Thesis B Report

rUNSWift 2004
The University of New South Wales 2004 RoboCup team

Chi Kin CHAN

3019932

15th September 2004

Table of Contents

Chapter 1

Introduction

1.1 Abstract

This thesis report mainly presents the behaviour level of the software system developed for the rUNSWift team to compete in the Four-legged League of the RoboCup Competition 2004. The whole team strategies and some of the skills used or attempted will be described in details, along with the problems that have been encountered. All the changes from previous years will also be discussed and the reasons for the changes are provided as well.

1.2 Background

The Four-Legged League is one of the five leagues in the RoboCup soccer competition. In this league a team has to use the Sony ERS robots to compete with other teams. Each game consists of two 10 minutes halves, with an additional penalty shootout if a tie happens. The team that scores the most goals in a match will win. Each team has four robots; three are forwards and the other is a goalie.

The robots that comprise this year's team are the new ERS-7 Sony Entertainment robots. No physical modifications on these robots are allowed, so the competition is purely based on software implementation.

1.3 Changes to the Competition and Rules

Beside the robot model change, there are a number of other changes to make the competition more challenging. The soccer field used in this year is shown in figure 1.1. All the rules are specified in the rule book of this year [3].

1.3.1 Border Height

The height of the white border surrounding the soccer field was reduced from 60 cm to 30 cm, which poses more challenges for the robot visual object recognition system because the robot cannot recognize any false objects in the space above the border.

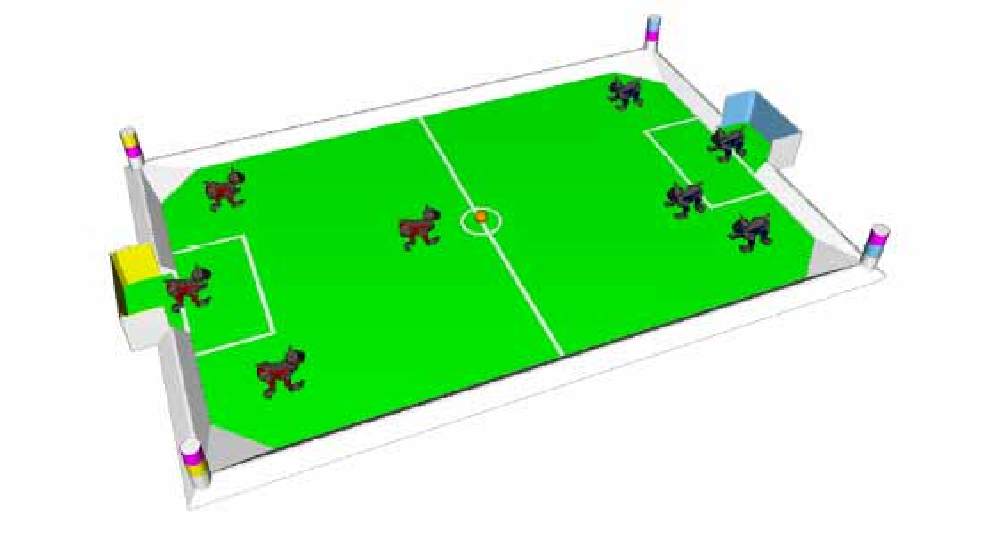

Fig 1.1 The competition field of this year competition.

1.3.2 Beacons

The two beacons located at the middle of the field were removed. The remaining four beacons at the corners of the field remained. It is intended to advance the localization technique.

1.3.3 Obstruction

In this year the definition of obstruction is redefined. Obstruction is called only when a robot is actively and intentionally blocking another robot from moving. So if a robot is being pushed, it is not called for an obstruction but the other robot is more likely called pushing.

1.3.4 Pushing

The pushing rule is called when any robot pushes another robot for more than 3 seconds, it will be penalized for 30 seconds field removal and then replaced at the halfway line. If two robots are charging into each other, both are called for pushing. However there a few exceptions to the pushing rule:

- The closest robot to the ball on each team, if it is within 2 dog-lengths of the ball.
- A robot standing still.
- If the goalie is within its goal box with least two legs.

The pushing rule cannot apply to the above three cases.

1.3.5 Automatic Start-up Position

During the ready state, all the robots are required to return to its start-up positions. If the robots cannot return to their start-up positions, manual placement is allowed but penalty is imposed by moving the robots further back. For implementation of this please consult [13].

1.4 System Architecture and Development

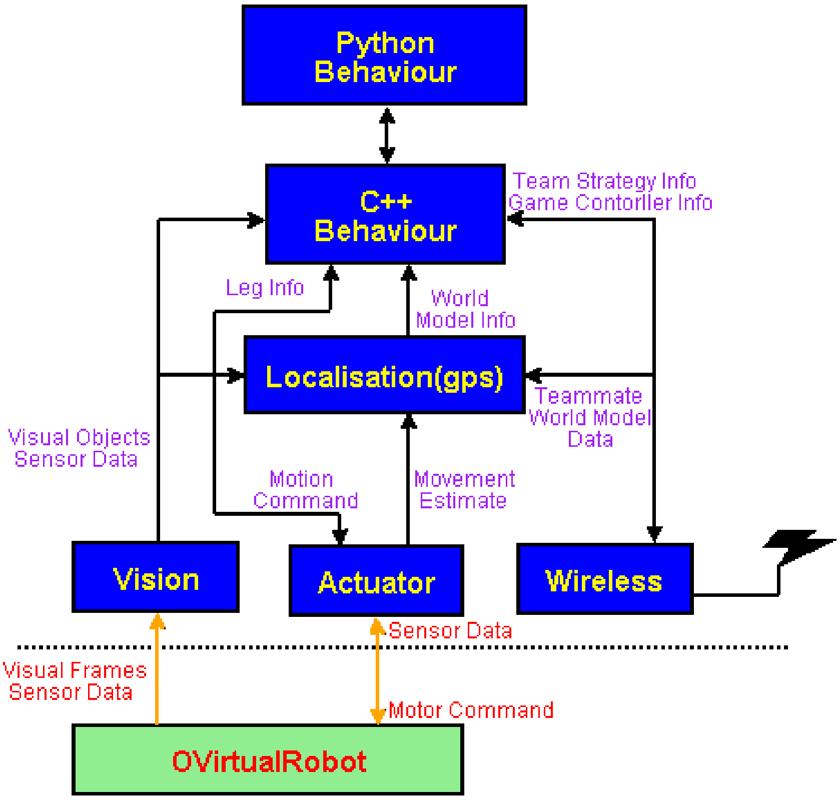

Fig 1.2: Software system architecture for the rUNSWift robot

The above figure demonstrates the overall architecture of the software system executed on the robots. This thesis report will concentrate on the top levels in the diagram, namely the behaviour. Before the Australian Open, the behaviour only exists in C++. Afterwards, due to the need to speedily and efficiently developing the behaviour and strategy, most of the behaviour code was ported to Python with a C++/Python interface.

1.5 Behaviour Architecture

The behaviour architecture is primarily a huge decision tree that has numerous branches. It accepts information from all different modules and then determines the next set of actions that a robot should perform. The tree is hierarchical in nature, which can be distinguished into several different layers.

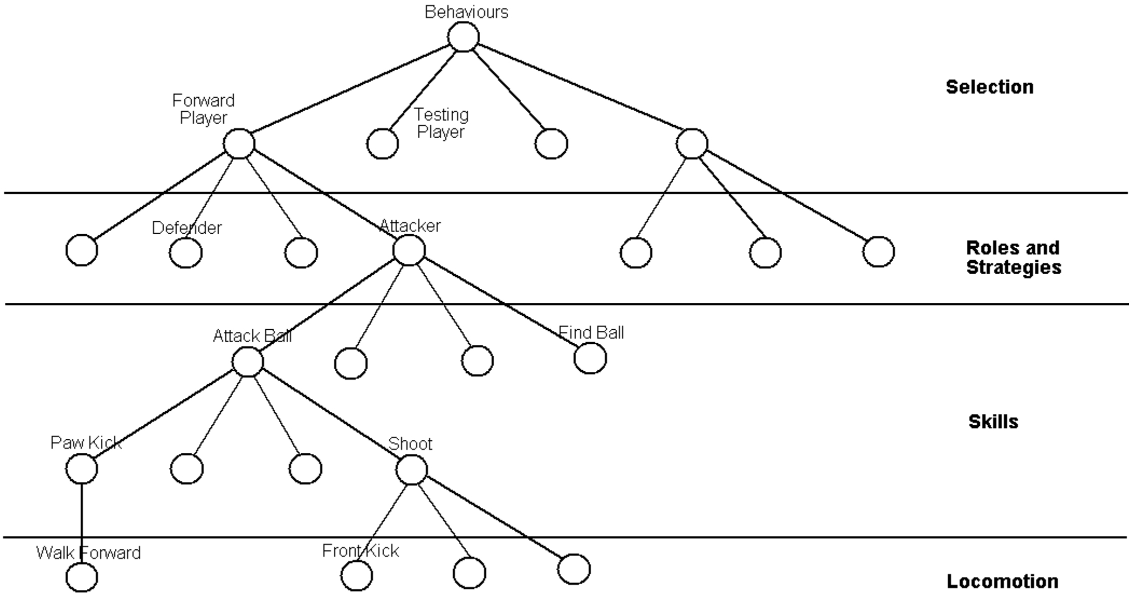

Fig 1.3 Behavioural architecture can be visualized as a huge decision tree.

The first layer handles some fundamental decisions such as accepting the game controller data, interfacing between C++ and Python, choosing different players, and forwarding messages to the actuator module. This layer exists in both Python and C++.

The next layer begins the overall strategy of game playing, in which the robot will dynamically decide what role it should take depending on the information it perceives from the environment. The roles that a robot can take include the defender, striker, back-off player, supporter and attacker, in the descending order of priority. The goalie is a fixed role for the whole duration of the game. The local and global team interactions are carried out by all these roles.

One layer further below the team tactics are concerned with skills and techniques that each role uses in fulfilling its responsibilities. Some complex skills can incorporate other smaller and simpler skills. Most of them are some computations that determine simple movement which in turn specifies the actuator parameters such as going forward and turning left, head panning and tilting, kicking, as well as the pose parameters, such as the back height. These movement and pose parameters are sent to the actuator control module at the end of the behaviour tree. This process of running through the decision tree is continually repeated at a rate of 30 times per second, also known as the vision frame rate.

In this year it is seen that the development of the robot’s behaviour totally in C++ is not manageable and extensible, because the process of compilation, copying the executables onto the memory stick and re booting the robot consumes a significant amount of time, if not tedious, thus it is not suitable for rapid development. Although the use of simulation as in [4] could speed up the development, the simulated environment could be so much different from an actual game play and the computed result would be useless.

The introduction of Python in defining the robot’s behaviour is very effective. One can modify the behaviour, upload the file to the robot through wireless network and reload the code to test and verify the behaviour. It has simple and highly readable syntax. Given that there are interpreter and a number of development environments widely available and the compatibility with C++, it is ready to use in our case. Yet long execution time is an identified problem. [11] details the Python introduction in this year’s system. In the future the whole behaviour should be ported to Python so that there would not exist the confusion whether a decision is made in Python or C++.

1.6 Literature Survey

Most of the researches done are based on the thesis reports from the previous two years [1, 2]. They are the good sources for team strategies analysis and skill description. For PID control theory used in the directional paw kick, the CMU and University of Michigan have a joint online tutorial even including a number of MATLAB examples [5]. Other sources team strategies and skills that have been used in the research can be found in University of Pennsylvania [6], University of Newcastle [7] and the German Team [4]. A lot of data and programming manuals of the ERS-7 are required as well, which can be found in OPEN-R [8].

Section 1

Skills and Elementary Decision

In the following few chapters, the skills and the elementary decisions of the rUNSWift robots are explained and elaborated. At the end of each chapter there are also discussions of the current implementation and the possible future improvements.

Chapter 2

Desired Kick Direction (DKD) and Range

The desired kick direction (DKD) is a global heading originated from the ball that specifies which direction the ball should travel to maximize the chance of scoring a goal at any position on the field. It is calculated every frame to reflect the latest update from the vision and gps module. The DKD is used in a lot of techniques, like the visual back off decision and the directional paw kick.

There are a number of changes to this year's DKD calculation. First of all the DKD does not only specify a direction, but also two ranges, a narrow one and a broad one. The motivation of the narrow DKD range is to indicate the correct range of directions for moving the ball forward, rather than having the robot try to line up to the exact DKD. It is especially useful when the ball is in front of the target goal, where the robot is already facing the goal with the ball in front. In this case the robot should not try to waste time and line up to the DKD, but kick as long as its heading lies inside the narrow DKD range.

However, the idea of narrow DKD range does not mean the DKD itself is not applicable. The DKD is still very important in defining the DKD ranges, and in defining the back off regions and generating the vector filed for the directional paw kick, for example.

The broad DKD range, however, is less often used. It is used basically to warn the robot that outside the broad DKD range it is heading in the wrong direction, that is the robot should never kick the ball outside the broad DKD range.

There are several sections of the field in which the DKD and its ranges are found:

- For the left side region, the DKD is 90^o, the narrow range is (70^o, 100^o) and the broad range is (10^o, 110^o).
- For the right side region, the DKD is 90^o, the narrow range is (80^o, 110^o) and the broad range is (70^o, 170^o).
- For the lower 40% of the field, the DKD radiates out from the centre of the own goal, such that the ball will travel out of the own goal, the narrow range is (20^o, 160^o) and the broad range is (10^o, 170^o).
- For the upper 40% of the field, if the robot can see the target goal with no more

than 3 obstructing robots, it will use the “visual opponent avoidance kick” (VOAK) calculation [2] to find the global left and right headings of the largest gap for the target goal, which becomes the narrow DKD range. The DKD is the average of these two headings. Otherwise if it does not use the VOAK calculation, the DKD is directed into the centre of the target goal, with the narrow range being the headings to the left and right goal posts. The broad range is (10°, 170°).

- For the middle 20% of the field, the DKD is the linear interpolation of the above 2 calculations (combining the upper and lower 40%), such that it produces a more gradual change in the middle region. The narrow range is (20°, 160°) and the broad range is (10°, 170°).

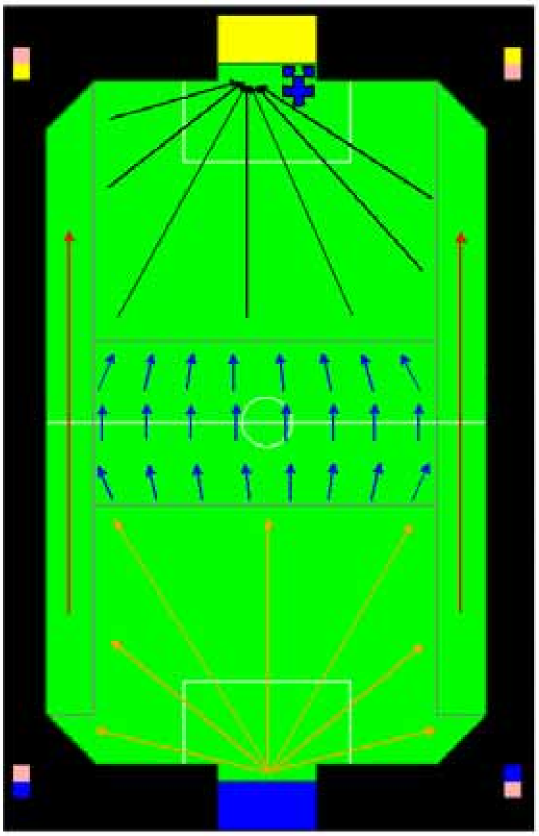

Fig 2.1 The vector field of the DKD. Notice the upper region shown is VOAK calculation.

