addition to this fundamental change in the role switching system, we have also tried to generalise the way of switching roles dynamically. Thus, when we create a new role behaviour, we can use this generalised switching framework to integrate the new role into formations. This was particularly true, when we introduced the new role called striker.

Dynamic role switching is very advantageous as each field player is not fixed to a particular role and can change to an appropriate role in different situations. However, this switching mechanism must be tuned well enough, so that the robots play as a team. Otherwise, a situation where two attackers chasing after the same ball causing both players to be in the way of each other, or other disastrous situations could arise.

7.4.1 Time to Reach Ball

Although this idea has been incorporated into different teams' role switching systems for several years, we believe that the German Team has nicely formulated and documented this idea in their technical report in 2004 [German Team 2004].

The three field players send each other their own time to reach ball information through wireless, and the player with the lowest time to reach ball attacks the ball. The other two players are assigned to a supporter and a striker. The formula which the German Team has used in calculating the time to reach ball was the following:

$$timeToReachBall = distanceToBall/0.2$$
$$+400.0 * fabs(angleBetweenBallAndOpponentGoal)$$
$$+2.0 * timeSinceBallWasSeenLast$$

To stabilise the role assignment for the attacker role, there was also 500ms bonus for the robot who was previously assigned to the attacker role. Thus, the same robot has more chance of staying as the attacker.

This year, we have used this concept and extended it to suit our role switching system. The formula which we have used to calculate the time to reach ball was the following:

$$timeToReachBall = attackerBonus$$
$$+grabbingBonus$$
$$+timeToTurnTowardsBall$$
$$+timeToWalkToBall$$
$$+timeToTurnToDAD$$
$$+obstacleDelay$$

attackerBonus - This bonus is given to the field player which was assigned to the attacker role in the last frame.

grabbingBonus - This bonus is given to the field player which has grabbed the ball. When the player grabs the ball, it must stay as the attacker to do something with the ball. Thus, this grabbing bonus must dominate over other factors involved in calculating time to reach ball so that the player is guaranteed to stay as the attacker.

timeToTurnTowardsBall - The estimated time taken to turn towards the ball from the current heading of the robot.

timeToWalkToBall - The estimated time taken to walk towards the ball from the current position of the robot.

timeToTurnToDAD - The estimated time taken to turn towards the DAD after the robot has reached the ball.

obstacleDelay - The estimated time delays caused from obstacles detected in between the ball and the robot.

This calculated time to reach ball for every robot is sent around to each other during the game. Thus, when a player runs a role switching algorithm, it has estimated times for all the other players through wireless messages.

7.4.2 Time to Reach Position

Time to reach position is the generalised idea of the time to reach ball discussed above. With this, it is possible to apply to any position on the field to determine the estimated time to reach the specified position from where the robot is. We have used this to determine other roles including supporter, defender and striker.

For example, to determine which robot should be a supporter, we first calculate the estimated time for two non-attacker field players to get to the supporter position. Then, we compare these estimated times and the one who is the closest to the supporter position becomes a supporter. The other player who did not get assigned to a supporter gets assigned to a defender. The same technique applies when deciding the striker role.

7.4.3 Role Assignment Criteria

Although time to reach ball and time to reach position are the core of our dynamic role switching mechanism, there are other several important criteria which are essential for assigning roles correctly in some special cases. This section provides algorithms used in the role switching and describes each criterion included in the role switching algorithms. Each role has a separate algorithm to be executed to decide whether a player is appropriate to play that role. All the algorithms which are discussed below return eit her True or False. True indicates a player should be assigned to a role and false indicates it should not be assigned.

There is a specific order or priority of roles which must be followed in assigning each robot a role. For Supporter/Defender formation, the order of role assignment is Attacker, Supporter, Defender. For Striker/Defender formation, the order is Attacker, Striker, Defender.

Attacker Role Criteria

Algorithm 7.1: Attacker Role Criteria

Data:
Result:
1 **if** *there is no other attacker on the field* **then**
2 | return True
3 **end**
4 **if** *I am grabbing the ball* **then**
5 | return True
6 **end**
7 **if** *my estimated time to reach ball is the smallest among all the field players OR my estimated time is less than 1500ms* **then**
8 | return True
9 **end**
10 return False

The first criterion checks whether there is no attacker on the field or not. If there is not, then the attacker role is assigned.

The second criterion checks if a player is currently grabbing the ball or not. If it is, then it should continue its role as the attacker until it releases the ball. In fact, there is no reason to be some other role like supporter, striker or defender, because the player actually has possession of the ball. Thus it should continue to grab and attack the ball. Although this grabbed situation is also handled by the grabbing bonus included in the time to reach ball, we still have the second criterion for its simplicity and just in case.

In section 7.4.1, we have discussed the time to reach ball. The third criterion uses this estimated time to the ball and compares it against other field players' estimated times to check who has the smallest time. A player with the smallest time becomes the attacker. In addition to this, we would also allow a player with the estimated time less than 1500ms, because the time less than 1500ms implies the player is very close to the ball.

