The UNSW RoboCup 2000 Sony Legged League Team

Bernhard Hengst, Darren Ibbotson, Son Bao Pham, John Dalgliesh, Mike Lawther, Phil Preston, Claude Sammut

School of Computer Science and Engineering
University of New South Wales, UNSW Sydney 2052 AUSTRALIA
{bernhardh,s2207509,sonp,johnd,mikel,philp,claude}@cse.unsw.edu.au

Abstract. We describe our technical approach in competing at the RoboCup 2000 Sony legged robot league. The UNSW team won both the challenge competition and all their soccer matches, emerging the outright winners for this league against eleven other international teams. The main advantage that the UNSW team had was speed. The robots not only moved quickly, due to a novel locomotion method, but they also were able to localise and decide on an appropriate action quickly and reliably. This report describes the individual software sub-systems and software architecture employed by the team.

1 Introduction

Each team in the Sony legged robot league consists of three robots that play on a pitch about the size of a ping pong table. All teams use the same Sony quadruped robots. The 2000 competition included entries from twelve international laboratories. Since all teams use the same hardware, the difference between them lies in the methods they devise to program the robots. The UNSW team won the championship as a result of their innovative methods for vision, localisation and locomotion. A particular feature of these methods is that they are fast, allowing the robots to react quickly in an environment that is adversarial and very dynamic.

The architecture of the UNSW United software system consists of three modules that provide *vision*, *localisation* and *action* routines. A *strategy* module coordinates these capabilities. Currently two strategy modules implement the roles of forward and goalkeeper. Each role can invoke a set of behaviours to achieve its goal. In the following sections, we describe the infrastructure modules that perform the vision processing, object recognition, localisation, and actions. We then describe the basic behaviours and strategies.

2 Vision

Since all the objects on the field are colour coded, the aim of the first stage of the vision system is to classify each pixel into the eight colours on the field. The colour classes of interests are orange for the ball, blue and yellow for the goals and beacons,

P. Stone, T. Balch, and G. Kraetzschmar (Eds.): RoboCup 2000, LNAI 2019, pp. 64-75, 2001.

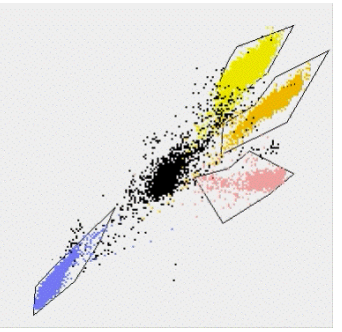

Fig. 1 A polygon growing program finds regions of pixels with the same colour.

pink and green for the beacons, light green for the field carpet, dark red and blue for the robot uniforms.

Currently, we only use the medium resolution images (88 x 60 pixels) available from the camera. The information in each pixel is in YUV format, where each of Y, U and V is in the range 0 to 255. The U and V components determine the colour, while the Y component represents the brightness. The Sony robots have an onboard hardware colour look up table. However, for reasons that will be explained later, we have chosen to perform the colour detection entirely in software.

Our vision system consists of two modules: an offline training module and onboard colour look up module. The offline software generates the colour tables and stores them in a file. At boot time, the onboard software reads the colour table from the file and then performs a simple table lookup to classify each pixel in the input image. We next explain how the colour table is generated.

Because colour detection can be seriously affected by lighting conditions, we need a vision system that can be easily recalibrated. The first step is to take about 25 snapshots of different objects at different locations on the field. Then for each image, every pixel is manually classified by "colouring in" the image by hand, using a purpose designed painting program. The labelled pixels form the training data for a learning algorithm.

In the 1999 team's software, all pixels were projected onto one plane by simply ignoring the Y value. For each colour, a polygon that best fits the training data for that colour was automatically constructed. An unseen pixel could then be classified by looking at its UV values to determine which polygons it lies in. As the polygons can overlap, one pixel could be classified as more than one colour.

Figure 1 shows a screen grab at the end of a run of the polygon growing algorithm. It also illustrates why we chose to use polygonal regions rather than the rectangles used by the hardware colour lookup system. We believe that polygonal regions give greater colour classification accuracy.