Future Improvement

One aspect that the DKD has ignored is the opponent avoidance in a general play. VOAK calculation near the target goal merely maximizes the chance of scoring but to avoid ball captured by opponents and entanglement with other robots in the whole field the DKD calculation should take into account of other visual robots. This relies on good estimates of robot distance in mid range. For details of robot recognition and distance estimates, please consult [12].

Chapter 3

Dynamic Motor Gain

Before the Australian Open, the ERS-7 robot was still suffering from a frequent hardware safety crash, battery over current, that caused the major difficulty for the development. Even when the normal walk is optimized, the robot was not likely to sustain for more than 3 minutes in a practice match.

In fact, out of the three kinds of movement (forward, side walk, turn), for a monotonous movement without obstacle stuck and collision for a 12 minute duration, experiments showed that side walk with turning was the only movement that produced the battery over current within the 12 minutes period.

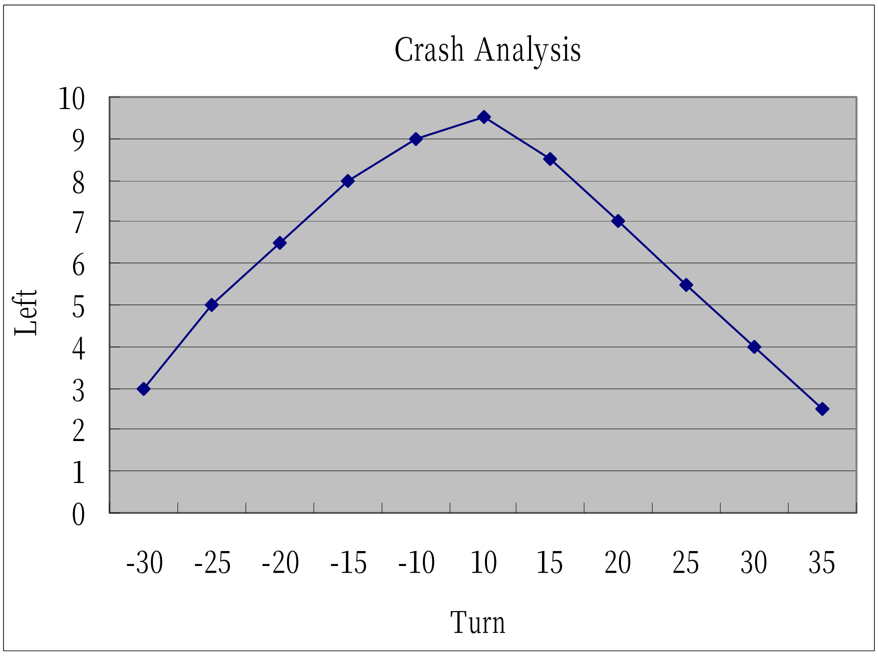

Fig 3.1: given a fixed turn, find the highest side walk value that will crash the robot.

The relationship shown above is approximately linear, hence a simple way to avoid a crash is to cap the turn and side walk according to a linear equation.

Another way to combat these adverse effects is a simple hysteresis motor gain control used to avoid these sudden crashes. When the battery current is above a certain upper limit and the current gain is high, the robot will switch to a set of low motor gains. Conversely, when the battery current is below a certain lower limit and the current gain is low, the robot will switch to a high set of motor gains. Also when the robot is about to perform any kick, such as the “Upenn” kick, the gains are switched to high such that the kick is powerful and effective. The motors involved in this mechanism are the twelve leg motors, as they are the only ones that draw the majority of the robot’s power. The gains of the head motors are always high because it needs to perform various motions in a short response time.

Conclusion

Changing the motor gains will introduce a glitch to the robot movement, which is not noticeable under usual game condition but can be clearly observable if the change is intentionally controlled through the commander. This is because the set of gains differ by significant amount. In the actual game play dynamic motor gain is effective and most of the times the robot is actually using high gains and there are no instances continuous glitches due to fluctuation between the gains. Therefore, the idea of introducing a continual gain control, that is varying the gains every actuator frame in a continuous fashion according to the battery current and/or the PWM duties of the motor joints, is not attempted. The turn/side walk cap is also not used after the Australian Open as the new and stable version OPEN-R is released and optimized gait is available.

Future Improvement

One future improvement is rather varying the gains straightly based on battery current, the strategy should be taken for consideration as well, such as forcing the high gain for ball chasing when opponent robots are observable and close by, forcing a weak gain when involved in scrum and “corner game”. This let the robot to decide when to exert the maximum power for certain behaviour.

Chapter 4

Stuck Detection and Resolution

The motivation of implementing a stuck detection is to:

- Prevent the robot crashing and leg entangled (also known as "leg lock") into other robots that could lead it being called for pushing and penalized by field removal for 30 seconds. Crashing into own robots effectively destroys its own team tactics.
- Avoid getting lost of direction and position. Once the robot is stuck, the robot odometry update does not reflect the stuck situation and the position and direction estimates will be incorrect.
- Avoid hardware crash.

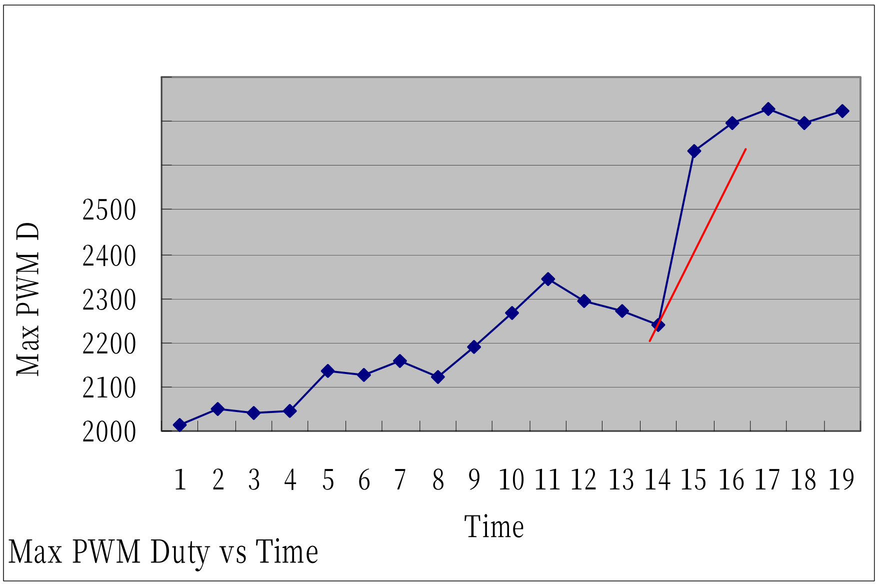

Fig 4.1 Stuck detection by PWM duty value. The red line is the gradient threshold which indicates a possible stuck.

When a robot is involved in collision and stuck, there will be a number of changes occurring that could serve as indicators. First of all, the PWM duty of motors belonging to the stuck leg will be high due to mechanical resistance. But in a fierce game the PWM duty is building up constantly to a high level even without any stuck situation. Thus, a better scheme is to detect if there is a significant increase in the differential in the PWM duty. This is done by comparing the highest PWM duty of the present frame against the highest PWM duty 60 frames earlier, see figure 4.1. To improve the alarm accuracy, the same check is repeated for 5 consecutive frames. In order words, if the present frame number is x, then the comparisons are between the frames x and x-60, x-1 and x-61, and finally up to x-4 and x-64. It concludes a possible stuck situation if all the 5 differences are larger than a certain threshold.

A second sign is the change of the ball distance. If the robot is moving forward in a constant speed, the ball distance is expected to decrease over a number of consecutive frames. Thus over the last 10 frames in which the ball has been seen, if the difference in ball distance is less than zero, than the ball has not been closing in at all and it concludes another possible stuck situation. This “sanity check” is only carried out if the above PWM duty differential check is passed.

If both checks suggest a possible stuck, then the robot will start its actual detection process. It will begin a quick scanning of its surrounding space with a low tilt and low crane as well as uses the close range infrared sensor. If it can see any robot while the IR reading reflects a close obstacle, it will walk backward and concurrently side walk in the direction opposite to the current head pan, and effectively getting out from the scrum.

Conclusion

The performance of the stuck detection was initially perceived satisfactory when the first version was completed. However, due to the lack of fine-tuning during it was not accurate and too sensitive. It often incorrectly triggers the head scan which is costly in a competition and was therefore abandoned in the competition.

Future Improvement

Although PWM duty is sensitive to stuck yet its inconsistency is a major problem. Using ball distance as an additional check could be error prone since the ball distance is distorted by motion blur in the image and the ball is not necessarily available in every image. A more reliable way is to compare the sensor joint readings with the intended joint angle sent by the actuator control because this is the direct indicator of an entangled joint. [9] details this implementation and indicated the result is indeed accurate. This is an important area because it not only helps the robot to resolve the stuck and prevent leg entanglement but also reflects the correct motion update to the localisation system [14]. Another source of stuck detection algorithm is by the traction monitoring as explained in [15].

Chapter 5

Special Motions

5.1 Ball Grabbing

Grabbing the ball is an essential skill, as it is the basic building block for a number of kicking skills and aggressive dribbling. To grab the ball, the robot starts with slowing down and using the head to fetch the ball when the ball is close in front, and open the mouth to lock the ball directly underneath its chin and up against it chest. In the mean time it will bring the front paws forward to stop the ball from straying to the sides. The mouth and chin sensors will indicate whether the ball is trapped or not.

The details of the ball grabbing can be illustrated in the following diagrams.

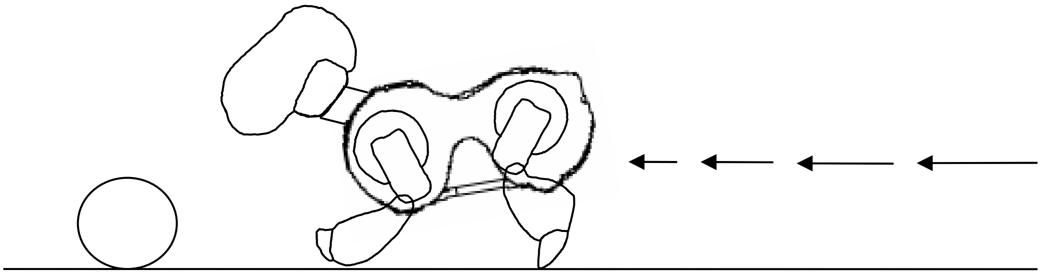

Fig 5.1: As the robot is approaching close to the ball, it slows down. Slowing down is a way to prevent the robot's chest from knocking the ball out, although it gives the opponents chances to catch up and interfere with the robot's attack.

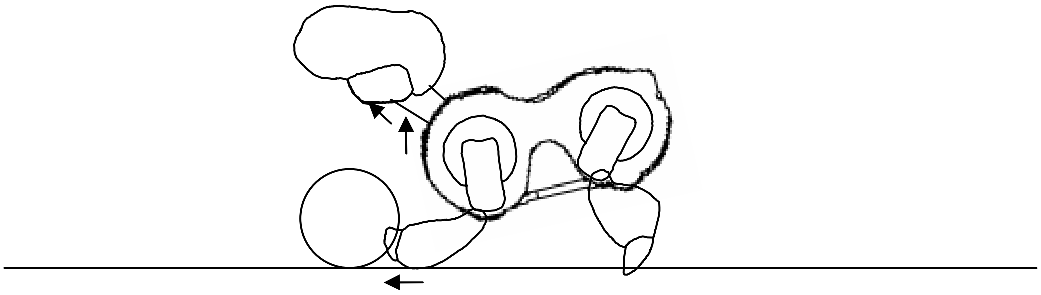

Fig 5.2: At the right position, the robot will stop and extend its neck and lift it up for

a short period of time, that is, increase the tilt and crane, in order to prepare for the next step. Without this step, when the crane increases at a low tilt angle, the snout will knock the ball out. Moreover the front paws are brought forward.

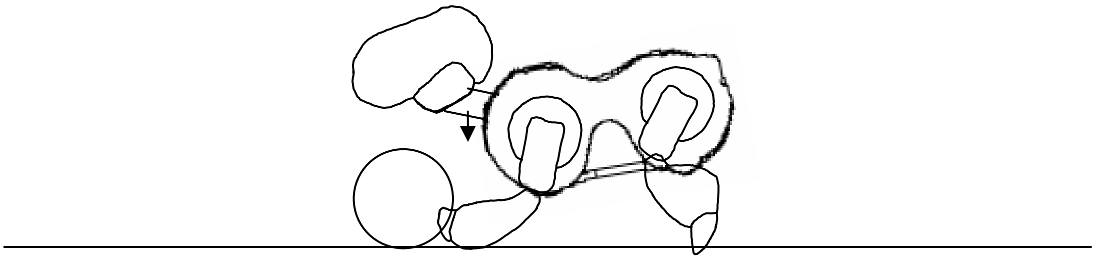

Fig 5.3: In this step the robot only lowers its tilt, maintains its crane and the stance.

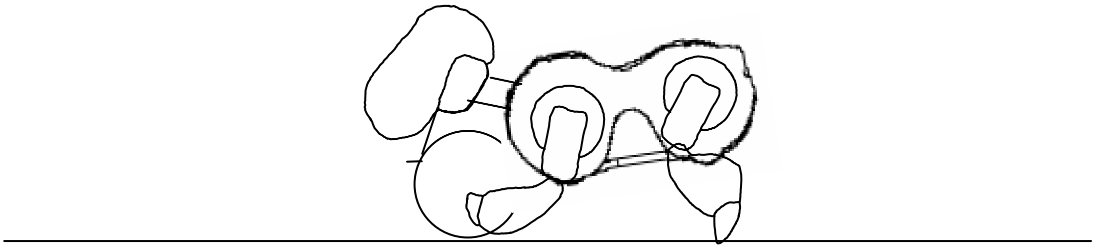

Fig 5.4: The robot opens its mouth to trap and the ball. During this time it checks whether it can see the ball, if so it means it has not got the ball underneath its chin yet, and aborts the operation. At the end, it also check the chin and mouth sensors whether the ball is sensed and if not the operation is aborted.

Future Improvement

The current ball grabbing routine needs an improvement in speed. One possible improvement that should be tried is when the robot is walking towards the ball from 2 dog-lengths, it should already extend its neck while looking at the ball. So when it arrives at the right position, it just needs to open the mouth and lowers the tilt further. In other words, the robot is already in the pose demonstrated in Fig 5.3 before the ball is close in front.

5.2 Upenn Kick

The Upenn kick was developed by the University of Pennsylvania [6] in 2003. It is a side kick that propels the ball in the heading of 60^o to 70^o. The attractive point is that it does not require any set up time, takes a small recovery time and consists of one step only. The following diagram portrays the left upenn kick.

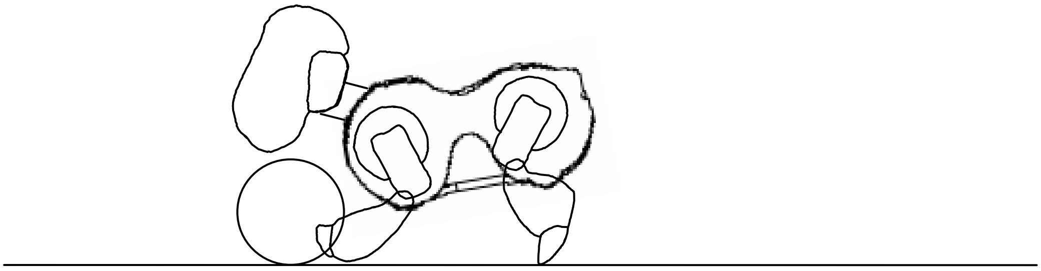

Fig 5.5: Start of an Upenn kick is an ordinary standing pose.