Supporter Role Criteria

Algorithm 7.2: Supporter Role Criteria

```
   Data:
   Result:
 1 if there are at most two field players on the field then
 2 |  return False
 3 end
 4 if there is an attacker and no supporter then
 5 |  return True
 6 end
 7 if there is no attacker and a supporter then
 8 |  return True
 9 end
10 if I am the furthest back on the field by far then
11 |  return False
12 end
13 if my estimated time to reach defender position is the largest among
   all the non-attacker field players then
14 |  if there are at two supporters on the field then
15 |  |  if a player needs to break the symmetry then
16 |  |  |  return False
17 |  |  end
18 |  end
19 |  return True
20 end
21 return False
```

The first criterion is important when we have one field player missing from the field for some reason. In this situation, it is not desirable to have the supporter backing up the attacker, rather the defender is required to cover any unexpected counter attacks or ball out situations. Thus, this criterion stops a player being a supporter when there is a third field player missing from the field.

The second criterion allows a player to be a supporter if there is already an attacker, but no supporter.

The third criterion checks for no attacker and one supporter. When this

is true, a player will be assigned as a supporter. This means that we will get two supporters and we hope that eventually either one of them will become an attacker as they come close to the ball. This criterion will not be fired, if the attacker criterion which checks for no attacker works properly. This third supporter criterion is there for just in case.

The fourth criterion checks if a player is the furthest back among all the other field players. "by far" means the player must be at least 30cm behind all the other field players.

The last criterion checks if a player has the largest time to reach defender position among all the non-attacker field players. If there is another supporter already assigned, it breaks the supporter symmetry by comparing time to reach supporter position.

Striker Role Criteria

The striker role criteria is exactly the same. We have only replaced the term "supporter" to "striker", and none of the criteria has changed.

Algorithm 7.3: Striker Role Criteria

```
   Data:
   Result:
 1 if there are at most two field players on the field then
 2 |  return False
 3 end
 4 if there is an attacker and no striker then
 5 |  return True
 6 end
 7 if there is no attacker and a striker then
 8 |  return True
 9 end
10 if I am the furthest back on the field by far then
11 |  return False
12 end
13 if my estimated time to reach defender position is the largest among
   all the non-attacker field players then
14 |  if there are at two strikers on the field then
15 |  |  if a player needs to break the symmetry then
16 |  |  |  return False
17 |  |  end
18 |  end
19 |  return True
20 end
21 return False
```

Defender Role Criteria

At the moment, the two formation behaviours, both have a defender as the lowest priority role. Thus, if any of the higher priority roles was not assigned to a player, then this defender role gets assigned.

7.5 Wireless Down Strategy

Because our dynamic role switching mechanism depends on information sent around players through wireless signals, we must consider and be able to handle a situation when wireless network is down and no communication between players is possible. The strategy which handles this situation is called the wireless down strategy.

We have a wireless down checking function to be called on every vision frame by each robot. This function is defined by no wireless messages received from the other players in the last 45 vision frames which equates to 1.5 seconds, and no game controller data received in the last 2 seconds.

If a robot detects the wireless network is indeed down by invoking the checking function, then it assigns itself a fixed role which is derived from its player number. A player with the number two or three fixes its role to an attacker and four fixes its role to a defender. Thus, in the wireless down strategy, we have neither a supporter nor a striker.

Not to mention that a player with a number one always stay as a goalie, because it runs a goalie behaviour and it does not use the dynamic role switching system at all.

7.6 Distributed Ball Finding Strategy

This strategy is only fired if none of the players can find the ball for a while. The strategy distributes the field players to fixed positions to cover the area of the field as much as possible to look for the ball quickly.

Each player sends each other a wireless information on whether it has lost the ball or not in a boolean value. A player will send true if it has lost the ball for more than 66 vision frames. Once a player receives that all the teammates have lost the ball, it activates the distributed ball finding strategy. As shown in Figure 7.5, the furthest player back will go to position 1, the other player on the left hand side of the field will go to position 2 and the remaining player will go to position 3. The goalie always stay inside the goal box when this strategy is firing.