For the 2000 competition, we kept the polygon growing algorithm but now also use the Y values. Initially, Y values were divided into eight equally sized intervals. All pixels with Y values in the same interval belong to the same plane. For each plane, we run the algorithm described above to find polygons for all colours.

Once the polygons have been found, they must be loaded onboard the robots to allow them to perform the colour lookup. Because we cannot use the Sony hardware, the colour information must be stored in such a way as to allow fast operation in software. We chose a set of two-dimensional arrays, where each <U, V> pair specifies one element in an array. The value of the element is determined by the polygons in which the <U, V> values lie. To determine the colour of an unseen pixel, the Y value is first examined to find the relevant plane, then <U, V> index into the array and the value of that element gives the colour.

Discretisation of the Y values into eight equal intervals leads to better colour discrimination, but the robots were still unable recognise the red and blue colours of other robots. To the onboard camera, those colours appear very dark and were being mapped to the same plane as black and other dark colours.

Being able to classify these colours is vital for robot recognition and consequently, team play, so a further refinement was attempted. A manual discretisation of Y values was attempted, settling on 14 planes of *unequal* size. More planes are assigned to lower Y values, reflecting the fact that dark colours are hard to separate. With 14 planes, the robots can recognize the colour of the robot uniforms with reasonable accuracy, but further work is required to obtain greater reliability.

The 1999 version of the polygon growing algorithm allowed polygons to overlap. Pixels could be classified as more than one colour. This caused two problems. One is the obvious ambiguity in classification; the other is inefficiency in storage. By ignoring pixels that occur in overlapping polygons, we removed the overlap. Object Recognition

Once colour classification is completed, the object recognition module takes over to identify the objects in the image. Four-connected colour blobs are formed first. Based on these blobs, we then identify the objects, along with and their distance, heading and elevation relative to the camera and the neck of the robot.

2.1 Blob formation

The robot's software has a decision making cycle in which an image is grabbed, and object recognition and localisation must be performed before an appropriate action is chosen and then executed. Thus, every time the robot receives an image, it must be processed and action commands sent to the motors before the next image can be grabbed. The faster we can make this cycle, the quicker the robot can react to changes in the world. Blob formation is the most time-consuming operation in the decision making cycle. Therefore a fast, iterative algorithm [3] was developed that allows us to achieve a frame rate of about 26 frames/second most of the time,.

2.2 Object Identification

Objects are identified in the order: beacons, goals, ball and finally the robots. Since the colour uniquely determines the identity of an object, once we have found the bounding box around each colour blob, we have enough information to identify the object and compute various parameters. Because we know the actual size of the object

and the bounding box determines the apparent size, we can calculate the distance from the snout of the robot (where the camera is mounted) to the object. We then calculate heading and elevation relative to the nose of the robot and the blob's centroid.

Up to this point, distances, headings, etc, are relative to the robot's snout. However to create a world model, which will be needed for strategy and planning, measurements must be relative to a fixed point. The neck of the robot is chosen for this purpose. Distance, elevations and headings relative to the camera are converted into neck relative information by a 3D transformation using the tilt, pan, and roll of the head [2].

Every beacon is a combination of a pink blob directly above or below a green, blue or yellow blob. The side of the field the robot is facing is determined by whether the pink blob is above or below the other blob. The beacons are detected by examining each pink blob and looking for the closest blob of blue, yellow or green to form one beacon. Occasionally, this simple strategy fails. For example, when the robot can just see the lower pink part of a beacon and the blue goal, it may combine these two blobs and call it a beacon. A simple check to overcome this problem is to ensure that the bounding boxes of the two blobs are of similar size and the two centroids are not too far apart. The relative sizes of the bounding boxes and their distance determine the confidence in identifying a particular beacon.