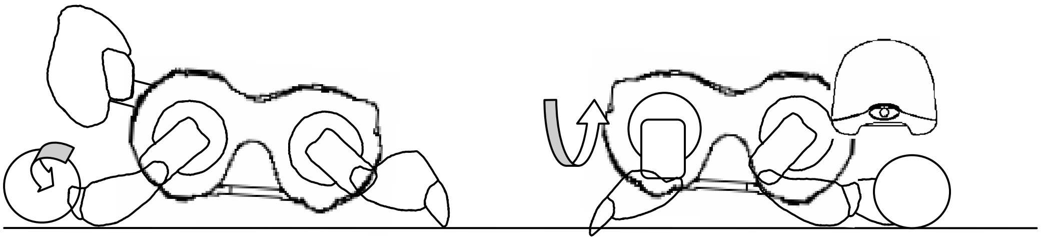

Fig 5.6: (Left) shows the left side view. The front leg is extended forward and swings toward the centre of the body, the right rear leg gives a thrust by extending and pushing. (Right) shows the right side view. The 2 right legs are contracted to allow the body to fall lower, such that momentum is built up and helps pushing the ball further. The head is fixed to look in the direction of where the ball travels. The 2 rear leg are also spread out from the body to stably support itself.

Between these two steps is a linear interpolation of the joint angles which allows the changes to be smooth and the whole movement is recorded in the .pos file.

Conclusion and Future Improvement

Although the upenn kick is powerful, it is quite hard to set up. The ball must be close to the chest and the hitting arm in order to produce a consistent result. Grabbing the ball before the kick increases the consistency. In the competition it was observed that a better kick could replace the upenn kick, which is the head swipe and were used by several strong teams. [13] details his experiment with it.

5.3 Dive Kick and One Handed Kick

Dive kick is a forward kick that is fast and very consistent in shooting the ball straight ahead. The only set up requirement is that the robot has to define the right stopping point to trigger the kick which requires the robot to slow down. On the other hand the one handed kick needs ball grabbing and takes a significant set up and recovery time. Their actuator details and implementations can be found in [11].

Chapter 6

Spin Dribble

The spin dribble was an important skill widely used in last year’s strategy to aggressively drive the ball towards the goal with the maximum speed. Since the robot needs to keep the head down to maintain the possession of the ball, the robot is not able to visually determine when to stop spinning. Thus this heavily relies on having a precise heading before spinning to a right dribbling direction. So the robot has to perform active localisation after it has grabbed the ball, in order to obtain a precise heading estimate.

The following are the detailed steps in the spin dribbling:

1. Grab the ball as described in the last chapter.
2. Change the stance such that it is not the head that holds the ball but instead only the front legs that grasp it. If this step is skipped, that is the robot proceeds straight to active localisation after the grab, the ball will usually roll away from the robot because it is not a perfect sphere.
3. Once the ball is held by the front legs, the robot active localizes.
4. After the localisation, it moves the head back to the normal grab ball pose
5. It starts spinning until it gps heading is between the two goal posts.
6. It looks up and sees if the target goal is in the view.
7. If the target is in the view, it will dribble the ball forward. If the target goal is not in the view, it will abort the dribble by performing a chest push, such that the ball leaves the robot.

Conclusion

The dribble performs reliably and consistently on this new robot. Nonetheless by the time the robot has finished step 6, it has already consumed approximately 80 vision frames or almost 3 seconds. During a game ball holding less than 3 seconds does not incur removal penalty but it offers enough time for opponents to arrive and resists the attack, by blocking the robot’s view and even pushing. The most problematic aspect is the grabbing state in which the robot spends too much time in tightly grasping the ball. In step 2 the need of changing to a ball grapping stance for active localisation also decreases the desirable swiftness. As a result it is not used in the current forward strategy.

Future Improvement

During the competition, the UTS team was capable to demonstrate how the robot could move fast (side walk, forward and turn) with the ball underneath its chin, while at the same time it could still see the goal. This would be a vital step in the current spin dribble because the robot could omit the active localisation and the incurred overhead before the spin (i.e. steps 2 to 4). It means that once the robot has grabbed the ball it only needs a rough heading estimate to determine which direction to turn and then it can visually line up the goal and force the ball forward.

In addition being able to view the front can let the robot not only line up with the visual goal more accurately regardless of the heading estimate, but also capable of avoiding opponents and even catching glimpses of beacons.

Chapter 7

Variable Turn Kick

The variable turn kick this year has incorporated the Upenn kick. While the turn kick propels the ball at 45^o, 90^o or 180^o, the Upenn kick itself usually propels the ball at 60^o. Unlike last year where the robot will walk up to the ball, grab it and turn kick, on this new robot it will grab the ball, turn and do the Upenn kick. So we can adjust the kick angle by changing the amount of turn needed.

The steps in executing the turn kick by ball grabbing are as follows:

1. The robot grabs the ball as fore mentioned and holds it for a sufficient number of frames to be certain that the ball will not roll out. It also finds out at what global heading it should cease turning and does the Upenn kick.
2. The robot starts turning with the ball underneath its chin.
3. When the robot gps heading matches the pre-computed one, it forces the gait to complete and does the Upenn kick.

Conclusion and Future Improvement

Rather than Upenn kick, the one handed kick was also tried in the turn kick. Nonetheless it was found to be slow since it took considerable time to both set up and recover from the kick. While the Upenn kicks are not very consistent in shooting in terms of angle accuracy, it shoots the ball every time if the ball is grabbed. On the other hand for the one handed kick, sometimes the ball could get stuck by its chin and the paws yet it is shoots the ball in a more consistent range. But the one handed kick was not used in competition due to speed (close to ball holding time limit) and reliability (if the ball is stuck when it shoots, it is very likely to be called for ball holding).

Dive kick was attempted as well, but once the robot turned to a desired heading the robot has to release the ball and stand up before the dive kick is executed. This is risky and time consuming.

As mentioned before a necessary improvement is to develop a ball grabbing that is fast to possess the ball and allows the robot to move quickly with the ball under its chin and still be able to visually see the front.

Chapter 8

Get Behind Ball

The get behind ball is one type of approach that aims to play defensively and safely. As the name suggests, its action is to circle around the ball in side walk fashion with the robot's body always facing the ball. Inside the attacker its main aim it to approach the ball without accidentally knocking it towards the robot's own goal, for example, the robot is running downfield to possess the ball. It is also used in the back off strategy for two robots to beak the symmetry.

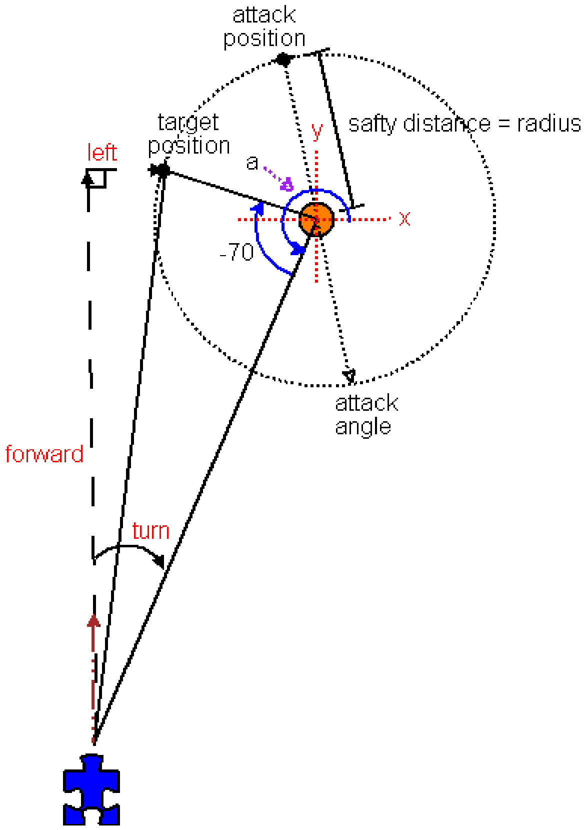

Fig 8.1: Get behind ball. The attack position is behind the ball in the attack angle. The figure shows the target is the circling position, which is 70° offset from the angle a, the absolute angle from the ball to the robot. Since the robot needs to face the ball constantly, the turn is proportional to the difference between robot heading and heading to ball.

The details (see Fig 8.1) of how the get behind ball works as follows:

1. Given the attack angle, direction of turn and the safety distance (or radius), it finds the attack position which is the point lined up with the attack angle behind the ball by the length of the safety distance. This is static relative to the ball.
2. The circling position, which is dynamic, is 70° offset from the angle originated from the ball to the robot. As the robot is moving, the circling position is also moving towards and eventually past the attack position.
3. If the distance to the circling position is shorter than the distance to the attack position, which is the case when the robot is initially getting behind the ball, the

robot will move to the circling position.

4. On the other hand if the distance to the attack position is shorter than the distance to the circling position, which is the case in the final phase of the get behind ball, the robot will move to the attack position.
5. Once the point that the robot moves to is found (either the attack or circling position), which is called the target, we can assign its horizontal component as the left vector, its vertical component as the forward vector and the turn is proportional to the absolute heading to ball relative to the robot. So:

   Forward = distance to target * cos (heading to target)
   Left = distance to target * sin (heading to target)
   Turn = heading to the ball * constant

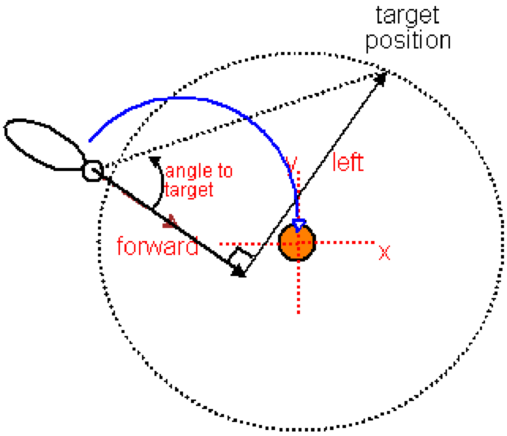

Fig 8.2:

The robot is rotated, and since the angle to target is between 0° and 90° the forward vector is positive, so the robot keeps walking closer to the ball and the result is a spiral path, as shown by the blue path.

However as the robot is rotated and is below the target position the vertical component is still positive, which means the forward is positive, the robot walks up to the ball and results in a spiral path instead of a circular locus (see Fig 8.2). The robot is drawing closer and closer to the ball where it should keep a constant safety distance from the ball. Depending when the get behind ball is used, if the attacker uses it to get around the ball, it is beneficial because it provides maximum protection. But it causes a problem for a back off player because it effectively does not back off.

In previous years for the back off forward the forward and left vectors are hard-coded with constant values to avoid this approach. One way to stop the forward being positive is to multiply the heading to target by some constant larger than 1, so that the heading to the target is larger than 90^{o}. Thus the forward vector is either small or negative. This does indeed allow a circular locus for get behind ball. This “tweak” only applies to the forward parameter, the other two parameters remain the same calculation and by adjusting this multiplication constant the robot can vary between spiral and circular path.

Chapter 9

Directional Paw Kick

The directional paw kick, also known as run behind ball, is developed for a continuous dribbling of the ball using a series of short range paw kicks. The motivation of using a series of paw kicks is that ideally the robot has more control of the ball while it can still drive the ball up field quickly, rather than hitting the ball up field but leave no other teammates being present around the top of the field to follow the attack and leaving opponents to regain the ball. To allow the paw kick to be directional, the robot has to line up in the desired direction before it executes every paw kick.

The idea of the directional paw kick is originated from the effective strategy developed in last year, the bird, which is a defensive move to intercept any ball on the wrong side of the field relative to all teammates' positions. For the details of bird please consult [2]. In fact, the directional paw kick is designed generically enough that it can be used as a general approach tactic since it only specifies a locus about how the robot approaches to the ball.

The skill essentially consists of two components, a vector generator and a PID controller. The vector generator takes the position of the ball, the heading and position of the robot, the angle that the robot wants to line up with (this is the DKD) and other auxiliary set up parameters (tangential offset, radial offset, radius) and returns the vector the robot should follow. The vector field is generated according to two circles situated next to the leaving point (see Fig 9.1). The PID controller will accept the computed vector and the PID gains and returns the corrected turning for robot.

Vector generator

The steps in finding a vector, which is simply a global heading, is explained in the following:

A. Given the current ball position, desired attack angle, offsets and radius, the coordinates of the leaving point is found.
B. Depending on which side the robot lies on relative to the tangential offset, it determines the direction (clockwise and anticlockwise) that the robot should travel.

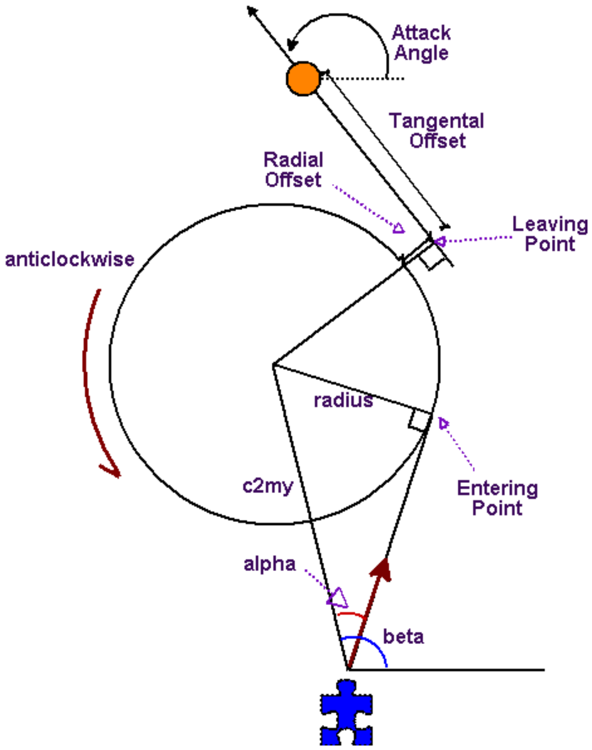

Fig 9.1:
The robot is outside the circle. Positive radial offset means the leaving point is outside of the circle, and negative means inside the circle. Positive tangential offset means the leaving point is behind the ball in the direction of the attack angle, negative means in front.

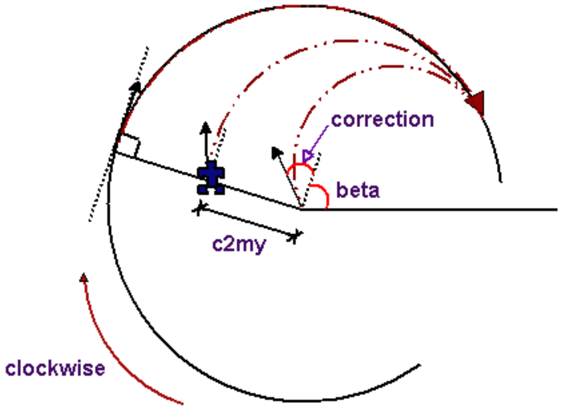

Fig 9.2:
The robot lies inside of the circle. In this case correction must be added so the robot path converges to the circle.

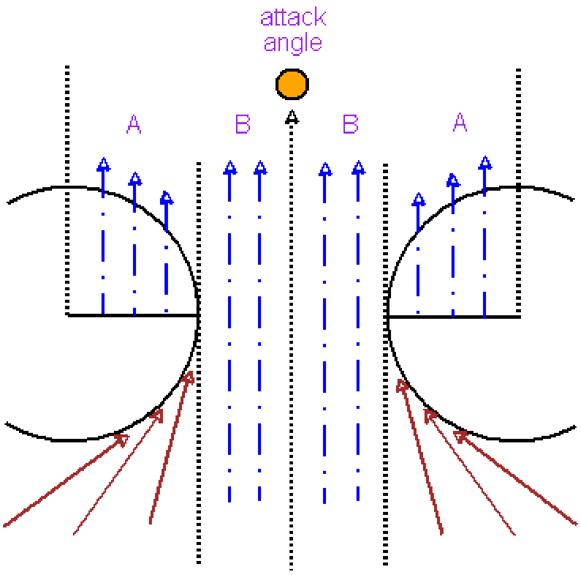

Fig 9.3:
Blue arrows show necessary modifications to prevent undesirable behaviour. Otherwise in regions A the robot will keep going around the circle. In regions B, the robot will zigzag because it will cross the boundary and turn left and right intermittently.

