The UNSW United 2000

Sony Legged Robot Software System

Bernhard Hengst
Darren Ibbotson
Son Bao Pham
Claude Sammut

School of Computer Science and Engineering

University of New South Wales

Sydney NSW 2052

Australia

http://www.cse.unsw.edu.au/~robocup

November 14, 2000

Abstract

In this report we present a comprehensive description of the software system developed to compete in the Sony legged robot league competition at RoboCup 2000. The UNSW team won both the challenge competition and all their soccer matches, to take the championship in a field of 12 teams. At RoboCup 2000, the UNSW robots had distinct advantages in locomotion, localisation and vision. This report describes the individual software sub-systems and the overall agent software architecture developed for the competition.

Acknowledgments

The 2000 RoboCup team thanks the 1999 team of John Dalgliesh and Mike Lawther. Although substantially rewritten, a number of the subsystems owe their origins to software developed last year.

We thank the Artificial Intelligence Research Group in the School of Computer Science and Engineering at the University of New South Wales for its technical and financial support of the RoboCup team. In particular, we thank Phil Preston.

Table of Contents

1 Introduction

1.1 The RoboCup Competition

The purpose of the RoboCup competition is to stimulate research in advanced mobile robotics. For a mobile robot to operate effectively it must be capable of at least the following:

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

The robot's planning becomes much more complex when its environment includes other active agents, such as other robots. These may cooperate with the robot or try to thwart its attempts to achieve its goals.

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

1.2 The Sony Legged Robot League

The Sony legged robot league is one of four leagues that currently form the RoboCup competition. In some leagues, the teams are required to design and build the robot hardware themselves. In the case of the Sony legged robot league, all teams use the same robotic platform, manufactured by Sony. The robots operate autonomously, meaning the robot is entirely on its own during the play of the game. Since all teams use the same hardware, the difference lies in the methods they devise to program the robots.

Each team in the robot soccer match consists of three robots with the matches consisting of two 10-minute halves. The 1999 competition included nine teams. In 2000 this was expanded to 12 teams in 2001, 16 teams will compete.

In 1999, the UNSW team were runners up and in 2000 UNSW become champions of the Sony legged robot league. Their success was due to the 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.

1.2.1 The Robots

The robots used in the Sony legged league are a slightly modified version of the Sony AIBO entertainment robot. The key differences are the inclusion of a PCMCIA slot

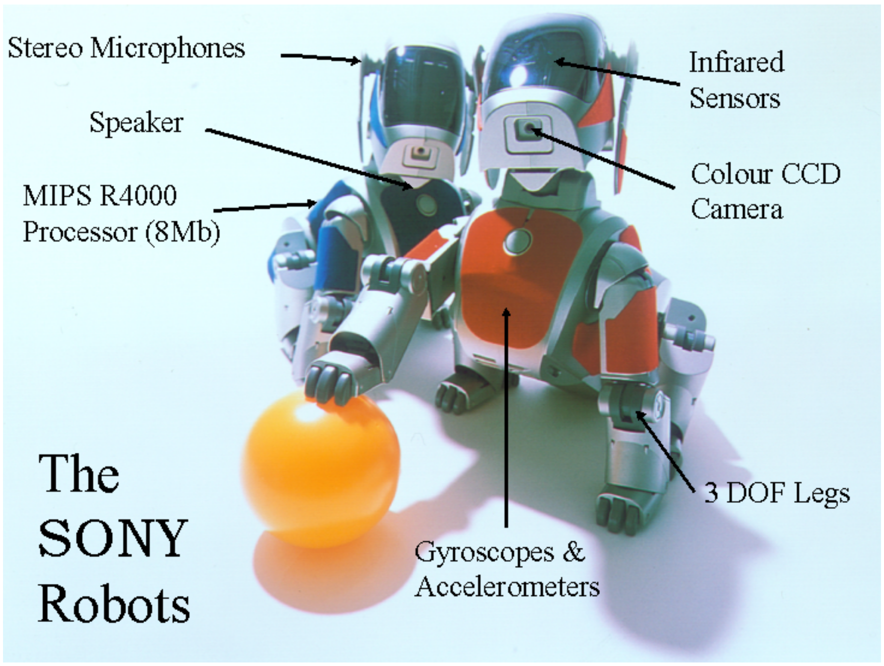


Figure 1 Features of the Sony legged robots

(currently unused) and NTSC video plug, giving a direct feed from the robots camera. They also differ in the software that controls the robots.

Although designed for the entertainment market, the Sony robots extremely sophisticated robots, with an on board MIPS R4000 processor, colour camera, gyroscopes and accelerometers, contact sensors, speaker and stereo microphones. Each of the four legs has three degrees of freedom, as does the head. Programs are written in C++ using a PC-based development environment. Code is loaded onto a memory stick that is inserted into the robot.

1.2.2 The Field

The field used in the Sony legged league measures 2 metres wide by 3 metres long. The field has an angled white border designed to keep the ball within the field. The game ball is coloured orange and the two goals are coloured yellow and blue. The field also contains six coloured distinguishable landmark poles to aid the robots in localising themselves in the field. In front of each goal a region is defined whereby only 1 defending robot is allowed. multiple attackers are allowed to enter the indicated region.

1.3 UNSW United 2000

The team began work on the project at the start of January 2000. The work of the 1999 UNSW United team was the starting point for this year s development. With advice from the previous team, we identified weaknesses in the existing architecture and devised a revised model.

The architecture used at RoboCup 2000 consists of the 4 main modules shown in figure 3.

The Vision and Object recognition module accepts images from the camera and identifies any key objects on the field that are visible. It delivers to the behaviour and