After the beacons have been found, the remaining blue and yellow blobs are candidates for the goals. The biggest blob is chosen as a goal of the corresponding colour. Since the width of the goal is roughly twice as long as the height of the goal, the relative size between height and width of the bounding box determines confidence in the identification of that goal. There are also some sanity checks such as the robot should not be able to see both goals at the same time and the goal cannot be to the left of left-side beacons nor to the right of right-side beacons. Sometimes, the robot would try to kick the ball into a corner because it could only see the lower blue part of the beacon in the corner and would identify that as the goal. To avoid this misidentification, we require the goal to be above the green of the field.

The ball is found by looking at each orange blob in decreasing order of bounding box size. To avoid misclassifications due to orange objects in the background, the elevation of the ball relative to the robot's neck must be less than 20°. The elevation must also be lower than that of all detected beacons and goals in the same image.

When the camera is moving, pixels are blurred, resulting in the combination of colours. Since red and yellow combine to form orange, red robots in front of a yellow goal can be misclassified as the ball. A few heuristics were used to minimise the problems caused by this effect. If there are more red pixels than orange pixels in the orange bounding box then it is not the ball. When the ball is found to be near the yellow goal, it must be above the green field. That is, if the orange blob must be above some green pixels to be classified as the ball. These heuristics allowed our robots to avoid most of the problems encountered by other teams. However, more work is required to completely overcome this problem.

2.3 Robot Recognition

The robot recognition algorithm used at RoboCup 2000 uses a combination of visual and infrared sensors to identify the presence of, at most, one robot in the visual field and to approximate the distance from the camera to the object. For the purposes of obstacle avoidance, important frames generally don't contain multiple robots.

The on-board infrared sensor provides accurate distance information for any obstacle aligned directly in front of the head of the robot at a distance between 10-80cm. Below 10cm, the IR will read somewhere between 10-20cm. The main noise factor for the IR sensor is the ground. A work-around for this is that the IR reading is passed on as full range (1501mm) when the IR sensor is pointing downward more than 15°.

The initial design of the robot recognition algorithm was based upon a sample of 25 images of robots taken from the robot's camera, as well as manually measured distances to each of the robots in the samples. A further set of 50 images was taken when new uniforms were specified by Sony. The colour detection and blob formation algorithms were run over the sample images and blob information obtained. Blobs with fewer than ten pixels were discarded as noise and the following two values calculated: total pixels in each blob and the average number of pixels per blob. From this information, a curve was manually fitted to the sample data and a distance approximation was derived based purely on the feedback from the camera.

While the vision system is used to detect the presence of a robot and estimate its distance at long range, the infrared sensor is used at short range. Once the distance has been approximated, several sanity checks are employed. These filter out spurious long-range robot detections (> 60 cm) and robots that are probably only partially on camera, that is, the number of patches of the uniform is unlikely in comparison to distance.

Although robot recognition was not very reliable, using the infrared sensors at short ranges allowed the algorithm to identify situations where there is a risk of collision. The primary weakness of robot recognition is its reliance on accurate colour classification. The algorithm does not adapt well to background noise that often causes it to misclassify a robot or produce a grossly inaccurate distance approximation. This main weakness is almost exclusive to blue robot detection, with the red uniforms being far easier to classify accurately.

3 Localisation

The Object Recognition module passes to the Localisation module the set of objects in the current camera image, along with their distances, headings and elevations. Localisation tries to determine where the robot and other objects are on the field. It does so by combining its current world model with the new information received from the Object Recognition module. Since all beacons and goals are static, we only need to store the positions of the robot and the ball. We do not attempt to model the other robots.

The world model maintains three parameters for the robot: its x and y coordinates and its heading. The left-hand corner of the team's own goal is the origin, with the x-axis going through the goal mouth. The robots first attempt to localise using only the objects detected in the current image. Being stationary, beacons and goals serve as the landmarks to calculate a robot's position. Because of the camera's narrow field of view, it is almost impossible to see three landmarks at once, so any algorithm that requires more than two landmarks is not relevant. If two landmarks are visible, the robot's position is estimated using the triangulation algorithm used by the 1999 team [2]. This technique requires the coordinates, distance and heading of two objects relative to the robot.