From the direction it deduces the centre of the circle and the entering point.

C. The vector is then determined by the direction of the tangent originated from the robot and directed to the circle.

D. However if the robot is inside the circular locus (see Fig 9.2), then it will add a corrective heading proportional to the radial distance from the circumference to the centre. Hence the correction is:

correction heading = 45^{o} * (radius – c2my) / radius,

and it is added to the tangent and the result is the required vector.

E. But on certain regions the vector should head the same as the attack angle such that the robot will not keep running in a circle (region A) or the robot will trace a zigzag path (region B) due to oscillation (see Fig 9.3).

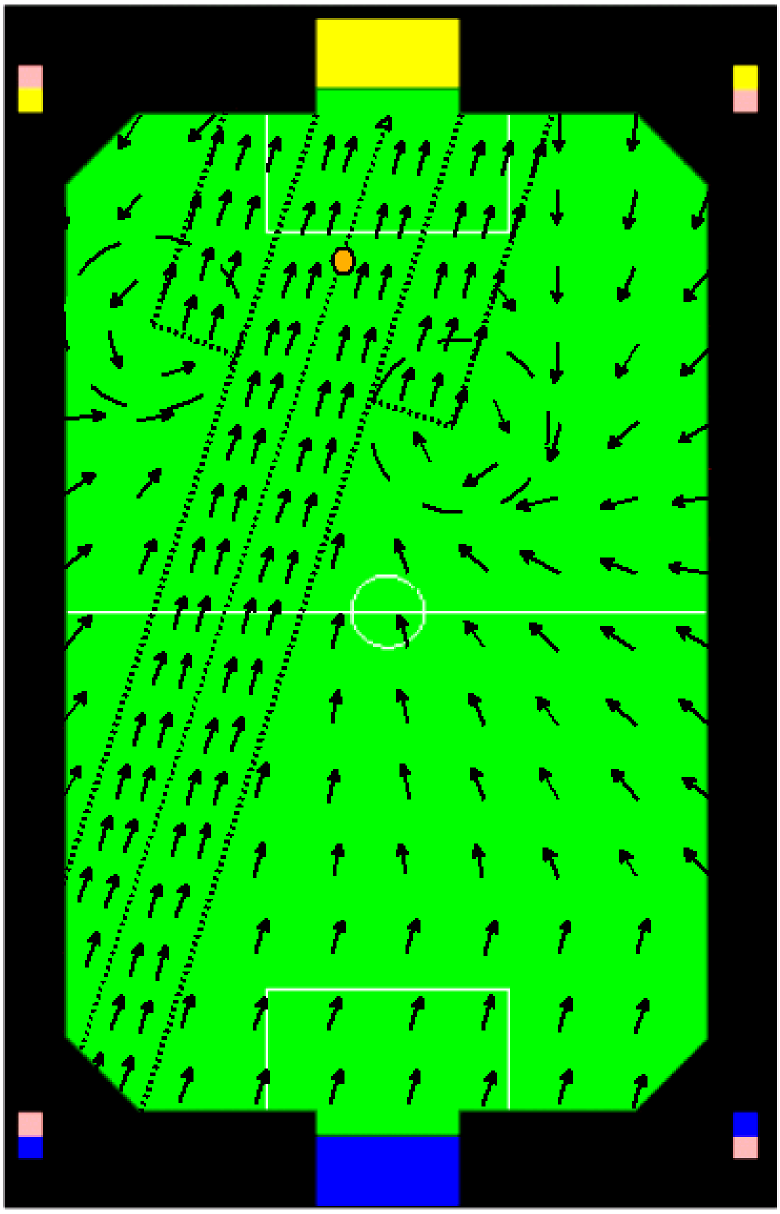

Fig 9.4: The resultant vector field

PID controller

The PID controller is responsible to maintain the robot to travel on a more stable path. The equation for the PID controller is:

adjusted turn of this frame =
Kp * unadjusted turn of this frame +
Kd * (unadjusted turn of this frame – adjusted turn of the last frame) +
Ki * running sum of adjusted turn

where the unadjusted turn simply equals the heading of the vector found by the vector generator minus the current robot's heading.

Conclusion

Initially the directional paw kick was much simpler but continuous improvement was carried out to refine the skill. Two aspects that need tuning effort are the PID gains and the radius, tangential, radial offsets combination. For the elliptical walk, only the P part is necessary because the effective turn is not very strong compared to the maximum forward speed. In contrast, the normal walk requires all P, I and D parts to control the robot. Although it is harder to tune the values when using the normal walk, it gives a better approach path (in terms of accuracy) than the path using the elliptical walk. But the elliptical walk is preferred because of its higher speed.

The offsets and radius are also carefully experimented to reduce the number of instances that the robot walks straight pass the ball, though this phenomenon could not be totally eliminated. This was once tackled by reversing the robot when it started to walk pass the ball, but it was inefficient in terms of speed.

Another interesting point is that the directional paw kick, as mentioned before, is quite generic. It is used as one type of approach technique, like get behind and hover to ball, and is be combined with other close ball contact skills such as paw kick and dive kicks.

Future Improvement

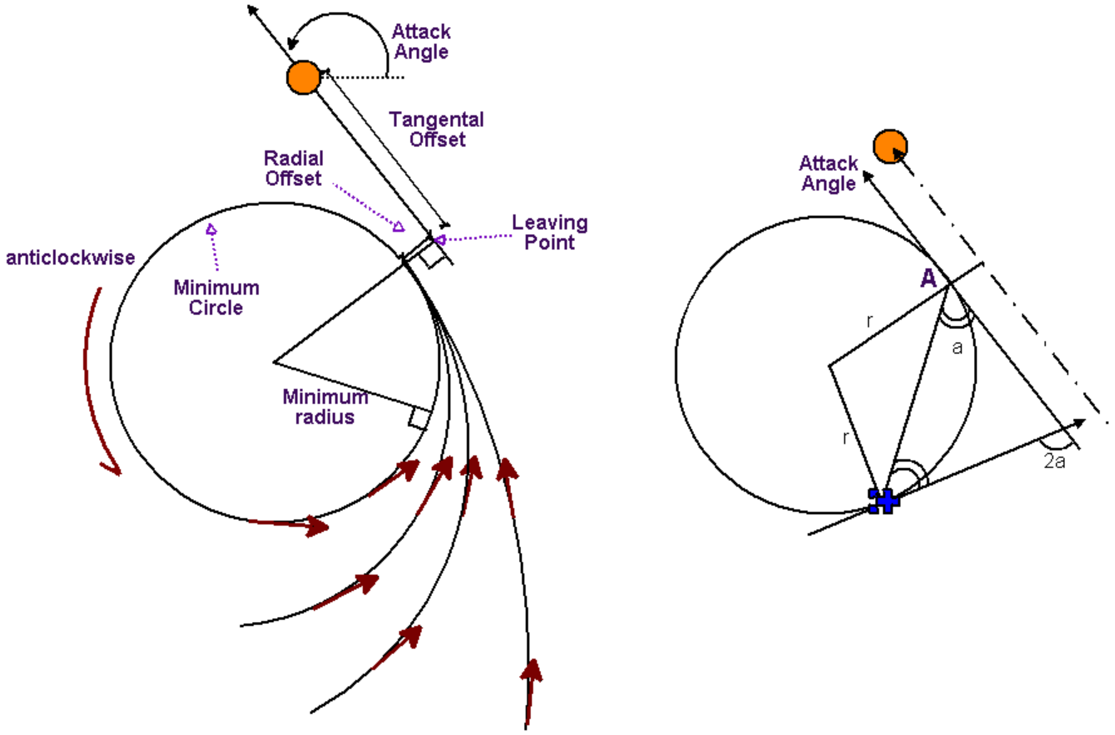

Fig 9.5:

(Left) Improvement model for the vector generation, by dynamically varying radius if it is above the minimum radius. This could line up the robot earlier.

(Right) The implementation is equally simple. The new vector is an offset of 2a from the attack angle, where a is the angle subtended by point A and where the robot is.

One experiment that should be tried in future is to change the vector field. A suggestion that is easy to implement is dynamic radius where a new circle is formed every frame such that the robot is always on the circumference on this new circle, and then the new vector is always tangential to this new circle, see Fig 9.5. When the robot is moving towards the ball, the circle will diminish in size in a gradual manner and this could line up the robot earlier than the current implementation. Once the circle is smaller than the predefined radius, it then switches back to the current implementation of the vector correction demonstrated in Fig 9.2 and 9.3.

Chapter 10

Locate Ball

The locate ball is a strategy that dictates what the robot must do once the robot loses sight of it. The locate ball routine consists of four phases; it proceeds to the next stage if the ball is still not found at the end of the present stage.

First Stage:

The robot stands still on the spot and the head makes two revolutions starting from straight below and in the direction that it last saw the ball. A head revolution means the head is circling in a rectangular loop. Since it is quite common the robot loses the ball when the ball is very close to the chin, looking straight down when first starting the routine will maximize the chance of finding it. But during the competition it was found that the robot had to take a whole revolution to find a ball situated under the chin. It was because it did not look down for the first two frames when the locate ball began, instead it looked up. But after the rectification this incorrect behaviour has not occurred again.

Second Stage:

In this stage the robot is spinning slowly on the spot while the head is making two revolutions in the same direction as the spin. The spin direction is determined on which side of the field the robot is on such that any contact with a nearby ball will more likely hit the ball towards the target goal. So if the robot is on the left it will turn clockwise and is if the ball is on the right side it will turn anticlockwise.

During the competition the second phase was changed. Rather than turning on the spot the robot simply reverses while the robot head does the same circling, and this turned out to be more effective. The reasons are as follows:

1. In most instances of ball loss the ball is actually quite close to the left or right side of the robot, and the time it takes for the robot to reverse one-dog length is less than making a whole body spin. So the duration for the robot to catch sight of the ball is shorter for the reverse search than the turn search.
2. Whether the ball is on the right or left it makes no difference for the reverse search. But turning the wrong way for spin search can take a lot more time.

3. Unless the ball is at the back, which is not the usual case, it will not hit the ball in the wrong way, that is towards its own half.

Third Stage:

In this phase the robot makes a fast spin on the spot for one revolution and it keeps looking in the direction that it is turning. Because the head is turned on its side, so the robot head is pointing at the ball before the body does when the ball is suddenly in its view, and it does not spin past the ball but has enough time for the body to line up and the head returns to the normal tracking position.

Fourth Stage:

Finally, the robot will start the final phase and stays in that state until it can find the ball. In this phase the robot spins slowly towards a predefined point close to the centre of the field and circles its head. The reason it does not stay on the same spot is that it cannot see a ball too far away, so spinning to the half field area would be sensible. But in a game this is not likely to happen because its teammates will indicate the ball’s position through wireless communication so that the robot will walk to that wireless ball position while performing a standard head cycling search.

Chapter 11

Track Ball

To track the visual ball, the robot will project the centre of the top edge of the ball on the image plane onto a plane parallel to the ground but raised up to the height equal to the radius of the ball, the so called “ball plane”. Once the projected point is found, it is in terms of a three dimension coordinate relative to the robot’s origin. The robot’s origin is a point on the ground underneath the robot’s neck base. Finally, the projected point then forms the input to the head motion control.

But since there are two transformations (one form the image to a 3d coordinate, and from the 3d coordinate to the actual head angles) in the case of a close fireball, (especially when the robot is ready for a grab) the errors become large leading the occluded image of the ball occupies a minor lower portion of the image plane. This causes the recognized ball easily fall out of the robot’s image plane due to significant oscillation at the camera.

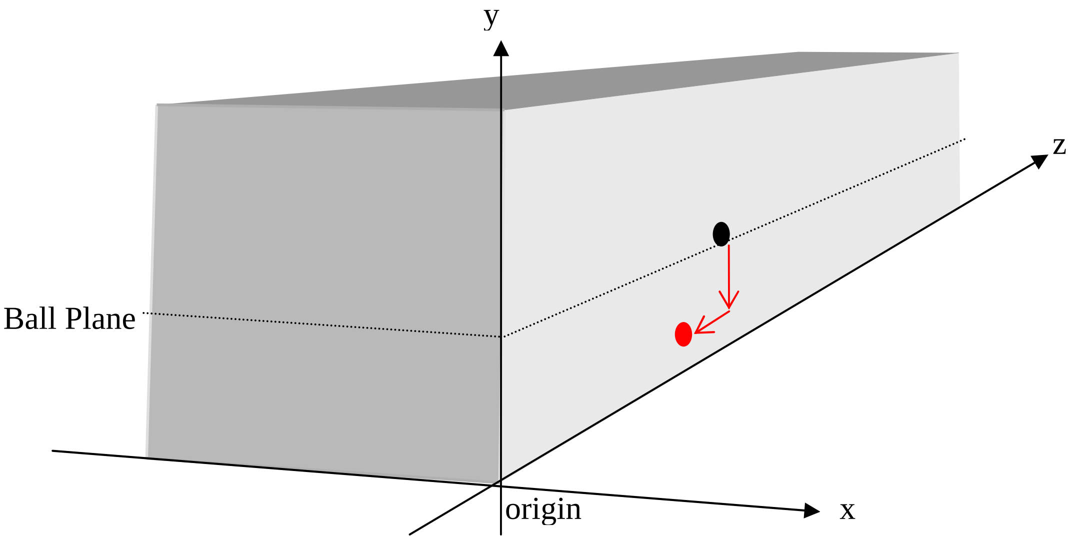

Fig 11.1: Adjustment for fast approaching ball (using velocity prediction) and close fireball. The black dot shows the original projected point. The red arrows are the corrections. The red dot shows the new projected point.

To solve this problem a simple adjustment is added to the projected point coordinate.

If the ball distance is less than a certain distance (25 cm), the projected coordinate would be closer to the robot and closer to the ground. The projected x is also set to zero when the ball is close in front such that the head is forced to look straight, otherwise the head will swing in oscillation.

Another problem that arises with the ball tracking is that when the ball is approaching fast relative to the robot (for example the robot is walking with maximum speed towards the ball), it could lose the ball quite easily. As a result, the tracking makes a small adjustment which is linearly interpolated by the ball velocity relative to the robot. This effectively points the robot head to track the next ball position.

Section 2
Strategy

In the rUNSWift team, a major consideration in the behaviour level of the robot is the team interactions. There are two main levels of team interaction. The first is a local level which defines the strategy and positioning between the two forwards that are attacking after the ball. The other is a global level which considers the two attacking forwards as one single tier and the third robot as a striker across the field (see figure below). Therefore our strategy is to always allow two forwards closely charging the ball and the striker is staying further away ready to attack. The strategy does not have any ball passing; each robot determines whether it should charge in or not based on its own position, its teammates location, the ball position and the teammate wireless broadcast information.