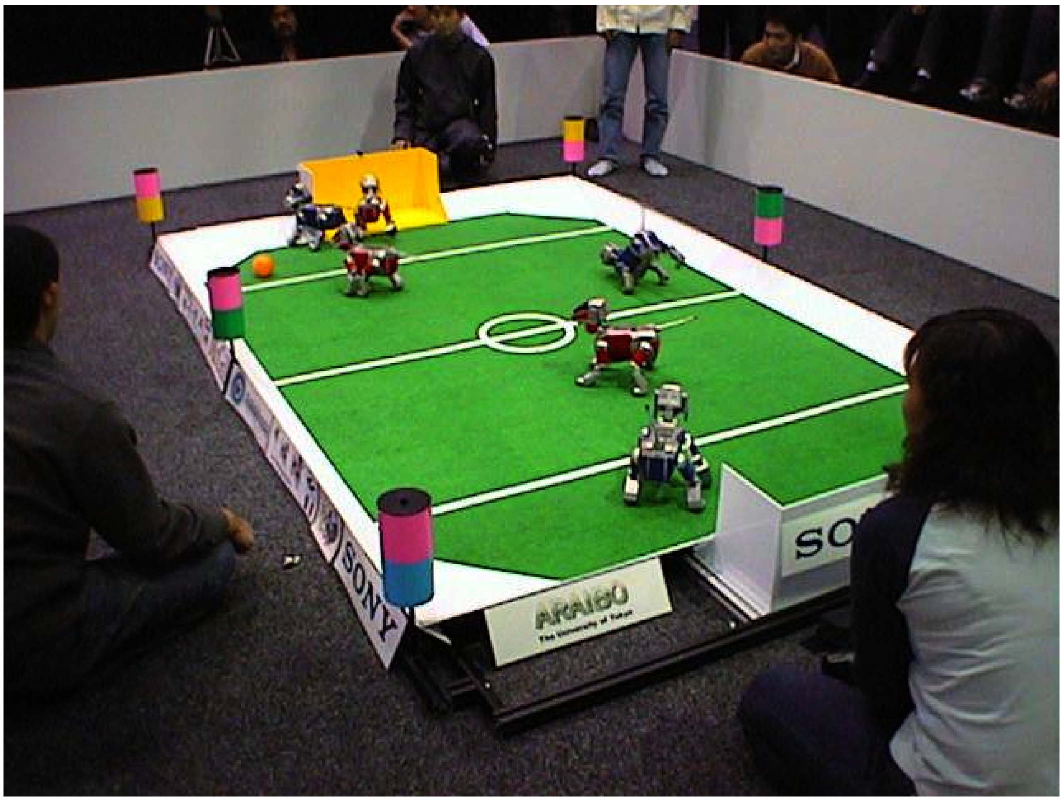


Figure 2 Sony Legged Robot League Field

localisation model any identified objects for the current frame as well as the information about them. For Robocup 2000 the colour classification model was expanded to incorporate the brightness of pixels, whereas the 1999 model only utilised chrominance information.

During the year, debugging tools were developed allowing the user to see the colours and objects identified by the robot. Utilising these tools, previously unforeseen problems were easily identified. Once identified, these problems were fixed by adding the appropriate mechanisms and checks.

The localisation module keeps track of identified objects and odometry from the action module. This information is used to approximate the position and heading of the robot on the field, as well as the position of the ball on the field.

Early in the year tools were developed to display a live feed of the robots perceived position on the field, as well as the robot s memory of landmarks. We noticed quickly that the robots perceived position would jump about wildly and the remembered location of landmarks was inconsistent with their true localisation. We decided that the existing representation was not very natural and increased the influence of noise in the system.

The existing representation places the robot as the centre of the universe, remembering the relative locations of landmarks. However, we felt it was more natural to represent the field as being a static entity with the robot moving about inside it. Therefore, the representation of the world model was changed to model only the position and orientation of the robot on the field. The methods for obtaining and updating the robots position have been changed accordingly to operate with the new representation.

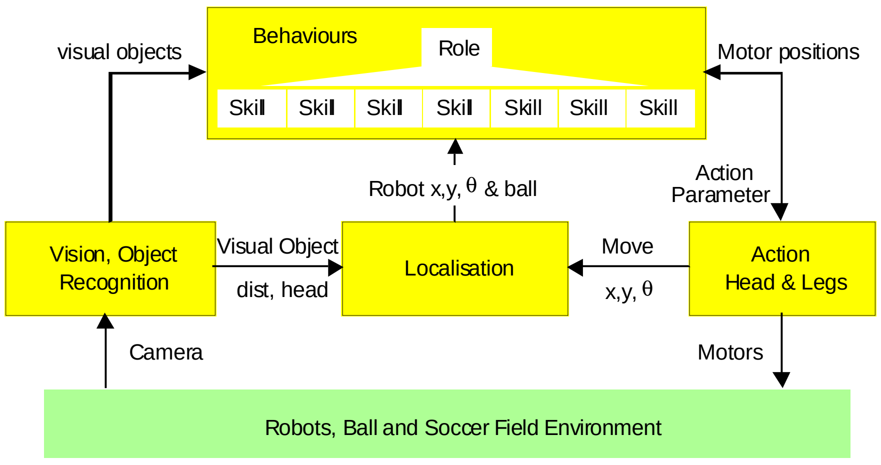


Figure 3 Architecture of the UNSW United software

In 1999 the team utilised Sony s provided walking routines to drive the robot. Unfortunately, we found the provided routines were not flexible enough to support the complex behaviours needed in a soccer match. Therefore, the team decided to implement a new action model for RoboCup 2000 with a focus on speed and manoeuvrability.

The initial ideas for the action module were developed and refined amongst the entire team, but the development and tuning were predominately the work of Bernhard Hengst, who we must thank greatly for such a sensational job.

The Action module accepts combined leg and head commands from the Behaviours module. The action module controls all of the motors on the robot and is responsible for creating motions and movements that the behaviours require. The Action module also approximates the odometry that will result from its actions and sends this to the localisation module in 1/25-second slices.

The behaviour modules assess the current situation for the robot and decide upon an appropriate action to help achieve their intended objective. The high level role has access to several complex skills that it can invoke. If the behaviours module has insufficient information to assess the current situation, it will perform an action to help acquire information from the environment.

During the course of the year, each team member developed different strategies. A process of natural selection was imposed with weekly matches determining which aspects would Ŝurvive . Only the final behaviours used at RoboCup 2000 are discussed in chapters 7 and 8.

2 Background

The four main elements of the software for the soccer playing are: vision, localisation, locomotion and behaviour. This chapter briefly describes some related work.

2.1 Vision

There are many different ways to represent colours including Red Green Blue (RGB), Hue Saturation Intensity (HIS) and YUV. The U and V values together describe the colour (without brightness) of the pixel and Y value represent the brightness of the colour. The YUV format can be converted into RGB format by one matrix multiplication (Dalgliesh and Lawther, 1999).

We just focus on the algorithms that classify pixels in YUV format for speed reason as images we get from the camera are in YUV format. Although RGB is an equivalent format, to classify a pixel in RGB format one matrix multiplication is required to convert a YUV pixel into RGB pixel. As we have to classify every pixel in one image, using RGB approach clearly is very time consuming.

The Sony legged robot has hardware on board to perform the colour classification. It is capable of classifying 8 colours simultaneously. 256 levels of luminance are divided into 32 equal size levels. At each level, user has to provide rectangles coordinates for each colour s/he wants to classify. The rectangle that a pixel lies in determines its colour. After providing all the colour rectangles, upon receiving an image in YUV format, the hardware returns an image in YUVC format where YUV values are unchanged and C is the classified colour of the pixels. Because C value is stored in one byte and each bit encodes one colour, the system is capable of classifying only 8 colours at a time. Most of the team competing last year used Sony hardware to classify images. The variation is how they determine the rectangles for colours in each of 32 levels.