More information can be gathered by combining information from several images. Thus, the localisation algorithm can be improved by noting that if the robot can see two different landmarks in two consecutive images while the robot is stationary, then triangulation can be applied. Typically, this situation occurs when the robot stops to look around to find the ball. To implement this, we use an array to store landmark information. If there is more than one landmark in the array at any time, triangulation is used. This array is cleared every time the robot moves.

The world model receives feedback from the locomotion module, *PWalk*, to adjust the robot's position. The feedback is in the form the distances, in centimetres, that the robot is estimated to have moved in the x and y directions and the number of degrees through which the robot is estimated to have turned. This feedback is received about every 1/26 second when the robot is moving. Odometry information is clearly not very accurate and small errors in each step accumulate to eventually give very large inaccuracies. Also, if the robot is blocked by an obstacle, *PWalk* is not aware of this and sends incorrect information to the world model.

Since the robot usually sees only one landmark in an image, we devised a method for updating the robot's position from a single landmark. This is explained in Figure 2, with details given in [3]. The main feature of the algorithm is that with a fast frame rate of about 26 frames/second, it converges on an accurate estimate of the robot's position quite quickly and accurately. Within a reasonably short period of time, the robot usually catches sight of several landmarks, thus approximating triangulation. One landmark update overcomes many of the problems caused by odometry error. Even when the robot's movement is blocked and the odometry information is incorrect, if the robot can see one landmark it will readjust its position. Because we use a low trot gait, the robot can see goals and beacons most of the time, if they are not obscured by other robots.

One problem remains due to the perception of landmarks. A goal is large and often the robot is only able to see a part of it or much of it may be obstructed by the goalie. Consequently, distance measurements may be inaccurate. Therefore, when the robot is in the middle of the field or near the edge, the goals are ignored, if the beacons are visible. Near a goal, the beacons are often difficult to see, so the heading of the goal is used to update the robot's heading. However, the robot's (x, y) position is not updated because the measurement of distance to the goal is too unreliable. Thus, the goal is never used in triangulation.

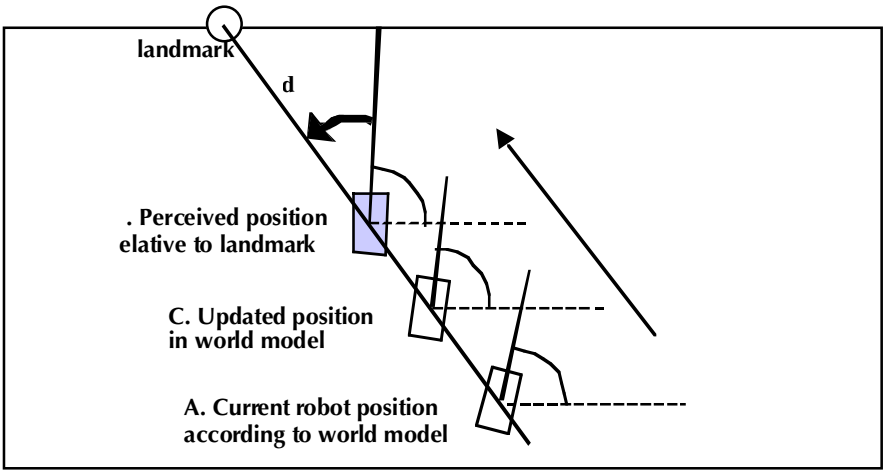

Fig. 2. To estimate the position of the robot, we draw a line between the landmark and the estimated current position (A) of the robot in the world model. The robot's perceived position (B), relative to the landmark, is the point on that line *d cm* away from the landmark. The localisation algorithm works by "nudging" the estimated position in the world model (C) towards the perceived position relative to the landmark.

4 Action/Execution

The purpose of the action module is to move the head and legs in response to commands from the behaviour module. Head and leg commands are given concurrently and are executed concurrently. The three primary design objectives of the action module were to:

1. Drive the robot as if controlled by a joystick with three degrees of freedom: forward, backward, sideways left or right and to turn on the spot clockwise or counterclockwise.
2. Move the robot over the ground at a constant speed, thus reducing the strain on the robot motors by not accelerating and decelerating the body.
3. Keep the camera as steady as possible. Using other walks, we observed that images from the robot's camera showed wildly erratic movements due to the robots head and leg motions.