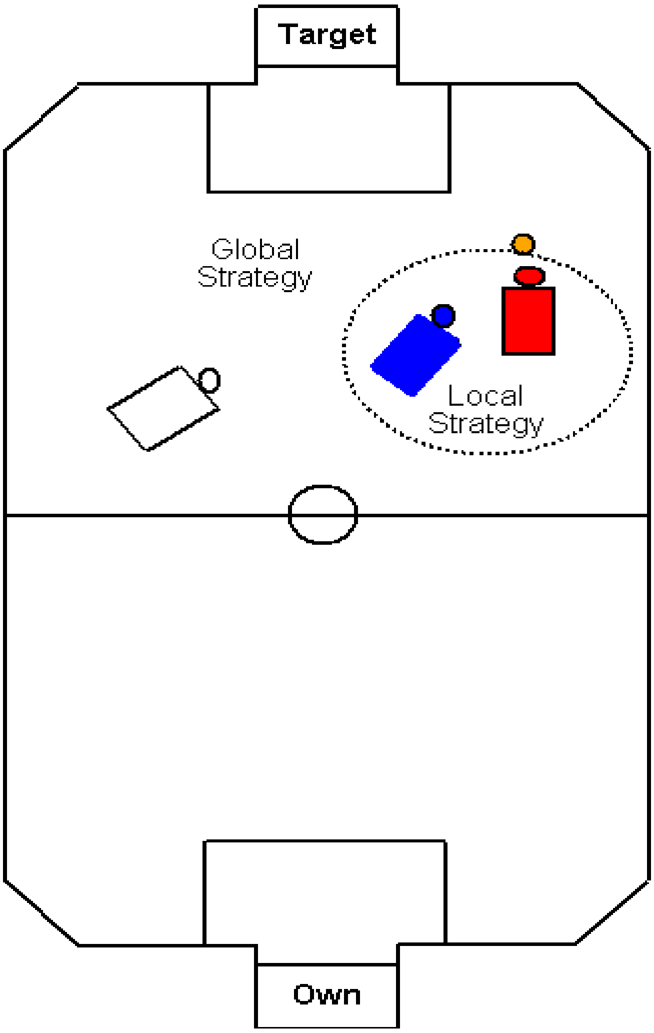

Fig: Two levels of strategies exist. The red is the attacker. The blue is the supporter. The white is the striker.
The global tactic considers the interaction of the striker and the two attacking forwards acts as a single entity. The local one considers the interactions of the attacker and the close supporter. Their roles often switch in a game.

Although the strategy is aggressive, there exists no explicit defender permanently located in the lower half of the field to protect its own goal. To compensate this disadvantage the bird was devised in last year strategy to take the role of a rapid defender which runs to intercept the ball.

The following subsections will describe the strategy of the three forwards. For the goal keeper and bird strategies, please consult [2, 13].

The forward strategy

The forward strategy defines what role is assigned for a general forward player. The figure below is the general description of the decision flow.

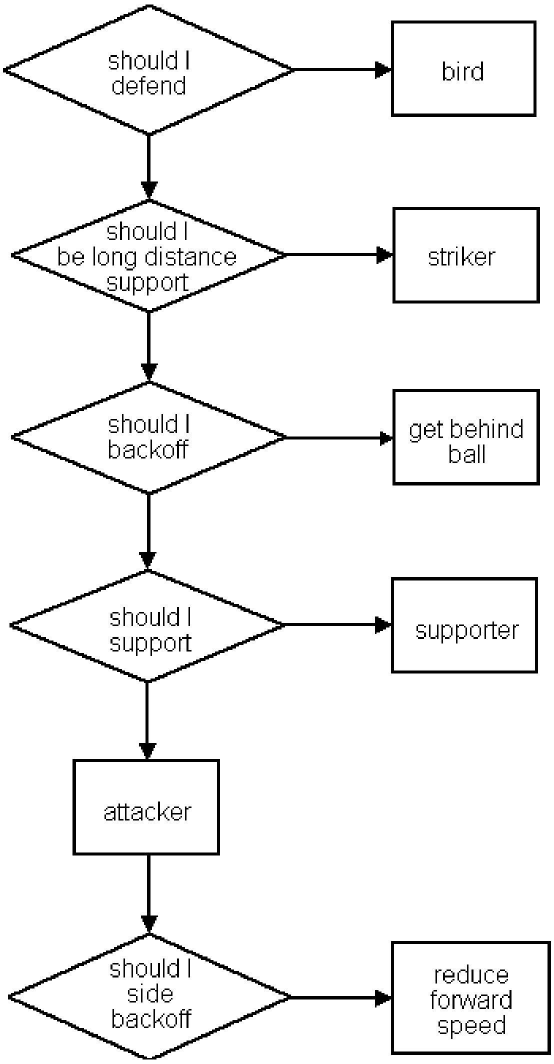

Fig: The decision flow of a forward player.

Chapter 12

Main Attacker Strategy

The following sections are details on the implementation of the main attacker strategy. All sections, beside 12.1, are shown in the logical order of the decision tree in which the attacker follows precisely.

12.1 Hysteresis and Lockmode

During the Python re-development, the implementation of the so called lockmode was abandoned. Lockmode was an idea to model a state in the behaviour where the robot tries to perform the same set of actions over a period of time until it decides when the action is finished, for instance ball grabbing. This type of action is called an atomic action. Symbolically the robot tries to go to the same node of the decision tree over a number of consecutive frames. The idea of a lockmode can be visualized through the following diagram.

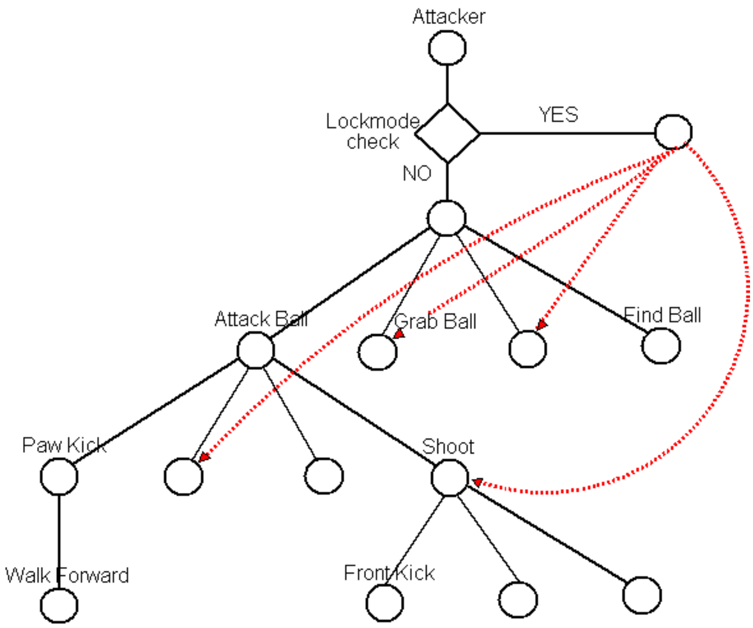

Fig 12.1:
Lockmode visualization.
It bypasses other parts of the tree and repeats the same node until the action is ended.

As illustrated from the diagram, the first stage in the decision that the attacker makes is whether it has been involved in a lockmode. If so the next decision will go to the specified node and it will perform the last atomic action. If the robot is not “locked” into any action, then it will continue to traverse the decision tree. Although this can suffice the purpose of modeling a state it is not flexible in a sense that the lockmode is always the first priority and no other actions are allowed besides the atomic action.

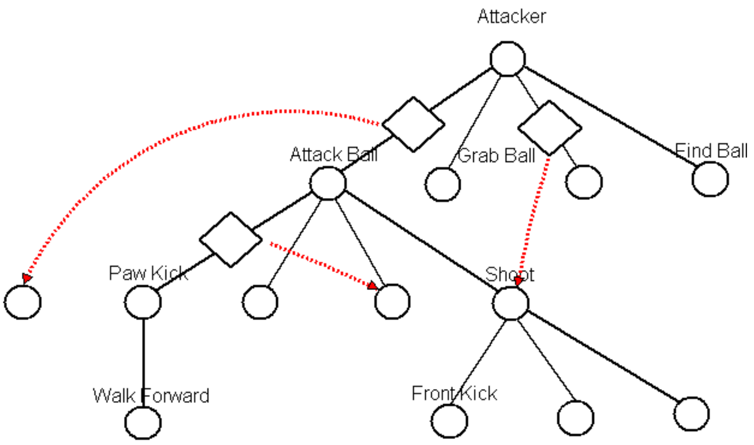

Fig 12.2: Hysteresis checks are located along the tree to replace the lockmode implementation.

Thus rather than forcing a lockmode check in the very top of the tree, the current strategy is to arrange the hysteresis checks at the appropriate places along the decision tree such that it is more flexible and dynamic. The tradeoff is that careful management of these hysteresis checks is necessary. Another way to manage state is to use state machine, in which the programmer defines actions for each state and conditions for state transitions. For details of behavioural implementation using state machine, please consult [4, 10, and 11].

12.2 Decision on Ball Tracking, Active Localisation, Scanning Localisation and Ball Search

Before the attacker decides how to approach and attack the ball, the most important decision is that it must draw up a plan of when it should track the ball, perform active and stationary localisation and search the ball. A wrong judgment could render the attack futile. This decision is explained in the following.

1. The robot will first check whether it should do a **scanning localisation**, in which the head keeps panning from left to right and vice versa for a number of frames such that it can see bacons and field lines. The robot keeps walking towards the

ball using any available ball source (wireless or gps), which is better than standing still. Though this kind of localisation is slow, this is given the highest priority because possessing a ball without any idea of position or direction can be very dangerous. Hence the conditions are very carefully defined as follows.

- The robot is very lost about its own direction or position.
- It has not seen an opponent nearby recently, told by gps opponent tracking.
- It can clearly see the ball.
- It is not doing active localisation at the moment nor locating the ball.

After the scanning localisation, the robot cannot perform active localisation for the next 3 seconds.

2. The robot then checks whether it should **locate the wireless ball**, in which it will cycle its head and walks towards the wireless ball position. It is triggered when:
   - The visual ball has been lost for at least 14 frames.
   - The distance to the wireless ball is more than 80 cm, because for far away ball the robot is not able to see it during the locate ball routine.
   - It is not active localizing.

3. The robot then checks whether it should perform **locate ball** as described in chapter 10 when it has lost the ball for more than 14 frames. The condition are:
   - The visual ball bas been lost for 14 frames.
   - It is not active localizing.

4. The robot then checks whether it should perform **active localisation**, in which the head will look in the direction of the closest beacon reachable by its head while it will let the behaviour to decide what the body should do. The conditions are:
   - It can see the visual ball and it is more than 30 cm away.
   - It has not seen any opponent very close by recently.
   - It is not sure of its position and heading.

   OR
   - It is doing get behind ball for more than 3 seconds.

   The reason why get behind ball needs active localisation is that when using the normal walk modified for faster side step, the odometry does not often reflect the correct motion update and its believed heading can be quite inaccurate.

5. Finally, if all the above is not satisfied, then the robot will simply track the ball if

the visual ball is available. If it is not available for the first 4 frames of ball loss, the robot will still keep its head in the last ball seen position since the ball is likely to re-appear. For the next 9 frames if the gps ball is in front of the robot it will track the gps ball since it contains velocity information, otherwise it will keep the head in the last ball seen position. More frames of ball loss will fall into cases 2 or 3 above, that is searching for the ball.

A summary of these decisions is visualized from the following flow chart. Note that for scanning localize, locating wireless ball and locating visual ball, they are the end of the behavioral decision because they have already controlled both the body and head. The remaining decisions only control the head so the decision tree traversal is still continued.

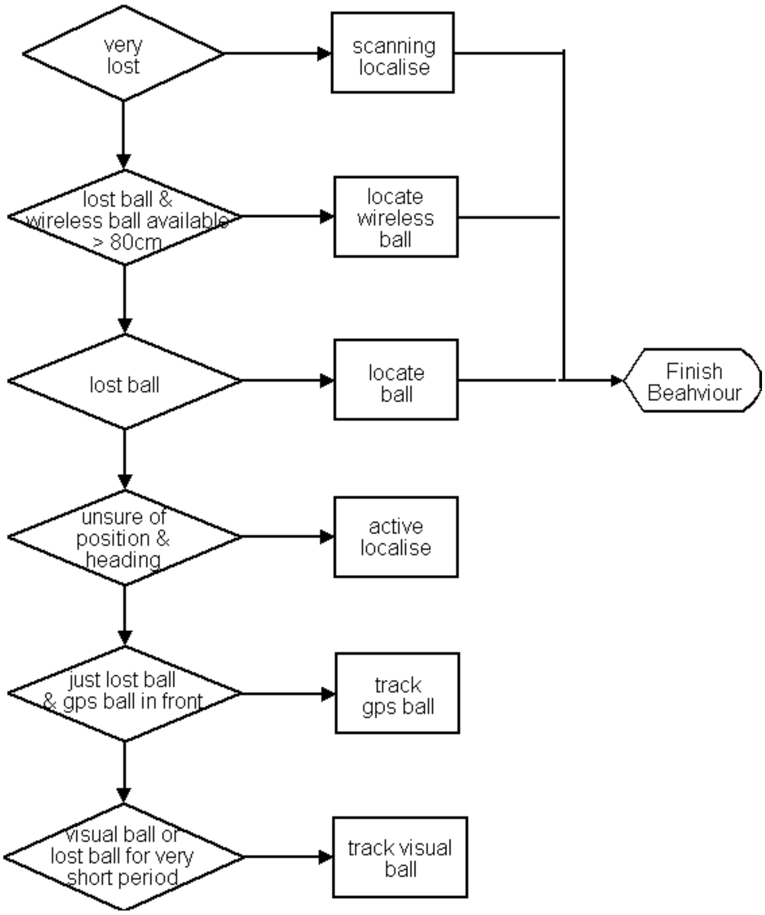
very lost
scanning localise
lost ball & wireless ball available > 80cm
locate wireless ball
lost ball
locate ball
Finish Beahviour
unsure of position & heading
active localise
just lost ball & gps ball in front
track gps ball
visual ball or lost ball for very short period
track visual ball

Fig 12.3: Fundamental decisions of a robot before it attacks the ball. This is the head control algorithm. Note this is only the general visualization of the code because it does not show all detailed conditions.

12.3 The Approach Strategy

The four main approaches that the attacker uses are:

- Edge paw kick
- Get behind ball
- Directional paw kick
- Hover to ball

12.3.1 Edge Paw Kick

Edge paw kick is using the straight paw kick on all the edges of the field. Although it cannot provide any control of direction of where the ball will head, on the edges the ball is usually best and securely driven along the field edge at maximum speed. It is fast and does not involve any set up or recovery time, such as in dive and Upenn kicks. Moreover using paw kick will allow the robot to kick the ball without the paw closer to the edge being caught by the edge, which is the case if the robot hovers to the ball along the edge. There are different cases of edge paw kicks explained in the following paragraphs.

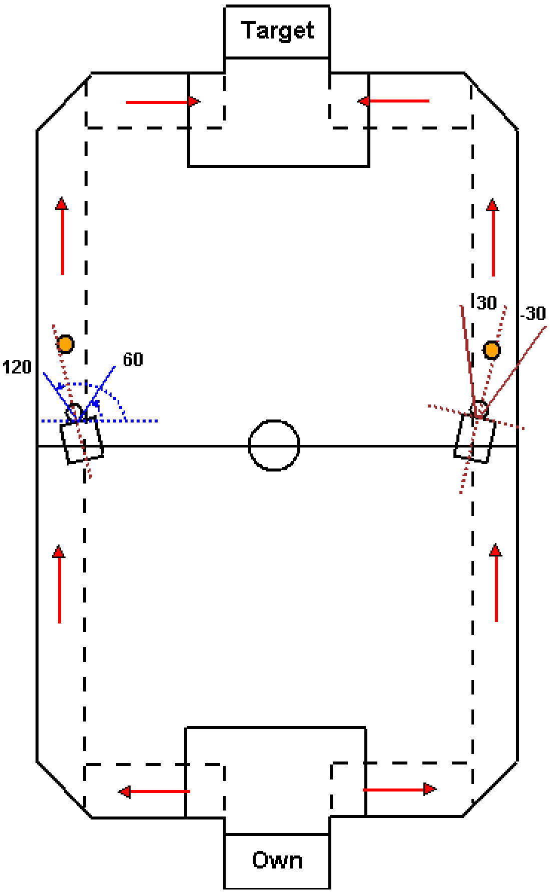

Fig 12.4:
Paw kick along the side edges, together with the triggering conditions. Top and bottom edges have similar conditions as well.
The dashed lines show the border region where paw kicks are used. The red arrows show the direction of the paw kick.