The UNSW 99 team had an interesting colour classification approach. Their colour classification is done completely in software and they still get a relatively high camera frame rate and accurate classification compared to other teams. They ignore the Y values and just look at UV values. Several images were collected and then all pixels are manually classified to form a set of training data. From the training data, for every colour, one polygon that best fits the training data of that colour is automatically generated. The colour of an unknown pixel is then determined by checking what polygons its UV values lie in. The learnt polygons are encoded in such a way that the onboard classification can be done fast (Dalgliesh and Lawther 1999).

One way to recognize objects is to first find 4-connected blobs of colours and then determine objects from those formed blobs. We found that the blobs forming part are very expensive and responsible for the speed of the system. Algorithms to find blobs of the same colour are called filling algorithms. Two classic algorithms are FloodFill and BoundaryFill (Foley et al). These algorithms take a pixel in a connected region and mark all pixels belonging to the same region. It does so by first mark the pixel and then recursively mark all adjacent pixels. These algorithms are simple but highly recursive. Many levels of recursion take time and may cause stack overflow if memory is limited.

UNSW99 team implemented a non-recursive filling algorithm. The algorithm goes through the image line by line and finds all spans of the same colour in that line. Whenever a new span is found, it checks if this span is below another span of the same colour in the previous line. If there is no span of the same colour above it then this span creates a new blob. If it is under another span of the same colour, then this span belongs to the same blob as the one above it. If this span is under two spans of the same colour but belong to different blobs then these blobs can be merged into one. The merging is done immediately by scanning from the beginning of the image to this current point while adjusting those two blobs to represent one blob only.

The merging step is implemented by going back to the beginning of the image. This is expensive and not effective. It may lead to unnecessarily scanning the images multiple times.

CMTrio99 took a similar approach but had a more effective merging method. Their filling algorithm is done in two passes. The first pass is to form all the spans. However they don t try to merge blobs during the first pass but remember what spans belong to the same blob. The merging step is done separately in the second pass.

2.2 Localisation

Odometry is a very popular method for robot positioning in general because it is inexpensive and it provides short-term accuracy. However just relying on dead reckoning to localise in playing soccer is unrealistic for a number of reasons. Firstly, odometry might not be accurate, especially when robots are stuck or collide with obstacles. Currently, it seems impossible to detect such situations. Secondly, small error in odometry tends to accumulate over time leading to huge error if not corrected on the way by other means. Most of the team uses odometry partially in their robot localization.

To find the robot s position and angle on the field, CMTrio-98 used Bayesian probabilistic localization. The filed is divided into a 20x30x8 (X, Y,θ) state grid. Every state has a probability representing how likely the robot is in that state. Whenever the robot moves or sees some objects in the world, the probability distribution is updated accordingly. The updating process is based on Bayes rule. The actual location of the robot corresponds to the state with the highest probability. Because 8 directions is not accurate enough for effective strategy, 2 more Bayesian probabilistic localization systems are used in parallel with the one above. One is used to track absolute angle on the field with 100 states and the other is used to track the angle of the goal. This is an interesting approach but clearly has a speed disadvantage. Every time the probability distribution needs to be updated, all states are revisited. The complexity is therefore proportional to the number of states.

Monte Carlo Localisation (MCL), a sample-based algorithm for mobile robot localization, is another method that could be used to localize the robot s position (Fox et al. 1999). MCL maintains a sample set constituting a discrete approximation of robot s position probability distribution. The density of samples in a particular area is proportional to the probability of the robot in that area. The mean of all samples reveals the robot s position. Robot motion and sensor readings are the two types of updates applied to the sample set. When the robot moves, movement samples are generated to approximate the new robot position. New sample set is then determined by drawing samples randomly from the movement samples based on the movement probability density. Sensor readings are incorporated by readjusting the probability distribution of sample set using the Bayes rule. It is claimed that MCL is more accurate and effective than the Bayesian probabilistic approach both in memory and computation requirements. However, this method is still quite computationally intensive in the domain of Sony robot soccer as it requires a large sample set size.

At the competition site, we were told by McGill team that they used Monte Carlo Localization but it was not clear how they actually implemented it.

Monte Carlo Localisation was used in the Sony Legged League soccer domain by the CMU-99 team with a slight modification. Their localization tries to improve MCL by using fewer samples and handling unmodelled movements. Unmodelled movements occur when robots collide against each other or into the wall. It is also possible that the robot be picked up by the referee and put at an unknown location on the field. The extension lies in the new sensor update step. If the probability of the locale based on the current samples given the sensor readings is low then they replace some samples with samples drawn from the probability density given the sensor readings (Veloso *et. al*, 2000).

The UNSW United 99 team uses triangulation based on sensor readings to localize and updates the robot position using odometry when the robot moves. To use triangulation the heading and distance of 2 landmarks are needed. Because this information is usually not available in a single image, a list of landmark information seen over time is maintained. This means that whenever the robot moves, this information needs to be adjusted relative to the robot. Consequently, error in odometry will have a greater impact on the inaccuracy of the localization system.

2.3 Locomotion

Fast and manoeuvrable quadruped locomotion can give a soccer playing robot a significant advantage on the field. If a robot can chase down the ball before its competitors and acquire a controlling positioning, the job for the opposition robots is made significantly harder. Locomotion has an important role in allowing the behaviours to reposition the robot in the environment.

Hornby, *et al*, describe three of the most common gaits used by quadruped robots, the crawl, trot and pace gaits.

- The *crawl* gait operates by moving each leg in turn lifting one at a time. This gait maintains static balance as it keeps the centre of gravity inside the triangle described by the three legs that touch the ground.
- The *trot* gait lifts the two diagonally opposite legs alternately. The robot balances along the diagonal joining the two feet that are on the ground. The Robot does not have static balance and will eventually fall onto a third leg.
- The *pace* gait lifts both legs on the same side of the body simultaneously, alternating between the sides. This gait requires the robot to dynamically shift its weight side to side in time with the legs. This is the least stable gait as it is easy for the robot to lose balance and fall over sideways.

Kimura, *et al*, evaluate the stability, maximum speed and energy consumption of the crawl (static), trot and pace gaits. The robot used is a quadruped known as ÒCollie-2Ó situated at the University of Tokyo.

Through experimentation, the authors find the crawl gait to be the most stable as it is the only gait with ÒStatic balanceÓOver all of the gaits tested, the authors conclude that they are all more stable with a faster period of motion.

An important conclusion made by the authors is that the maximum speed of quadruped gaits increases when the period is longer and the stride is longer. Eventually a limit is reach at which the walk will become unstable.

2.4 Behaviours

The objective of a soccer-playing robot is getting the ball into the opponent s goal. To do so, the robot must evaluate the current environment and determine the most appropriate action. Selecting the most appropriate action in robotic soccer is difficult because the environment is very dynamic and the robot may have limited information.

LRP-99 designed a set of soccer playing behaviours without a strong reliance on absolute localisation of the robot. The alternative adopted by LRP-99 was to rely on the positioning of objects relative to the robot, and adapt the robot s behaviour s accordingly.