The solution adopted was to move the paws of the robot's feet around a rectangular locus (Figure 3). The bottom edge of the rectangle describes that part of the path during which the paws make contact with the ground. The sides and top of the locus describe the path used to lift the paw back ready for it to take the next step. In the trot gait, used for the competition, diagonally opposite legs touch the ground alternately. If the paws that touch the ground move at a constant velocity, the robot should move at that same constant velocity. This requires that the time taken to move the paw

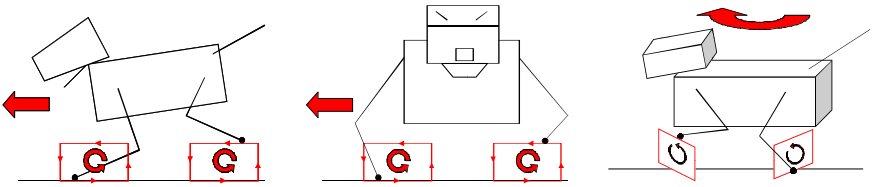

Fig. 3. Forward, sideways and turning achieved by adjusting angles of rectangular locus of leg movement

along the bottom edge of the rectangle is equivalent to the total time taken to move the paw along the other three edges.

Design objectives 2 and 3 were achieved in this way. The speed over the ground is constant as long as the size of the rectangular locus does not change and its traversal is at a constant frequency. The robot's camera is steadied because the bottom edge of the rectangular locus is a straight line lying in the ground plane. When the robot loses balance in the trot walk there is camera movement until it is arrested by falling on one of the legs that is off the ground. This movement can be minimised by lifting the legs as little as possible during the walk. Unfortunately in practice, it was necessary to specify a significant leg lift height to ensure that the spring-loaded claws would clear the carpet. This introduced some unwanted camera movement.

We now address design objective 1, that is, how to control which way the robot moves. The plane containing the rectangular locus for the paw is always perpendicular to the ground. By changing the angle of this plane relative to the sides of the robot, we determine whether the robot moves forward, backward or sideways. For example, if the locus plane is parallel to the sides, the robot will move forward or backward (Figure 3(a)). If we angle the locus plane perpendicular to the sides of the robot it will either move left or right (Figure 3(b)). Figure 3(c) shows how the robot can be made to turn by angling the locus planes tangentially at each shoulder.

Components of each of the three movements can be combined, so that the robot moves forward, sideways and turns simultaneously. The width of the rectangular locus and the speed at which it is traversed by the paw determine the speed at which the robot moves.

Twelve parameters influence the leg movements for a particular walk style. We considered automating the search for the best parameter settings. However, we were concerned about the wear on the robot motors and leg joints, given the long periods of training required. Hornby, *et al* [1] report training times of 25 hours per evolutionary run using their genetic algorithm (GA). The approach we adopted was to manually adjust the parameters after a number of runs of observing and measuring the performance. Unlike gradient ascent or a GA, we were able to adjust the parameters using our judgement and knowledge about the robot's dynamics.

A refinement that increased ground speed considerably was the introduction of a canter. The canter sinusoidally raises and lowers the robot's body by 10mm synchronised with the trot cycle. The parameters were manually tuned so that the robot was able to reach speeds of 1200cm/min. This compares to 900cm/min achieved in [1] using a genetic algorithm, which was reported to have improved on the

previously fastest hand developed gait of 660cm/sec. The camera is not as steady in this type of walk because of the additional canter movement.

5 Behaviours

The team consists of two Forwards and a Goalkeeper. Each role has its own set of strategies, described below.