A. Side Edges.

Paw kick on side edges can drive the ball up field effectively. It is triggered when:

- The ball and the robot are on right or left edge.
- The robot global heading is between 60^o and 120^o, that is, makes sure the robot is facing up field.
- The ball must be within 30^o of the robot's own heading, or in other words, the ball is directly in front of the robot reachable by its paw.

B. Top Edges.

Paw kick on top edges can drive the ball past the goalie. It is triggered when:

- The ball and the robot are on the top left or top right edge.
- The ball must be within 30^o of the robot's own heading.
- If the robot and ball is on the top right edge, then the robot's global heading is between 150^o and 210^o, that is, makes sure the robot is facing left towards the target goal.
- Similarly for the left edge, the robot's global heading is between -30^o and 30^o, that is, is facing right towards the target goal.

C. Bottom Edges.

The aim is to use paw on bottom edge is to fast defend, that is, to clear the ball safely. It is triggered when:

- The ball and the robot are on the bottom left or right edge.
- The ball must be within 30^o of the robot's own heading.
- If the robot and ball is on the bottom right edge, then the robot's global heading is between -30^o and 30^o, that is, makes sure the robot is facing right away from its own goal.
- Similarly for the left edge, the robot's global heading is between 150^o and 210^o, that is, is facing left away from its own goal.

12.3.2 Get Behind Ball

Get behind ball is used for the robot to defensively retrieve a ball when it is not within the correct heading, such that it would not accidentally knock the ball towards the undesired direction. And when opponents in scrums are attacking the ball, it is better for the robot to get behind the ball to physically block the attack. One major disadvantage is that the get behind ball routine is slow. It is used in several occasions as demonstrated in the following.

A. Side Edges.

Along the side edges, the robot will attempt to line up to the global heading of 90°, which is exactly facing up field. It is called when:

- When the ball is on right or left side edge.
- The robot global heading is between 195° and 345°, that is, makes sure the robot is facing down field.

B. Top Edges

When the robot is heading away front the target goal, it needs to get behind the ball to start the attack. It finishes the routine when the robot is facing the top edges rather than fully towards the target goal, because in the latter case the opponent goalie could easily clear the ball away. It is called when:

- If the ball is on the top right edge, the robot global heading is between -30° and 80°, that is, makes sure the robot is facing right. Then it will try to line up at a heading of 100°.
- Similarly if the ball is on the top left edge, the robot global heading is between 100° and 210° (the robot is facing left). Then it will try to line up at a heading of 80°.

C. Top Corners

It is called when:

- If the ball is on the top right corner, the robot global heading is between -30° and 70°, that is, makes sure the robot is facing right. Then it will try to line up at a heading of 100°.
- Similarly if the ball is on the top left corner, the robot global heading is between 110° and 210° (the robot is facing right). Then it will try to line up at a heading of 80°.

D. Defensive half of the field

If none of the above are satisfied, the final case is when the get behind ball is used for the defensive half of the field, it will try to line up for the Upenn kick. The get behind ball is called when the ball is in its own half and the robot global heading is between 200° and 340°, that is, makes sure the robot is facing down.

For all the get behind ball cases, the direction in which the robot will get behind depends on which side of the field the ball lies on. If it is on the left side of the field, the robot gets behind in clockwise direction, and if it is on the right side the robot get behind in anti-clockwise direction. This is desirable because the robot acts as an obstacle in between the ball and its own goal and it is also the only way to get behind ball along the field borders.

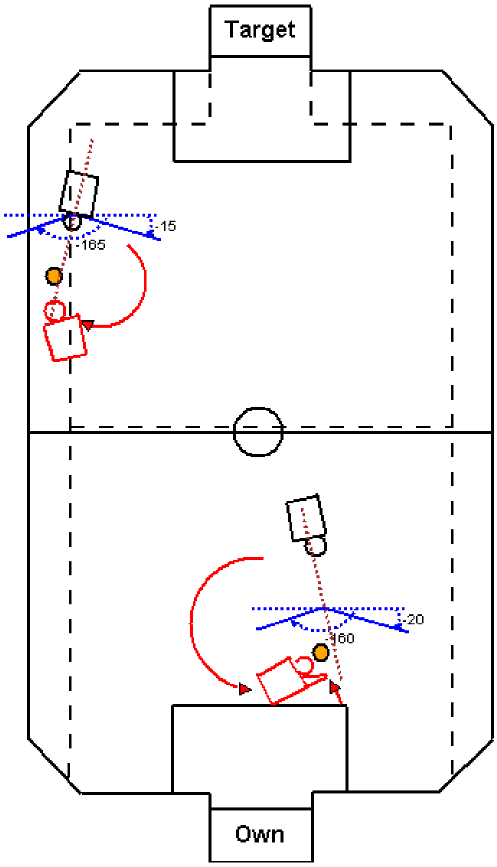

Fig 12.5:
Get behind ball is used on the border regions and the defensive half of the field. Note that for the defensive half, an Upenn kick is executed to propel the ball up the field to quickly save the ball. Upenn is not executed along the border.

12.3.3 Directional Paw Kick

As discussed in chapter 6, the aim of the directional paw kick is to allow continuous, fast yet safe dribbling of the ball towards the target goal such that the ball is close to and in front of the attacker most of the time. To powerfully hit the ball and leave no other teammates to follow the attack is not beneficial for the attack.

The directional paw kick is a type of approach triggered by the following conditions:

- The robot is in the heading of 0^o and 180^o, that is the robot is not heading down field.
- Both the ball and the robot are not along any edges.
- The ball is within 70^o of the robot heading.
- The robot heading is within 60^o of the DKD heading.

When the conditions that indicates the robot has indeed lined up it then uses the straight paw kick to actually fulfill the task. The conditions for triggering the final straight paw kick are:

- the robot's heading is within 20° of the DKD heading, or
- the robot is in the upper 60% of the field, and it is within the narrow DKD range.

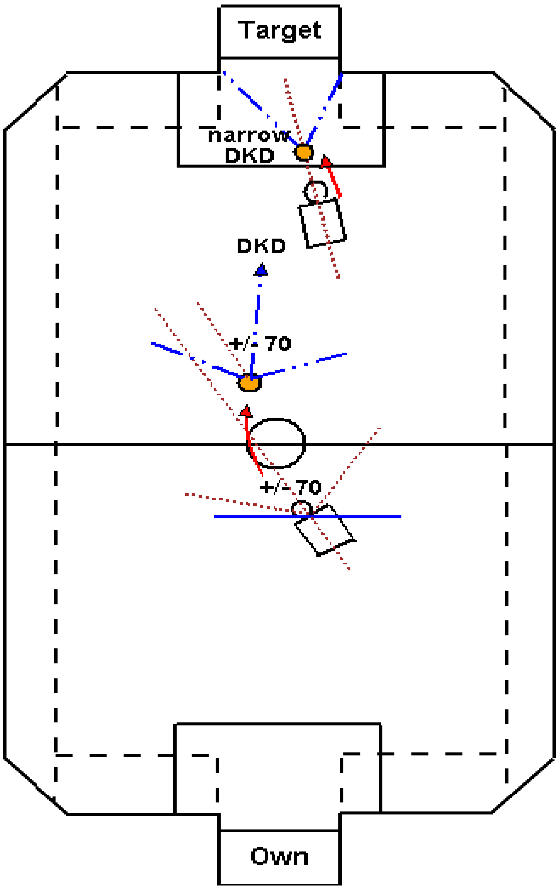

Fig 12.6:
Directional paw kick. The conditions for triggering it makes sure the robot can line up for DKD. Note also that once the robot is almost lined up or its heading inside the narrow DKD range when close to the target goal, it will execute a straight paw kick.

12.3.4 Hover To Ball

When none of the above approach strategy is chosen, the main attacker hovers towards the ball by default, which is simply approaching the ball in maximum speed directly to the ball. It uses stealth dog to avoid any opponents on its path. When hovering to the ball, the robot intends to slow down and shoots the ball. This will be described in the next section. [2, 13] describe the stealth dog in details.

12.4 The Shoot Strategy

One difference in the shoot strategy in this year is that the robot does not choose a kick once it possesses the ball but based on the information about its environment, the robot decides both the approach and shoot strategies. In other words the robot plans its shoot strategy in its approach, thus the separating line between the shoot and approach strategies becomes less distinct. The major reason is that depending on which kick it is using, the robot may not need to grab the ball to execute a kick. There are a number of selections for what kicks are used:

- Edge Turn Kick
- Variable Turn Kick
- Upenn kick
- Dive kicks

12.4.1 Edge Turn Kick

Edge turn kick is simply a modified normal walk with its front paws brought forward to hit the ball sideways. It is not a particular powerful and accurate kick as last year turn kick, however, when the robot is facing the edges with the ball in between, an Upenn kick is not going to help since the robot arm can hit the edges. Hitting edges with the arm cannot propel the ball at all and even worse damage the robot. So the

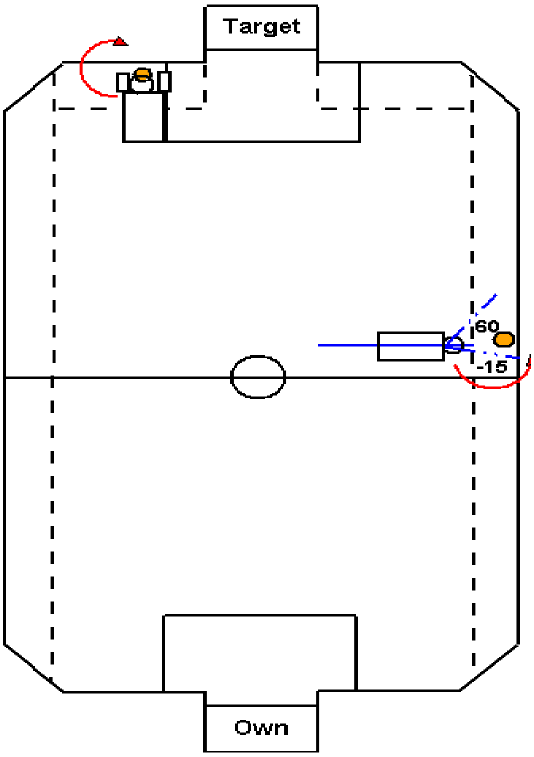

Fig 12.7 Edge turn kick

When the ball is along the side and top edges and the robot aligned in heading that is not suitable for paw kick nor get behind, then it executes edge turn kick.

edge spin kick is used to nudge the ball sideway. After the execution of spin kick, the robot can then follow the attack with the edge paw kicks.

When the edge turn kick is going to be used, the robot decelerates upon approaching the ball. The conditions for the edge spin kicks are as follows:

A. Side Edges.

Along the side edges, the triggers are:

- If the ball is on the right edge, the robot's heading is between -15° and 60°.
- Or if the ball is on the left edge, the robot's heading is between 120° and 195°.

B. Top Edges

It is called when:

- If the ball is on the top right edge, the robot heading is between 70° and 150°.
- Or if the ball is on the top left edge, the robot heading is between 30° and 100°.

Along the bottom edges, the edge spin kick is not as defensive as get behind ball and paw kick. So it is not used on the two bottom edges.

The problem of this edge turn kick is its front paws could be caught on the edges and the robot could not even turn especially when it is being pushed. One possible improvement, or replacement to be exact, for the edge turn kick is a head swipe, which is very strong and consistent as shown by a number of teams in the competition. The final game video is a good visual source of this head swipe and also [13] experiments with the head swipe.

12.4.2 Variable Turn Kick

Variable turn kicks, as mentioned in chapter 7, is implemented by first grabbing the ball, turn, and followed by an Upenn kick. Since it involves ball grabbing which could fumble the ball down the field, it is used conservatively, as defined by the following conditions:

- the ball is in the target half and not along any edges,
- the robot is facing down field, that is not in the heading from 20° to 160°.

The robot will then slow down, grab the ball and starts the turn kick routine.

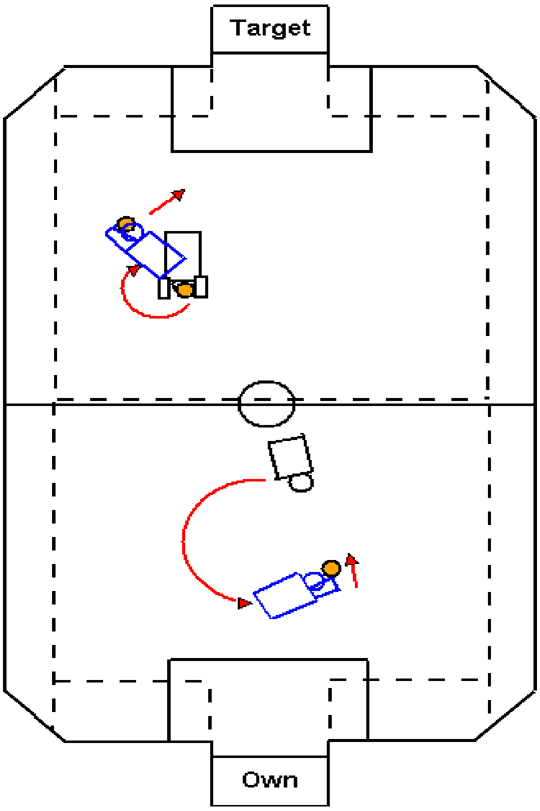

Fig 12.8 Variable turn kick and get behind ball turn kick. The difference is that the first one is used in the target half for fast attack and latter in the own half for conservative play.

12.4.3 Upenn Kick

As already mentioned in section 12.3.2, another way Upenn kick is used is that if the ball is in the own half and with similar conditions as the variable turn kick (the robot is heading down the field) then the robot gets behind the ball and does an Upenn kick rather than the variable turn kick which could fumble towards the own goal in a scrum situation, see Fig 12.8 above.

However there are times when none of the above decisions are chosen at all. When this is true by default the robot hovers to ball and executes an Upenn or dive kick (see later). The default Upenn kick is called when:

- the ball is close to and in front of the robot,
- the ball is not along the edges,
- if the robot’s heading is between -45^o and 45^o for a right Upenn kick and if the robot’s heading is between 135^o and 225^o for a left Upenn kick.

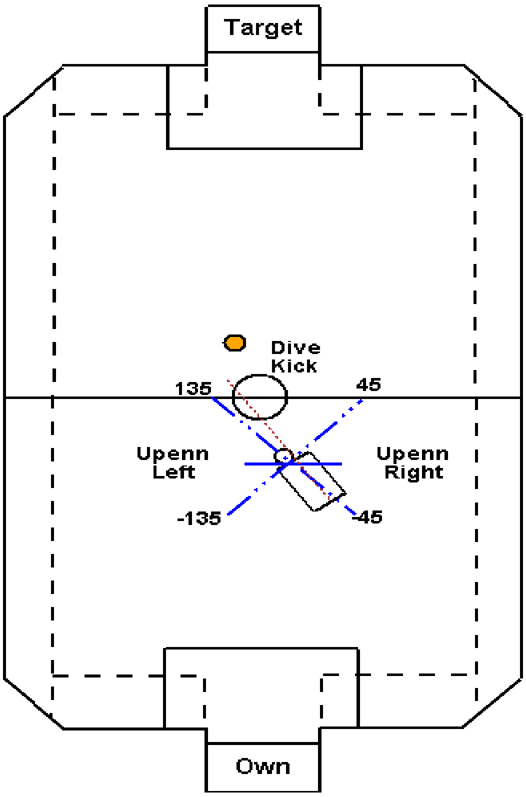

Fig 12.9 Default kicks

When none of the above decision criteria are satisfied the robot will does these default kicks when it is close in front of the ball.