As such, Individual behaviours utilise situation analysis to vary the way in which they control the robot. Every time a field landmark is identified it is held in a short-term memory, and the contents of this memory determine which variation of the behaviour will be executed. For example, if the robot is attacking the ball and beacons to the left of the opponent goal are identified, the robot will adjust its trajectory so that when it final gets to the ball, it is positioned to knock the ball towards the right.

Each of the behaviours within the robot were arranged into a simply hierarchy, with higher-level behaviours having priority over lower levels. At the top of the hierarchy was the getup routine, which should clearly pre-empt any other behaviour that was currently being performed.

CMTrio-99 designed a set of soccer playing strategies that improve and degrade with varying accuracy of localisation information. The strategies were designed to trade off the time wasted by stopping and localising against the benefits it can provide to the current action being performed.

The robot has different modes that operate with varying levels of efficiency based on the reliability of the localisation. When approaching the ball, if the localisation information is poor the robot will approach in a straight line, otherwise the localisation information is used to skew the robots path to help it get behind the ball.

When circling the ball, if the localisation is ÒusefulÓthe most efficient direction around the ball is evaluated, otherwise right is selected as default.

3 Colour Classification

The vision module receives one image at a time from the camera. There are three different resolutions available, namely, low, medium and high. Currently, we only use the medium resolution images containing 88˚x˚60 pixels. 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. 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, pink and green for the beacons, light green for the field carpet, dark red and blue for the robot uniforms.

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. This has necessitated an implementation that is as fast as possible to minimise the impact of this design decision on the overall performance of the robots.

Our vision system consists of two modules: an offline training module and onboard colour look up module. The offline software generates the colour table and stores it in a file. The onboard software reads the colour table from the file and uses it to classify each pixel in the input image. We first explain how the colour table is generated.

3.1 Offline Training Software

Because colour detection can be seriously affected by lighting conditions and other variations in the environment, we need a vision system that can be easily adapted to new surroundings. One way of doing this is by using a simple form of learning. This was originally developed by the 1999 team and extended in 2000.

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 Òcolouring inÓ the image by hand. All these pixels form the set of training data for the learning algorithm. Each pixel can be visualised as a point in a 3 dimensional space.

In 1999 team s software, all these pixels were projected onto one plane by simply ignoring the Y value. Pixels were then divide into different groups based on their manually classified colours. For each colour, a polygon that best fits the training data for this colour was automatically constructed. Therefore, every colour has one corresponding polygon. 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 4 shows a screen grab of the painting program used to manually classify the pixels and figure 6 shows another grab at the end of a run of the polygon growing algorithm. Figure 6 shows why we chose to use polygonal regions rather than the rectangles used by the colour lookup in the hardware. We believe that polygonal regions give greater colour classification accuracy.

For the 2000 competition, we kept the learning algorithm but changed the way it is used. Our conjecture was that by taking the Y values into consideration, we could

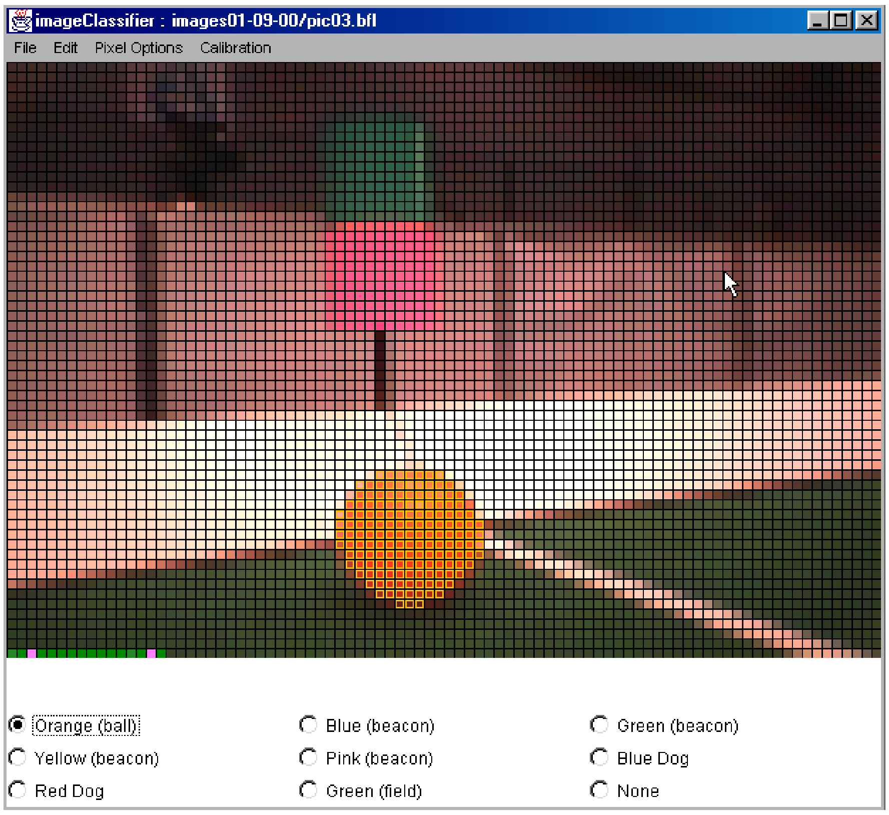


Figure 4 A painting program is used to manually classify pixels

improve the vision significantly. Figure 5 show all the pixels projected into one plane while figure 6 shows pixels in one plane based on Y values. By slicing the colour space into different brightness planes, the colours are more separated in UV space.

Initially, Y values were divided into 8 equally sized intervals. All pixels with Y values in the same interval belong to the same plane. Effectively we have 8 planes in total. 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 therefore chose a simple data structure: a set of two-dimensional arrays, where one dimension is U and the other is V. Each valid pair <U,V> is a coordinate of one element in the array. The value of the element is determined based on which polygons it lies in.

To determine the colour of an unseen pixel, the Y value is first examined to find the relevant plane. Only one bitwise operation is need for this. The next step is to use <U,V> to index into the array and the value of that element gives the colour.

With the addition of the Y planes, the vision system is now able to distinguish the orange colour of the ball from skin colour. This caused many difficulties in the 1999 competition when the arm or leg of a referee or member of the audience would sometimes be confused for the ball.