5.1 The Forward

The pseudo code below describes the Forward's high-level strategy.

```
if see team mate at a distance < 15 cm
      backup
else if no ball in world model
      findBall;
else if canKickBall
      kickBall;
else if canChargeBall
      chargeBall;
else getBehindBall;
```

There are five main skills namely *backup*, *findBall*, *kickBall*, *chargeBall* and *getBehindBall*, which we now explain.

backup

When a robot sees one of its teammates nearby, it backs away, reducing the chances of our robots interfering with each other. The backup behaviour tends to keep one robot on the "wing" of its teammate, which effectively makes one robot wait for the ball. If the robot with the ball loses it, the "wing" can quickly pick it up.

findBall

When the robot does not know where the ball is, it moves the head in a rectangular path searching for the ball. The head is moved in a direction so that if it hits the ball in the lower scan, the ball will roll in the direction of the target goal. If the ball is still not found, the robot turns 45°. The direction of the turn is also chosen so that if the robot accidentally hits the ball, it will hit the ball towards target goal. The robot continues alternately turning and scanning the head until it finds the ball or when it has made six turning moves. When it has turned 45° six times without seeing the ball, it is likely that the ball is obstructed or outside its field of vision. The robot then goes to a defensive position and spins on the spot until it sees the ball.

kickBall

The kick is intended to be accurate, easy to line up and powerful. We tried may variants such as using a single leg or dropping the head on ball, but found that bringing both fore-limbs down in parallel on the ball best met the objectives.

For the kick to be effective, the ball has to be positioned between the front legs of the robot and touching the chest. The kick is implemented as two sets of absolute leg positions executed sequentially. The motor joint positions were found by conducting many trials and adjusting them until the best performance was achieved.

It was found that when the robot is near the edge of the field, kicking is not very effective. When setting up to kick the ball, the robot approaches at approximately 80% of maximum speed and maintains a heading to the ball between ±15°. The robot only tracks the ball's movement with the head pan while the head tilt is held constant so that the head just clears the ball. Upon losing sight of the ball, the robot transitions into a very low stance with the head is placed directly above the ball. The mouth is then used to sense the presence of the ball, as it cannot be seen with the camera. If the mouth does not sense the ball or the ball is identified by the vision system, the kick is aborted. Once in possession of the ball, the robot can take a limited number of rotational steps, set at two complete steps for RoboCup 2000, to align the goal before triggering the kicking motion.

chargeBall

When the robot has the ball near the target goal, then it is worth taking time to line up on the goal. However, if the robot is far from the target, it is more effective to simply knock the ball into the opponent's half. This wastes little time and does not allow opponents the chance to take the ball away. Thus, the robot only tries to line up the goal and the ball in region the region in front of the target goal before it runs at the ball. There are two skills that the robot can use to charge with the ball namely *dribbling* and *head butting*.

Dribbling is invoked when the robot is facing the opponents' half, the ball is close and in front of the robot. The robot then starts walking forward with the head just above the ball, using the mouth to sense if it has the ball. If, after few steps, the robot does not sense the ball, it stops and takes a few steps backwards to try to bring the ball into view. If it sees the ball, the robot continues to walk with the ball at its "chest". Otherwise, this mode is exited.

If the ball is not in position to dribble, the robot will head butt or bunt the ball. Although head butting is not accurate, we only need to knock the ball to the other half. The bunting strategy is very simple in that the robot walks directly at the ball with full range head tracking enabled. Directional control of the ball is obtained by inducing a component of sideways walking in proportion to the heading of the target, relative to the robot.

With these strategies, the robots keep the ball moving constantly giving less chance for opponents to control the ball.

GetBehindBall

The ball circling technique used in both the Goalkeeper and Forward, defines parameters for the walk that drive the robot from any position on the field to a position directly behind the ball. This circling technique involves no aggressive transitions in the robot's movement, always keeps the ball in sight, and keeps the robot's body pointing toward the ball.

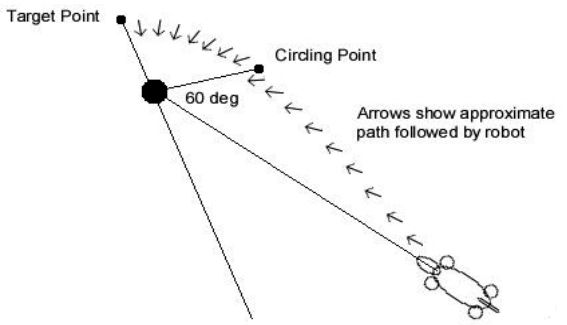

Fig. 4. Circling Skill

Ball circling is specified by two points (Figure 4). The target point is the intended destination and the circling point deflects the path around the ball. To perform the skill, the robot is simply driven towards the closer of the circle and target point, while the body is oriented towards the ball.