For -45° to 45°, it is Upenn right.

For 45° to 135°, it is dive kick.

For 135° to 225°, it is Upenn left.

12.4.4 Dive kick

The aim of dive kick is to propel the ball forward fast and in precise direction, and it does achieve this effectively. The dive kick is used as a default action and when:

- the ball is close and in front of the robot,
- the ball is not along any edges,
- the robot's heading is between 45° and 135°,
- the robot is not in the top corners.

Then the robot then slows down, stops in front of the ball and dive kicks the ball forwards.

Chapter 13

Local Robot Interactions

Local robot interactions refer to the activities of the close attacker for backing off from and assisting the current main attacker. These decisions are primarily based on the information provided by the vision and gps systems as they are the most up to date information about the current environment. The attacking robot makes decisions from the wireless team strategy information as well, though to a less extensive degree since the information source can be intermittent due to inconsistent wireless connectivity and the delay is significant. There are five specific results from the local level tactic, which are:

1. to immediate back off because a visual teammate is extremely close in front.
2. to back off from the main attacker and move to the supportive position,
3. to break the symmetry using get behind ball,
4. to side back off by slowing down,
5. to attack the ball as normal.

The details and discussions of these five types of local robot interactions will be expanded in the following sections.

13.1 Immediate Back Off

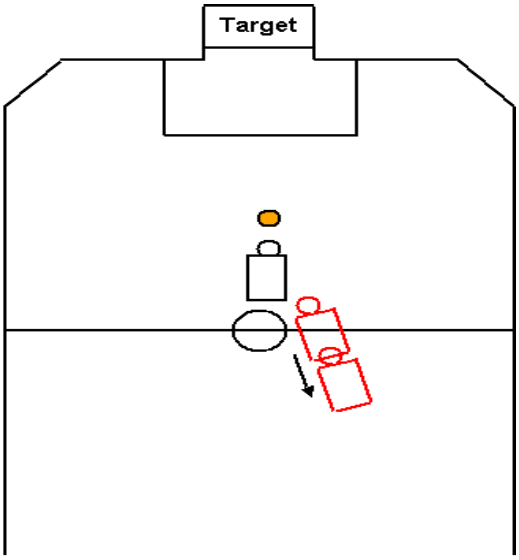

Fig 13.1
Immediate backoff is used in the case of extreme close teammate in front of the robot, it immediately takes several steps back to prevent any contacts.

When the robot detects that there is a visual teammate in front in a very short distance, the robot reverses instantly to avoid any unnecessary contacts with the teammate. This check is given the highest priority; the reason is that these contacts could quickly build up into a scrum and affects adversely the effectiveness of the local strategy and the attack. So this is an emergency response that should not always be triggered.

During a game it turned out that it could indeed prevent a scrum buildup and there was no observations of two teammates simultaneously backing off from one another, because most of the times the close supporter was behind the main attacker who was constantly tracking the ball. Unless both robots are facing each other very close, in which case it is logically and tactically the best solution for both robots to back off from each other by reversing and clearing up the scrum.

13.2 DKD Related Tactics

The close interaction strategy heavily relies on the DKD which has been discussed in chapter 2 in the start of this report. Since the DKD is a ray originated from the ball, one could consider a polar coordinate system with the DKD as the reference angle and the ball as the origin. The motivation behind it is to examine the robots' positions relative to the DKD and ball such that the players can determine among themselves who would be the best attacker. The strategy defines the attacker at the best attacking position is always the one with the highest magnitude of the angular offset (assume the coordinate system is from 180^o to -180^o) from the DKD, because intuitively it is lined up the best and hence the most ready to attack.

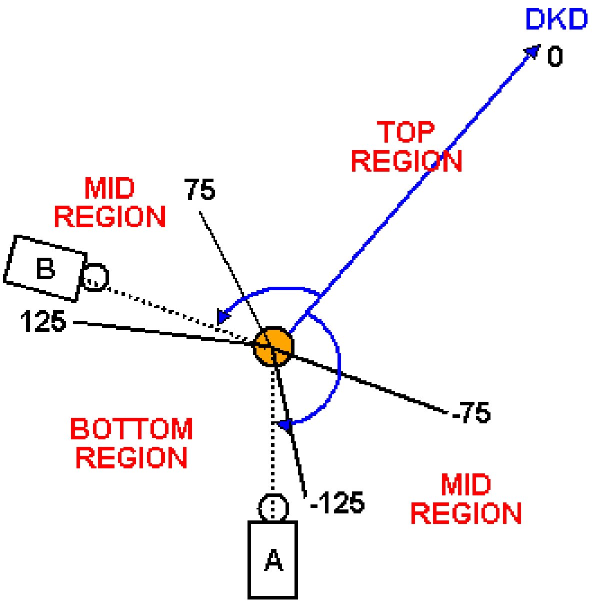

Fig 13.2

The principle of DKD related backoff. The surrounding space around the ball is divided into 3 types of regions. The strategy then decides who is the backoff player or attacker based on where the robots are.

In this figure, A in the bottom region is the attacker and B in the mid region is backoff player, because A has the largest angular offset and best lined up.

Given the DKD coordinate system the robot then divides its surrounding space into different regions, as demonstrated in the above diagram.

Then the decisions are made by the following criteria in the descending order of precedence:

- If the robot is in the bottom region, but its teammate is not, then it would possibly consider being an attacker. The reason why it is only possible because in the local cooperation strategy, when the robot decides to be an attacker there is another additional check, which will be discussed soon, to make sure that its teammate is not attacking the ball as well.
- Conversely if the teammate is in the bottom region, but the robot itself is not, then it would become a supporter.

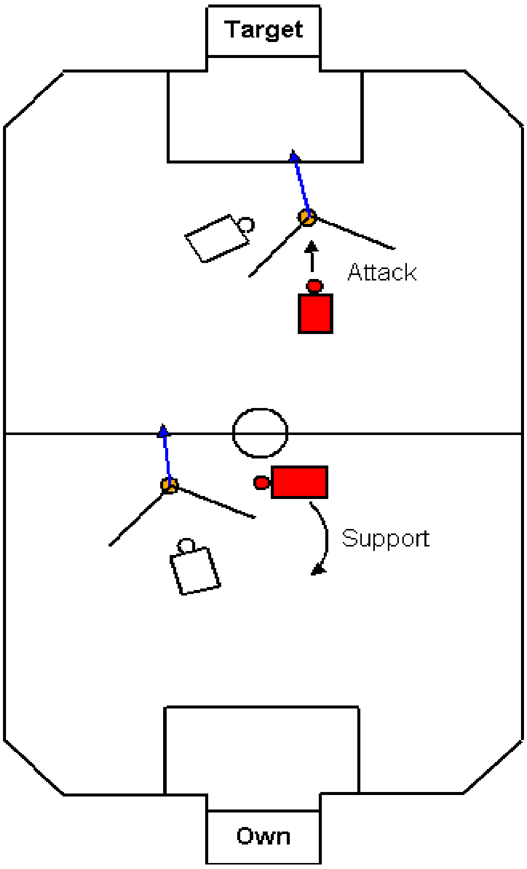

Fig 13.3
The upper pairs of robots represent the first point where the robot is in the bottom region but the teammate is not. Then it will attack (probably).

The lower pairs present the second point where the robot is not in the bottom region but its teammate is. In this case it will move to the supportive position.

- If both the robot and its teammate are in the bottom region, then:
  - If the robot itself can see the ball but its teammate cannot (from wireless broadcast), it is possibly an attacker. On the other hand, if the robot cannot see the ball but its teammate can, it becomes a supporter.
  - If the robot is closer to the ball than its teammate is, then it is a possible attacker. Otherwise if the teammate is closer then it is a supporter.

  - If the robot's angular offset is larger than its teammate's ones, then it is a possible attacker, otherwise it is a supporter.
  - If the robot's player number is smaller than its teammate's ones, then it is a possible attacker, otherwise it is a supporter.
- If none of the above cases is true, that is, none of the robots are in the bottom region it gets behind the ball to break the symmetry and to maintain the defense.

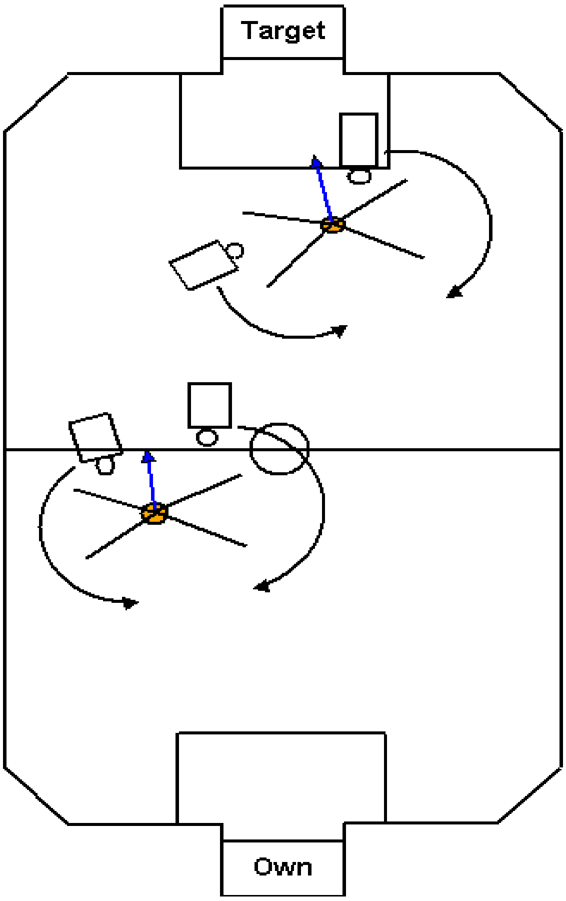

Fig 13.4 Symmetry breaking by getting behind the ball. When none of the robots are in the bottom region, they solve the ambiguity and break the symmetry.
The upper pair show one possible case where one robot is in the top region and one in the middle region. The lower pair show both robots are in the top region.

Possible Attacker

Note that for a possible attacker to become a real attacker, the robot must check whether a wireless message from its teammate is available. If it is available and

- the robot finds that its teammate in the bottom region,
- its teammate broadcasts that it is very close to the visual ball (less than 18 cm), and
- its teammate broadcasts that it is not backing off,

then the robot will become a supporter rather than an attacker. Otherwise it will still continue its attack. This sanity check serves as a final warning sign to make certain that the up coming attack would not cause any contacts with the teammate. No such test exists for the other forms of local interaction.

When the robot determines to be a supporter it moves to the supportive position during which it is constantly facing the ball to be ready to take over the attack. This support position depends on where the ball is. The same applies to the get behind ball, although the aim also includes breaking the symmetry so there is no ambiguity on which robot should attack.

Which Teammate to Back Off From

But before the robot can determine the outcomes from this region division, it has to decide which teammates it should take into consideration. There are two cases that need to be concerned with.

- The visual teammate with the shortest distance to the robot and is within 1 meter from the robot, or if both teammates have roughly the same distances from the robot,
- The visual teammate with the largest angular offset from the DKD and is within 1 meter from the robot.

If neither of this is the case, then the robot will not even carry out the region division and the list of checks stated above.

13.3 Wireless Support

Another type of local strategy comes from the teammate wireless strategy information. The aim of relying on wireless information as opposed to visual information is that either robot might not be able to see each other, or even the robot has temporarily lost the visual ball. In such cases if the teammate signals that it has the ball and the teammate is closer to the ball, then it should support its teammate. But this kind of wireless support decision occurs less often than the visual back off.

13.4 Matching a Visual Teammate to Its World Model

There are a number of reasons that world model teammate estimates are more preferable than visual information. Although visual information is up to date it can contain a significant amount of noise and can vary a lot from frame to frame. On the other hand the world model information is more stable and accurate which is suitable for strategic purpose due to fewer fluctuations. Besides, the wireless strategy information is also contained in the world model information but it is not part of the visual information. Examples of these wireless data are the teammate distance to the visual ball, whether the teammate can see the ball and its behavioural role. Hence if the robot tries to extract the wireless strategy information from a visual robot, it is necessary to find the corresponding gps teammate first.

Whenever the robot considers the positions of the visual teammate and furthermore needs any wireless strategy data, it first attempts to map the correct gps teammate by considering which gps teammate is closest to the visual one, and they must differ by less than a certain distance threshold and within a specific variance. But if it is not satisfied, the robot has no choice but rather continues the tactic based on the visual information.

13.5 Support Positioning

The implementation of the support position has not changed from last year, as illustrated by the diagram below.

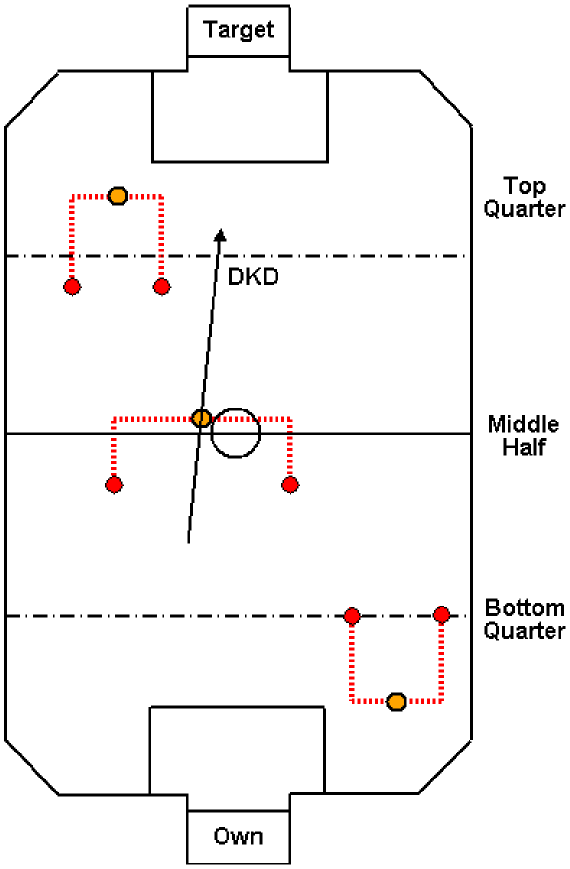

Fig 13.5
Support Positioning. The 2 red dots show the 2 possible points that the supporter can place itself. Generally the support positions are relative to the ball. But in the case of a lost ball it is relative to the teammate position if the teammate is close by.

The L shape is designed to allow the supporter to sustain the attack if the chief attacker loses the ball by quickly jump into the front and intercept the ball. Depending on which side the robot is on relative to the DKD line, it has two choices on the support position for each ball location. On the edges it is overridden such that it does not crash into the wall. Furthermore on the bottom quarter of the field the shape is inverted. If it were below the ball the robot would have no space to support as well as obstruct the goalie's view. The wider shape in the middle half is designed to allow the supporter to cover up wider area to intercept any ball propelled down to its own half by opponents.

13.6 Position Based on Teammate Rather Than Ball

Additionally if the ball source is the gps ball and the robot is close to the world model teammate within certain variance, then it uses the teammate gps position to determine the supportive position rather than using the ball since the gps ball can be quite inaccurate after a period of ball loss.