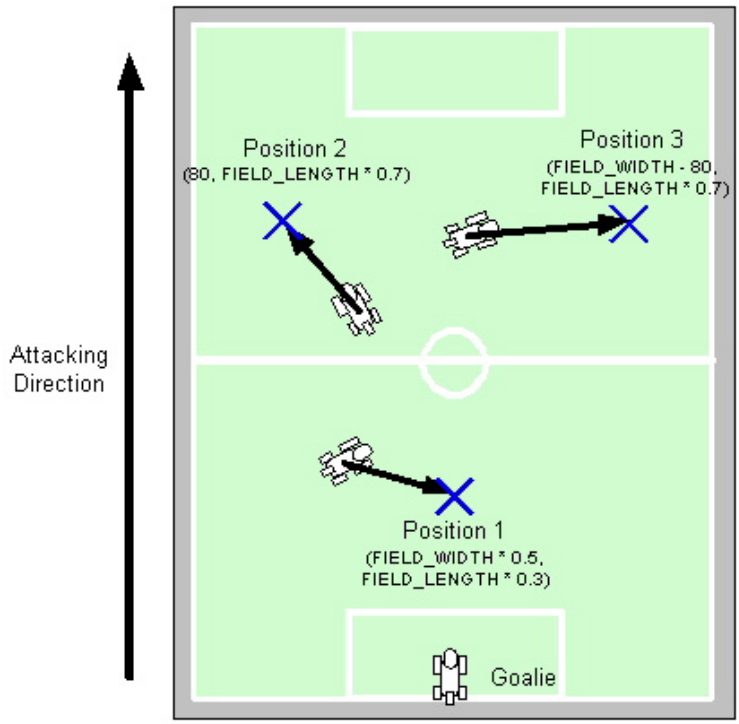

Figure 7.5: Distributed ball finding strategy

Chapter 8

Conclusion

8.1 Until Here ...

8.1.1 RoboCup 2005 Competition Results

First Round Robin

Team played	Score
Dutch AIBO	4 - 1
Cerberus	9 - 0

Table 8.1: Competition result : First round robin

Second Round Robin

Team played	Score
UChile1	10 - 0
UPenn	6 - 1
ARAIBO	7 - 0

Table 8.2: Competition result : Second round robin

Finals

Team played	Score
Austin Villa (Quater Final)	7 - 1
NUBots (Semi Final)	2 - 5
CMDash (3rd Play Off)	8 - 0

Table 8.3: Competition result : Finals

8.1.2 Competition Remark

This year, we, rUNSWift, have scored the most goals in the 4-legged league competition. We scored a total of 53 goals. This score count included the first round robin, second round robin and all the finals. The second team to score the most goals was our rival team, NUBots with 52 goals. rUNSWift

and NUBots were the only two teams who scored this many goals. We believe the use of the ball grabbing and grab dribbling skills had played an important part in scoring goals, as we were able to keep possession of the ball well and this had led to many goal scoring chances.

This year's overall attacking strategy was modified to adapt to the field which did not have walls. The modifications which were mentioned in this thesis had made a significant contribution and improvement in how robots played soccer this year. Such modifications included the ball tracking and finding, the ball grabbing and dribbling, the role behaviours, and the high-level strategy behaviours. As a result of these modifications and improvements, we successfully came third in the world in the 2005 RoboCup Osaka.

8.2 From Here ... Road To RoboCup 2006 Bremen

Next year's RoboCup will be held in Bremen, Germany where FIFA World Cup 2006 is also going to be held. We believe that there will not be any significant changes to the field for next year in terms of size and layout. However, there is a discussion on where the ball should be placed when it goes out of the field. This is likely to change and some of our assumptions which we had in our rUNSWift architecture this year may not be applicable to next year. Thus, it may involve significant changes in the behaviour module and this is something which we cannot avoid. The behaviour module is the most dynamic and volatile module which needs changes and improvements all the time.

Lastly, to conclude, we hope and believe that the insights which we have documented in this thesis will still be useful in the years to come as the foundation in understanding the behaviour module.

Bibliography

[German Team 2004] *German Team 2004 Technical Report.*

[rUNSWift 2003] J. Chen, E. Chung, R. Edwards, N. Wong. *Rise of the AIBOs III - AIBO Revolutions.* Bachelor Thesis, The University of New South Wales and National ICT Australia.

[rUNSWift 2002] Z. Wang, J. Wong, T. Tam, B. Leung, M. S. Kim, J. Brooks, A. Chang, N. V. Huben. *UNSW RoboCup 2002 Sony Legged League Team.* Bachelor Thesis, The University of New South Wales.

[UNSW United 2001] S. Chen, M. Siu, T. Vogelgesang, T. F. Yik, B. Hengst, S. B. Pham, C. Sammut. *The UNSW RoboCup 2001 Sony Legged League Team.* Technical Report, The University of New South Wales.

[UNSW United 2000] B. Hengst, D. Ibbotson, S. B. Pham, C.Sammut. *Sony Legged Robot Software System.* Technical Report, The University of New South Wales.

[W. Chan] W. Chan. *What's the title?* Bachelor Thesis, The University of New South Wales and National ICT Australia.

[A. North] A. North. *Object recognition from sub-sampled image processing* Bachelor Thesis, The University of New South Wales and National ICT Australia.

[J. Shammay] J. Shammay. *Real-Time Shared Obstacle Probability Grid Mapping and Avoidance for Mobile Swarms* Bachelor Thesis, The University of New South Wales and National ICT Australia.

[A. M. Sianty] A. M. Sianty. *What's the title?* Bachelor Thesis, The University of New South Wales and National ICT Australia.

[MISC : UTS Unleashed!] UTS Unleashed! *Participation in 2004 RoboCup Portugal* University of Technology, Sydney.