Discretisation of the Y values into eight equal intervals was a considerable improvement, however, the robots were still unable recognise the red and blue colours of robots. The reason is that those colours are very dark in the images we obtained

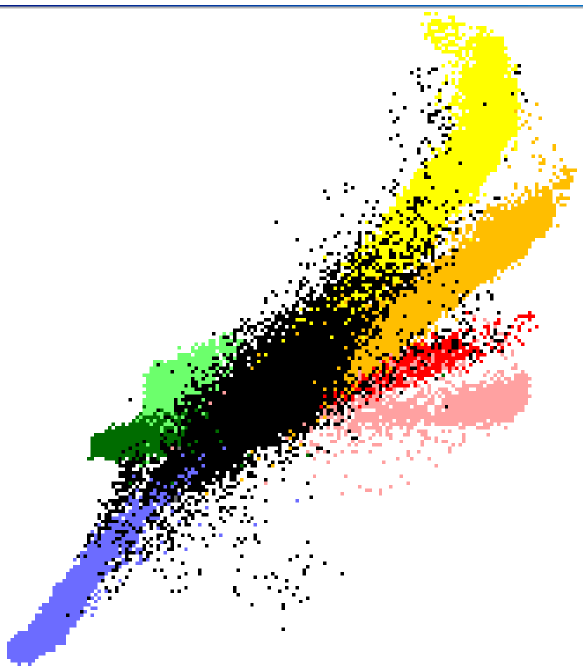


Figure 5 All pixels of the training samples with Y values ignored

from the robot camera. In the eight planes onto which we project the pixels, the red and blue pixels end up in the same plane as black pixels or other dark colour colours.

Being able to classify these colours is vital for robot recognition and consequently, team play, so a further refinement was attempted. The Y values of all pixels where examined to try to group them in such a way that dark red and blue pixels can be distinguished from black pixels. We did this manually and settled on 14 planes of unequal sizes.

In this configuration, we have more planes for lower Y values, reflecting the fact

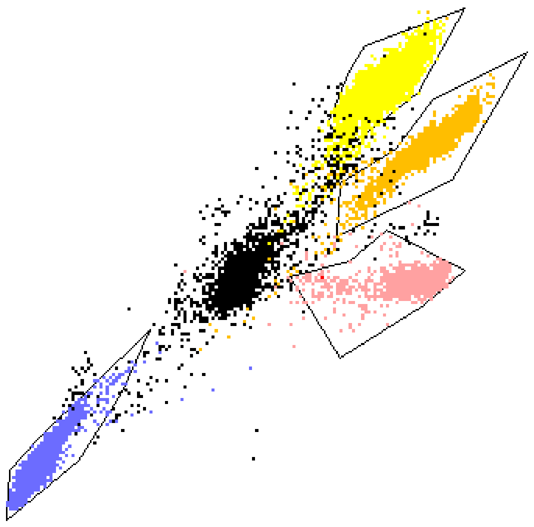


Figure 6 A polygon growing program automatically finds regions of pixels with the same colour in one plane

that dark colours are harder to separate. With these 14 planes, the robots can recognize the colour of the robot uniforms with reasonable accuracy. We believe that due to the limitations of the camera system and the choice of colours for the uniforms, more accurate recognition of other robots may require shape recognition, as well as colour detection.

The use of different planes based on Y values seems to generalise well. In an informal experiment, training images were taken with three floodlights turned on. The robots were tested with some lights turn off and with all lights turned off. Under the degraded conditions the robots could see the ball and other objects with smaller accuracy.

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. A classified pixel was coded as a byte with each of its bits indicating colour class membership in the eight colour classes. By ignoring pixels that occur in overlapping polygons, we removed the overlap. Each byte now only needs to encode one colour, allowing up to 256 separate colours to be specified. Removal of the ambiguity also improves classification accuracy.

3.2 Onboard Vision System

The vision system running onboard is rather simple since all the work has been done offline. The lookup tables for each of the fourteen planes are loaded into the robot s memory from the memory stick. One more lookup table is created to map the Y value into the appropriate plane. This operation is done once only at the start.

Each time an image is grabbed, we scan through every pixel and determine its colour by first looking at the Y value and finding to which plane the pixel belongs. The U, V values are then used as indexes to find the colour of the pixel in the appropriate lookup table. The output of this stage of processing is an image with all pixels labelled according to their colour. If a pixel does not fall into any of the colour classes it is left unclassified.

4 Object Recognition

Once colour classification is completed, the object recognition module takes over to identify the objects in the image. The possible objects are: the goals, the beacons, the ball and the blue and red robots. Four-connected colour blobs are formed first. Based on these blobs, we then identify the objects, along with and their confidence, distance, heading and elevation relative to the camera and the neck of the robot. The following subsections describe this process in detail.

4.1 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.

With the 1999 code, the camera frame rate is about nine frames/second and drops to six when the ball is close. We have found that the blob forming procedure is responsible for the slow-down and is the most time-consuming procedure in the decision making cycle. Therefore a faster blob-forming algorithm was developed. The new algorithm allows us to achieve a frame rate of about 26 frames/second most of the time and this drops to about 16 frames/second when the ball is close. The speed up in frame rate has a noticeable impact on the strategy and especially the head tracking routine.

The main idea in the algorithm is that in each row of a camera image, there are only a few different colours, so it is possible to construct a run of pixels of the same colour. We maintain a queue that stores only the right-most pixel for each such segment. Program 1 shows the pseudo-code for the algorithm.

As the colour detection module guarantees that each pixel is classified as only one colour, the outer loop to find unclassified pixels need only go through the image once. This contributes to the speed of the algorithm.

4.2 Object Identification

Objects are identified in the order: beacons first, then goals, the ball and finally the robots. The reason the ball is not detected first is because we use some sanity checks for the ball based on the location of the beacons and goals. We developed software that displays what this module gets from the vision module and what objects the robot sees.

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 useful parameters. The bounding box determines the size of each blob and once it has been found we can compute the object s distance, heading and elevation.

```
procedure rightmost(x,y)
  find the right most pixel of the segment in line x
  containing  consecutive pixels of the same colour
  including (x,y)
end rightmost;

initialise all pixels unmarked;
foreach pixel (x,y)
  if (x,y) is unmarked and is colour c
     initialise empty Queue;
     rightmost(x,y) à  Queue;
     while Queue is not empty
        (p,q) ß  Queue;
        if (p-1,q) is colour c
           rightmost(p-1,q) à  Queue;
        if (p+1,q) is colour c
           rightmost(p+1,q) à  Queue;
        mark (p,q) belonging to this blob
        foreach (i,j) of colour c in segment of (p,q)
        if (i,j) <> (p,q)
           mark (i,j) belonging to this blob;
           if (i-1,j)  is colour c and (i-1,j)=rightmost(i-1,j)
              (i-1,j)à  Queue;
           if (i+1,j) is colour c and (i+1,j)=rightmost(i+1,j)
                                        (i+1,j)à  Queue;
```

Program 1. Blob Formation

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. We also calculate the level of confidence in the identification of the object. Since this is only used in localisation, we will discuss it further in Section 5.

Up to this point, distances, headings, etc have been calculated 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 (Dalgliesh & Lawther, 1999).