13.7 Side Back Off

The final type of local interaction is the side back off. The situation of where the side back off is useful is when two robots running for the ball side by side, they cannot see each other at all but only concentrate on the ball. Their paths converge and the two robots can easily leg lock. To avoid such situation one of the robot should slow down and let the other teammate to attack as well as be able to see its teammate in order to perform the visual back off.

The detection for this situation needs careful justification though, because it relies on world model and wireless information only. It is only activated if:

- Recent wireless information is available.
- World model information has low variances.
- Both the robots and its teammate are close to each other.
- Both the robots and its teammates can see the ball in front.

Then if the robot finds that it is further then the ball than its teammate is, it starts the side back off. If there is no distinguishable difference in the distances but the robot has a higher player number it also starts the side back off. The side back off simply reduces the forward speed by 40%, such that the other attacker is clearly in the front and the robot can then back off visually to the support position. During the side back off, the robot is still in the normal attacker mode, the only difference is at the end its forward speed is forced to decrease.

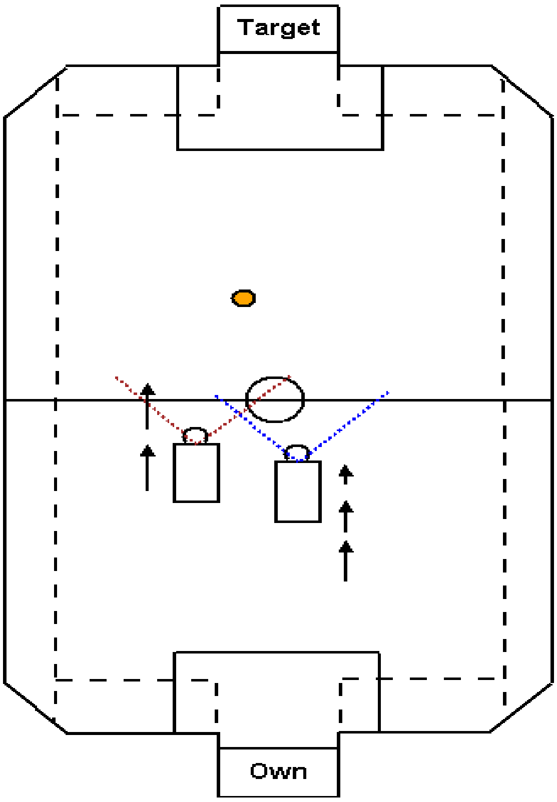

Fig 13.6

When both robots are focusing on the ball, they cannot see each other and can easily leg locked. Side backoff is useful in this case to slow down one of the robot.

Chapter 14

Global Robot Interactions

Global robot interactions refer to the tactics of the long distance support role, which is better known as the striker. The striker provides a long range support to the two attacking forwards, in a similar manner as the close supporter provides assistance to the main attacker. It usually stays on the side of the field opposite to where the ball and the other teammates are.

14.1 Trigger for the Striker Role

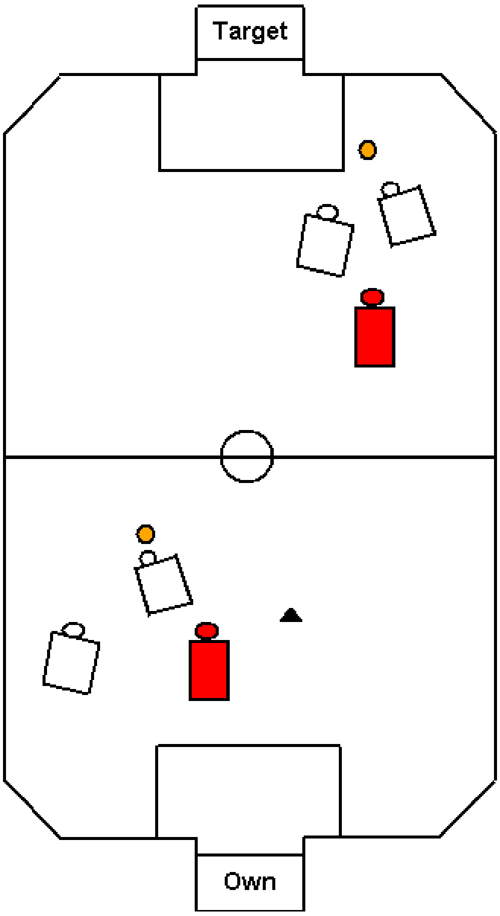

Fig 14.1
Striker role determination. For the upper team, the red robot is obviously furthest away from the ball and hence it is a striker. For the lower team the bottom 2 robots have roughly the same distance to the ball. But the red robot is much closer to the striker point, marked with a triangle, so it is the striker.

The assignment of the striker role is generally the robot that is furthest away from the ball (obviously if the robot can find a teammate which is furthest away from the ball it does not turn itself into striker because it assumes its teammate will). Each robot determines its distance to the visual ball and broadcasts this among its teammates. But if the teammates cannot see the visual ball, the robot will use its world model information about its teammates to estimate their distances to the ball.

However noises and distortions in the vision system can lead to inaccurate distance estimates and as a result incorrect role assignment. This is especially true when the robots are close to each other, in which case they cannot easily distinguish who is the furthest away from the ball and small errors could lead to frequent role fluctuation.

If the robot cannot tell which robot (whether itself or its teammates) is obviously furthest away from the ball, it proceeds to a more careful judgment process. It calculates the striker point, which is relative to the ball position, and finds which robot is closet to this point. If the robot is the closest, it becomes the striker otherwise it leaves its teammate to take this role.

Although the checks above work together in a robust way, there could still be seldom occasions in which two robots declare to be strikers simultaneously due to various errors. To prevent such situation the striker will broadcast in the wireless team strategy to explicitly inform the teammates that it has become the striker, and only the robot with the highest player number will take this role if there are 2 strikers detected.

14.2 Positioning of the Striker

Once the robot is assigned a striker role, it approaches to the striker point immediately. The positioning system remains the same as the last year X positioning. In this scheme four points are fixed on the field close to the corners of the goal boxes. These four vantage points are connected to the centre point to form a skewed X shape, as shown in Fig 14.2.

The striker desired position is on any points along these four lines, depending on where the ball is. It uses the left part of the X, that is AX and CX if the ball is on the right side of the field and uses the right part of the X (BX and DX) if the ball is on the left side, such that it can cover the field as large as possible.

To determine where on the lines the striker point is, the robot varies its y coordinate according to the ball’s y coordinate. When the striker point is below point X, the robot is always 140 cm behind the ball in the y direction along the lines DX and EX and the striker point is limited to the vantage points C and D as the lowest points.

Once the striker point is above point X, the robot is not staying behind the ball at a constant distance but decreases linearly as the ball is moving to the top of the field. So the striker is closer to the ball and thus more aggressive when close to the target goal.

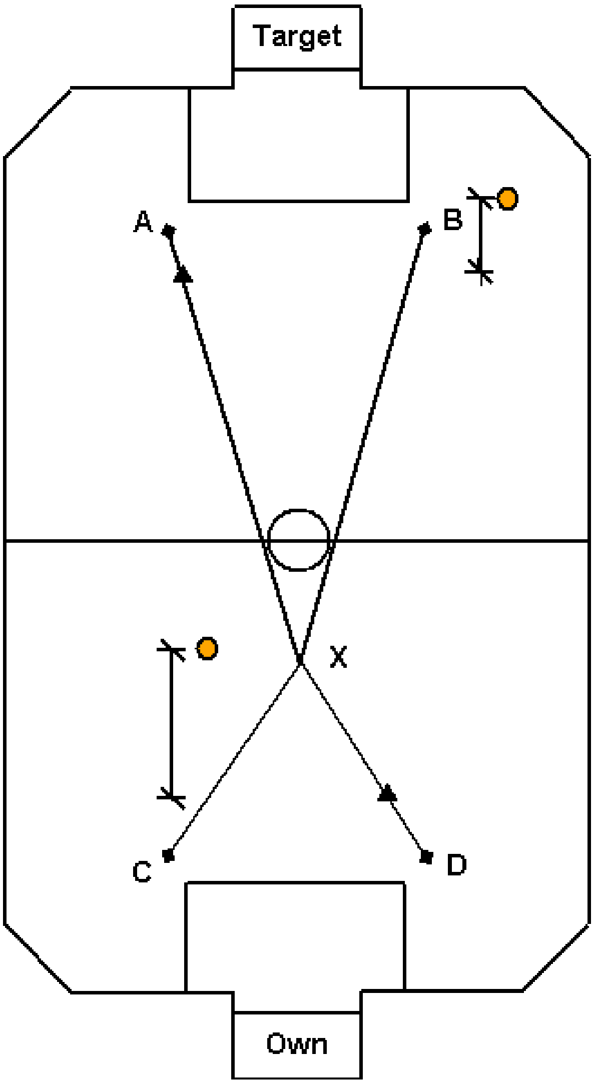

Fig 14.2
The X positioning finding the striker point. The point on the lines marked with a triangle is the striker point. A, B, C and D are the maximum and minimum points that the striker can be. When the striker point is below X the robot is staying behind the ball in a constant distance. While above X, this distance starts to decrease linearly.
Note the striker point is on the side of the field opposite to the ball.

When the striker is moving towards the striker point, it is constantly facing the ball such that it is always ready for attacking, although it is slow since the robot is side walking most of the time. To solve this problem the new implementation is that if the striker point is more than 80 cm away the robot walks straight towards the striker point and turns towards the ball right on the spot. This speeds up the striker movement a lot.

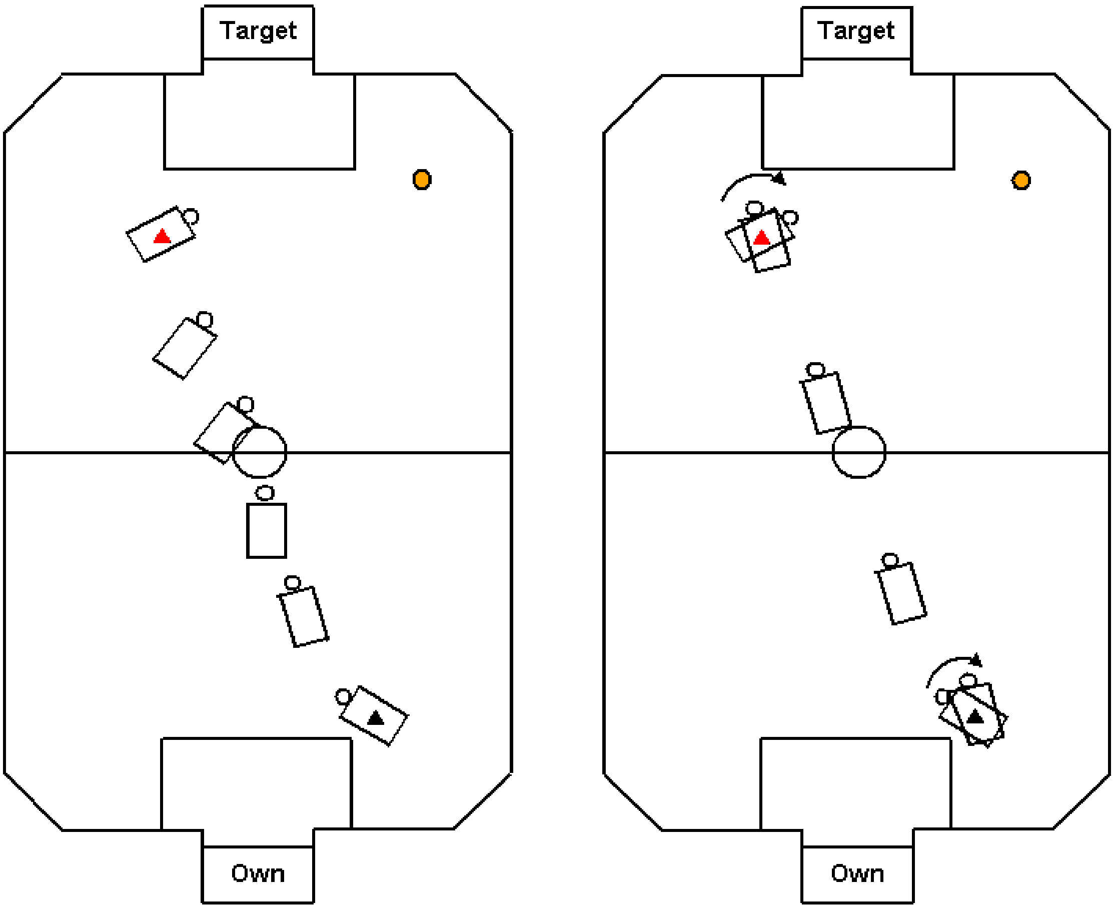

Fig 14.3

On the left the striker side walks a lot, and it is slow to move to a new striker position. But if the striker turns before and after the move, then it can run on a straighter path and it is faster, as illustrated on the right.

However, it can introduce another problem where the robot is facing away from the ball during the time it is running to the far away striker point. Hence another condition is to check whether the ball is within 60^o of the robot's heading, if it is not, it will switch back into the normal side walking fashion.

Since the striker point is calculated every frame, the new striker point tends to oscillate from frame to frame. Therefore a hysteresis is used in the striker. When it arrives within 10 cm of the striker point it stays there until the striker point has shifted more than 20 cm away from its current position.

Chapter 15

Bibliography

[1] Z. Wang, J. Wong, T. Tam, B Leung, M.S. Kim, J. Brooks, A. Chang, N.V. Huben, ***rUNSWift Team Report 2002***, http://www.cse.unsw.edu.au/~robocup/reports.phtml

[2] J. Chen, E. Chung, R. Edwards, N. Wong, ***Rise of the AIBOs – AIBO Revolutions (2003 rUNSWift Team Report)***, http://www.cse.unsw.edu.au/~robocup/reports.phtml

[3] RoboCup Technical Committee, ***Sony Four Legged Football League Rule Book***, http://www.tzi.de/~roefer/Rules2004/Rules2004.pdf, May 20, 2004.

[4] T. Rofer, H.D. Burkhard, U. Duffert, J. Hoffmann, D. Gohring, M. Jungel, M. Lotzsch, O.V. Stryk, R. Brunn, M Kallnik, M Kunz, S Petters, M Risler, M Stelzer, I Dahm, M Wachter, K. Engel, A Osterhues, C Schumann, J. Ziegler, ***German Team RoboCup 2003 Report***, http://www.robocup.de/germanteam/index-old.html

[5] University of Michigan and CMU, ***PID Control Theory***, www.engin.umich.edu/group/ctm/PID/PID.html.

[6] Homepage of the team ***UPennalizer***, University of Pennsylvania, http://www.cis.upenn.edu/robocup/index.php

[7] Homepage of the team ***NUBots***, University of Newcastle, http://robots.newcastle.edu.au/robocup.html

[8] Homepage of the ***OPEN-R SDK*** for the AIBO, Sony Corporation, http://openr.aibo.com/openr/eng/index.php4

[9] J. Hoffmann, D. Gohring, ***Senor-Actuator-Comparison as a Basis for Collision Detection for a Quadruped Robot***, http://www.robocup.de/aiboteamhumboldt/index.html, 2003.

[10] M. Lotzsch, J. Bach, H.D. Burkhard, M. Jungel, ***Designing Agent Behaviour with XABSL***, http://www.robocup.de/germanteam/index-old.html, 2003.

[11] Kim Cuong Pham, ***rUNSWift Individual Report 2004***, to appear.

[12] Chi Kin Lam, ***Thesis Report on rUNSWift 2004***, to appear.

[13] Ted Wong, ***Thesis Report on rUNSWift 2004***, to appear.

[14] Derricks White, ***Thesis Report on rUNSWift 2004***, to appear.

[15] M. Quinlan, C. Murch, R. Middleton, S. Chalup, ***Traction Monitoring for Collision Detection with Legged Robots***, RoboCup 2003 Symposium, 2003.