If the robot is trying to circle around the ball to line up the goal and it sees an opponent nearby, it will become more aggressive. The robot will run at the ball immediately as long as it is not facing its own goal.

5.2 The Goalkeeper

The goalkeeper has three behaviours used in defending the goal.

Finding the Ball

Finding the ball begins with a 360˚ rotation in the direction that would knock a ball stuck behind the robot away from the goal. Therefore, the robot will rotate clockwise on the left side of the field, otherwise anti-clockwise. During the rotation, the robot raises and lowers the head quickly to perform a combined short and long-range search. If the ball is not found during the rotation, the head of the robot begins following a rectangular search path scanning for the ball. At the same time, the robot orients itself facing directly away from the goal it is defending and walks backwards, to the goal. Once close to the goal, the goalie turns to face the opposing goal.

Tracking the Ball and Acquiring a Defensive Position

Once the ball has been found, the robot enters its tracking and defending mode. In this mode, the robot places itself on the direct line between the ball and the defended goal, at a position 45cm from the defended goal. As the ball position changes, the robot tracks along a semicircle around the defended goal, keeping the body oriented towards the ball. While tracking the ball, the robot oscillates the head side to side as much as it can, without losing the ball, to try to maximise the chances of seeing landmarks and help maintain localisation. However, watching the ball always takes precedence over localisation.

Clearing the ball

Clearing the ball is activated when the ball comes within 80cm of the goal or the ball enters the penalty area. It ends when the ball is kicked, lost or moves more than 120cm from the goal. Upon deciding to clear the ball, the robot determines whether it can directly attack the ball or should reposition itself behind the ball. Once the robot is clear to attack the ball, it aligns itself with the ball to kick it. On approach to the ball, if the ball gets out of alignment, the robot aborts its kick and simply bunts the ball with the front of the head.

6 Conclusions and Future Development

In reviewing why the UNSW team was successful, we can identify technical advances in locomotion, vision and localisation and the repertoire of behaviours that were developed. Some practical considerations also contributed to the team's win.

Following our experience in 1999, we decided that we would play regular games between teams of two robots. As we devised new strategies, these were played off against each other. We also insisted that whenever testing a new behaviour, we should have as many robots on the field as possible. These "management decisions" ensured that we tested the robots, as much as possible, under competition conditions and thus were able to discover and overcome many problems.

One consequence of this approach was that as a new special case was encountered, we introduced a new fix. It is evident from the description of our software that there are many *ad hoc* solutions to various problems. Thus, it might be argued that we are not learning much of general interest to robotics because we have not pursued solutions that are more general.

We believe there is much of value to be learned from our effort. It is clear that in a highly dynamic environment, speed of perception, decision making and action are essential. Our experience has been that implementations of very general approaches to these problems tend to be slow, whereas, problem-specific solutions are simple and fast. However, developing these problem-specific solutions is very labour intensive. Thus, one of the areas of future research for us will be in finding methods for automating the construction of domain-specific behaviour. The generality of our approach will, hopefully, be in the learning, and not in a particular skill.

References

1. Hornby, G. S., Fujita, M., Takamura, S., Yamamoto, T., Hanagata, O. (2000) *Evolving Robust Gaits with Aibo.* IEEE International Conference on Robotics and Automation. pp. 3040-3045.
2. Dalgliesh, J., Lawther, M. (1999). *Playing Soccer with Quadruped Robots*. Computer Engineering Thesis, Univesity of New South Wales.
3. Hengst, B., Ibbotson, D., Pham., Sammut, C. (2000). *UNSW RoboCup2000 Team Report*. http://www.cse.unsw.edu.au/~robocup.