In the 1999 team software, this transformation was done when the world model was being constructed. In 2000, this processing was moved to the earlier stage of object identification because there are some sanity checks that can only be done after the 3D transformation. For example, suppose that object identification finds two orange blobs and decides one of them is the ball and passes it to the World Model module. If the World Model module performs the 3D transformation and it finds that the elevation of the ball relative to the robot neck is too high, says greater than 30ß, it discards this object as being in an impossible position. Because this check is done at a higher level, we cannot go back to the object identification module to try the other orange blob as

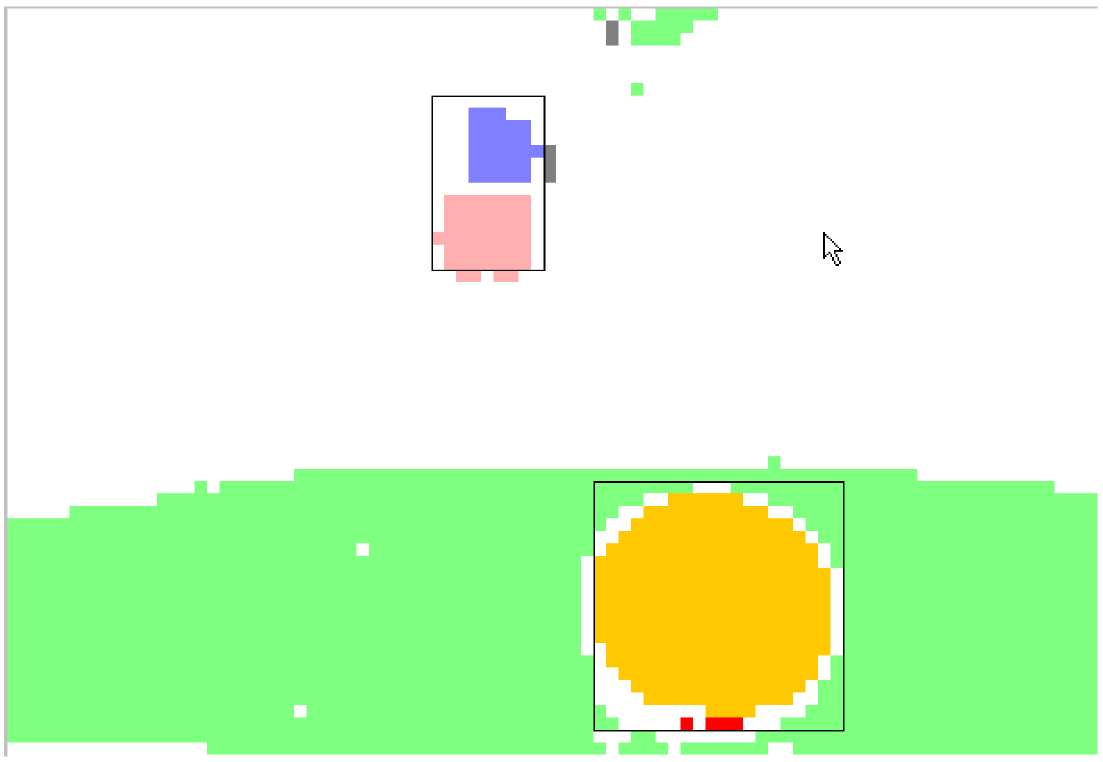


Figure 7 Colour classification and object recognition from the robot s camera

the ball. Consequently, the robot does not see a ball in that image even though there might be one.

4.2.1 Beacon Identification

Every beacon is a combination of a pink blob directly above or below a green, blue or yellow blob. One side of the field has the pink on the top of the colour in the beacons, while the other has it below. The beacons are detected by examining each pink blob and combining it with the closest blob of blue, yellow or green. This simple strategy was used with reasonable success in 1999, but was found to fail occasionally. 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.

4.2.2 Goal Identification

After the beacons are 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 or the goal cannot be to the left of left side beacons nor to the right of right side beacons.

During development, it was sometimes noticed the robot would try to kick the ball into a corner. The cause was the robot seeing only the lower blue part of the beacon in the corner and identifying that as the goal. When the robot is near the corner and only sees part of the blue half of the beacon, the blue blob may appear large, having its width roughly twice as long as its height, which matches a feature of the goal. To avoid this misidentification, we require the goal to be directly on top of the green of the field.

4.2.3 Ball Identification

Despite the use of polygonal colour regions, it is still possible that pixels will be misclassified and false orange blobs may appear. The ball is found by looking at each orange blob in decreasing order of bounding box size. The first orange blob that satisfies all of the following tests is deemed to be the ball.

A common source of misclassifications for the ball occurs when orange objects are present in the background. The first test is for the elevation of the ball relative to the robot s neck to be less than 20ß. The elevation of the ball must also be lower than that of all detected beacons and goals in the same image. This test is effective since any orange objects in the background will appear higher than the beacons or goals.

With the above tests, ball detection is reliable until we put red robots into the field in front of a yellow goal. This caused frequent problems for most teams during the competition. This seems to be because, when the camera is moving, pixels are blurred and that causes the combination of colours. Red and yellow combine to form orange.

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 is not at the bottom of the screen, it must be on top of 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.

4.3 Robot Recognition Algorithm (RRA)

The Robot Recognition Algorithm (RRA) used at RoboCup 2000 uses a combination of visual and infra-red sensors to identify the presence of a robot in the visual field, and to approximate the distance from the camera to the object.

The RRA is designed to feed information to higher-level strategies that control obstacle avoidance. The recognition has the following properties.

1. A maximum of one robot from each team is identified in each frame. For the purposes of obstacle avoidance, important frames generally don t contain multiple robots.
2. The algorithm favours over-estimation of distance as opposed to under-estimation. This approach was taken based on the belief that colliding with the occasional robot would be better than triggering spurious avoidances.

4.3.1 Inputs to RRA

Infra-Red Distance Measurement

Table 1. Sample Blob Classification Information

Blob ID	Area	Min_X	Min_Y	Max_X	Max_Y
1	541	7	11	39	34
2	125	39	24	53	38
3	71	5	1	16	9
4	9	58	20	60	25
5	3	15	2	16	3
6	2	12	9	13	9
7	1	14	1	14	1

The on-board IR sensor feeds accurate distance information for any obstacle aligned directly in front of the head of the robot at a distance between 10-80 cm. 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 15ß. The calculation is:

```
double headpitch = sensor[0]/1000000.0;
double headyaw = sensor[1]/1000000.0;
double effectivepitch = headpitch * cos(headYaw);
if degrees(effectivepitch) < -15
     range2obstacle_ = 1501;
```

Another source of inaccuracy is the time delay between the IR and camera readings. Connecting to the ÒmPSDÓ Sony oblet gives IR information that is delayed by approximately 0.5 seconds, causing problems during fast head movements. Obtaining IR data directly from ÒOVRcommÓ provided far better synchronisation between the IR and Camera.

Blob Classification Information

This information is obtained after the camera image has been captured, colour classification performed, and adjacent pixels connected into blobs. The RRA only uses blob information specific to the uniform colours of the two teams. See Table 1.

4.3.2 Design of RRA

The initial RRA design was based upon 25 sample images of robots taken from the robots camera, as well as manually measured distances to each of the robots in the samples. The colour detection and blob formation algorithms were run over the sample images and blob information obtained. Noise blobs were discarded (<10 pixels) and the following two values calculated:

1. Total Pixels in Blobs
2. Avg Pixels per Blob

From this, a curve consisting of logarithmic and polynomial components was manually fitted to the sample data. The base fitting equation is:

$$visualDist = A \times \left[\log\left(\frac{B}{AvgPixels} \right) \right]^C$$

A, *B* and *C* were manually adjusted to fit the curve adequately.

Using the above equation, a distance approximation was derived based purely on the feedback from the camera. The IR reading was compared to the *visualDist* and would be used instead of the *visualDist*, if:

1. IR reading shorter than VisualDist but not by more than 15cm.
2. IR reading shorter than 25cm and abs(heading) of robot < 20ß (relative to camera).

These criteria aim to use the IR when a robot appears to be within its range and utilises the accuracy of the IR reading at short range. Once the distance approximation has been finalised several sanity check are employed to filter out 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).

Compared to the 1999 uniforms the 2000 uniforms had the following characteristics.

1. More total patch area. (Increases range of detection)
2. Increased variation between patch sizes. (Reduces overall reliability at short range)
3. Patch edges are close/touching, giving unpredictable connectivity in blob formation algorithm. (Induces an element of randomness from frame to frame as multiple patches are detected as single patch)

Two modifications were made to adapt to year 2000 uniforms.

1. Change from a fixed 10-pixel noise floor to a variable noise floor based on size of largest patch. (Noise floor = Largest Patch / N)
2. When a large patch is identified (compared to smallest patch > noise floor) it was treated as if it was multiple patches.

This resulted in three new variables in the RRA bringing the total to six. To tune the algorithm accurately 50 new sample images were taken and the distance and blob information fed into a Java program. The Java program explored a constrained part of the parameter space by first breaking the parameter space up coarsely, then exploring the parameter space more finely around the best parameter set that had been identified. The algorithm attempted to find a solution for the six variables that resulted in the lowest cumulative distance error from the set of manually measured readings.

4.3.3 Effectiveness

Unfortunately, the accuracy of the distance approximated by the RRA degraded significantly with the transition from the 1999 to the 2000 uniforms. However, using the infrared sensor at short ranges still allowed the algorithm to identify situations when there is a risk of collision. The algorithm fulfilled its intended purpose for RoboCup 2000 but would not be a strong basis for more advanced strategies requiring knowledge of surrounding robots.

4.3.4 Weaknesses

The primary weakness of the RRA is its absolute reliance on accurate colour classification. The algorithm does not adapt well to background noise that will cause it to often 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.

5 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 relative to the robot s neck. Localisation tries to determine where the robot is on the field and where the others objects are. 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. That is a job for the future.

5.1 Robot Localisation

The world model maintains three variables: the x and y coordinates of the robot and it s heading. The left-hand corner of the team s own goal end is the origin, with the x-axis going through the goal mouth. The own goal of the robot is set by the higher-level strategy code. Strategies refer to Òown goalÓand Òtarget goalÓrather than Òblue goalÓand Òyellow goalÓ

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 a time, 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 (Dalgliesh & Lawther, 1999). 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, that is, when the odometry from the action module is not zero. We could maintain the array and move the objects in the array relative to the robot. However this type of calculation tends to give large errors over time, so this idea was abandoned.

The world model also receives feedback from the locomotion/action module, *Pwalk*, to adjust the robot s position. The feedback is in the form (*dx*, *dy*, *dh*) where *dx* and *dy* are the distances, in centimetres, that the robot is estimated to have moved in the *x* and *y* directions and *dh* is the number of degrees through which the robot is estimated to have turned. We try to get feedback as quickly as possible, which is about every 1/25 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. Another problem occurs when the robot gets stuck, for example, by being blocked by another robot, *Pwalk* is not aware of this and keeps feeding wrong information to the world model.

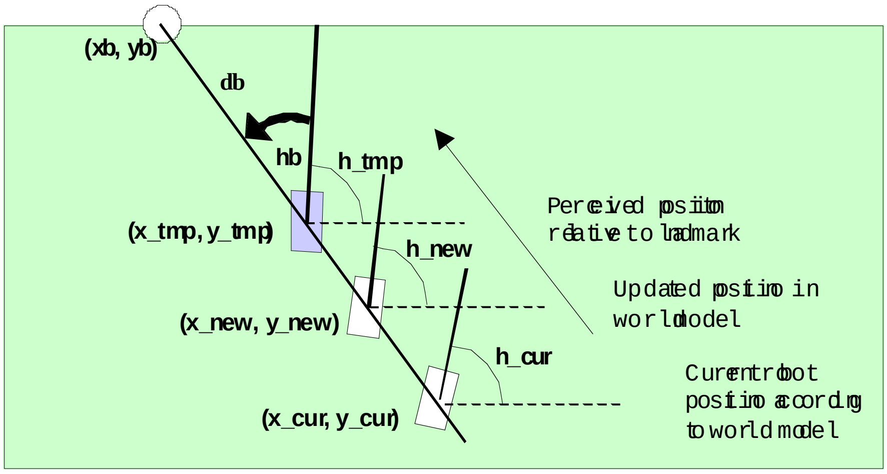

(xb, yb)
db
hb
h_tmp
(x_tmp, y_tmp)
h_new
(x_new, y_new)
h_cur
(x_cur, y_cur)
Perceived position relative to landmark
Updated position in world model
Current robot position according to world model

Figure 8 Update of position based on single beacon

The methods described above cannot provide reliable localisation because the robot usually sees only one landmark in an image and the robot is moving constantly. We therefore tried to devise a method for updating the robot s position using only one landmark. Suppose the landmark has coordinates (*xb,yb*) and its distance and heading relative to the robot are *db* and *hb* respectively, as shown in Figure 8. We draw a line between the (*xb,yb*) and the estimated current position of the robot in the world model, (*x_cur,y_cur*). The robot s perceived position, relative to the landmark, is the point on that line *d cm* away from (*xb,yb*) and on the same side as the current robot position, in the world model. The localisation algorithm works by Ònudging Óthe estimated position in the world model towards the perceived position relative to the landmark.

With a camera frame rate of about 26 frames/second, this algorithm converges quite quickly and accurately. 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 based on that landmark. Because we use a low trot gait, the robot can see goals and beacons most of the time, if other robots do not obscure them.

One problem remains due to the perception of landmarks. The beacons are good landmarks because they are relatively small. When visible in an image, distance and heading measurements to a beacon are reasonably accurate. However, this is not the case with goals. 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, using only the beacons for localisation. Near a goal, the beacons are often difficult to see and the odometry problem reappears if the robot is blocked by another robot or trapped in the goal.

Near a goal, the heading of the goal is used to update the robot s heading. However, the robot s (*x*, *y*) position is not updated because we deem the measurement of distance to the goal too unreliable. Thus, the goal is never used in triangulation. The heading to the goal is quite accurate when the robot is far from goal. When the goal occupies the whole image the robot must be very close to goal, so all that matters is that the robot is facing the goal.

A confidence factor is associated with each dynamic object in the world model. This is used when incorporating new data into the existing model. The

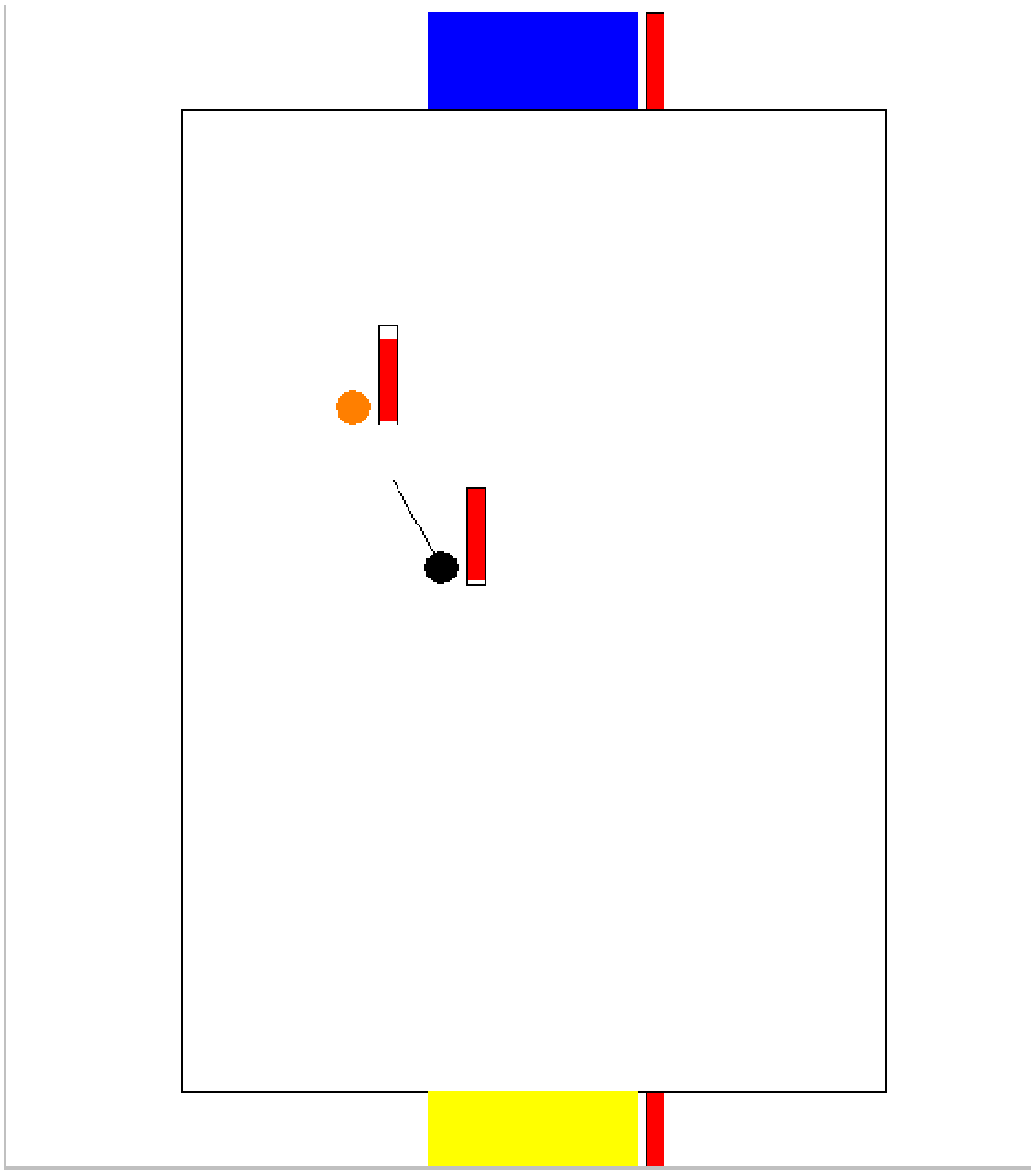


Figure 9: Display of world model

confidence factors were devised by the 1999 team and were used extensively before the one beacon update approach was developed. Previously, only triangulation was used and therefore it was necessary to store information about an object over a long period of time. To avoid the problem of error accumulation, the confidence factors of the objects were decayed after every camera frame. With the new approach and the high camera frame rate, localisation converges to an accurate world model very quickly. Therefore, the confidence factors are of less importance than they used to be.

5.2 Updating the RobotÕs Position

In this section, we give details of the method by which the robot s estimated position in the world model is updated. As described in the previous section, the position is updated by ÒnudgingÓ x, y, and heading variables in the direction of the values as perceived by the robot relative to a landmark, such as a beacon. The update may be conservative, giving more weight to the current position, or it may be more aggressive, weighting the perceived position more. The degree of conservatism is controlled by a *rate* parameter. At present, this is computed as follows:

$$rate = \begin{matrix} 0.4 & \text{if } curCF > 600 \\ 1 & \text{otherwise} \end{matrix}$$

where *curCF* is the current confidence factor for the position of the robot.

We compute the robot s new *x* position as:

$$x\_new = x\_cur \times CurrentRate + x\_tmp \times BeaconRate$$

where

$$BeaconRate = \frac{BeaconCF \times rate}{BeaconCF \times CurrentCF}$$

and

$$CurrentRate = 1 - BeaconRate$$

The new *y* position is updated similarly. Updating the heading is slightly more complicated since the heading is a circular measure that is always in the range –180ß.

```
if cur_h —h_tmp >= 180
        h_tmp += 360
else if h_tmp-cur_h >= 180
        cur_h += 360
new_h = cur_h * cur_rate + h_tmp * BeaconRate
if new_h>180
        new_h -= 360
if new_H <= -180
        new_h += 360
```

5.3 Ball, Beacon and Goal Localisation

All objects in the world model are stored with a tuple <*heading, distance, confidence*>. *Distance* and *heading* are relative to the position of the robot in the World Model. Because beacons and goals are fixed, we can work out the distance and heading by simple geometric calculations.

The ball is stored internally as a pair of (*x*, *y*) coordinates. We choose this representation because when the robot is moving without seeing the ball we want the ball to stay at the same position in the World Model. If the ball is seen in this camera frame, its information and the new robot position are used to calculate the ball new position. The old and new positions of the ball are combined in a same way as the robot position to get the final ball position. Using this new ball position, we calculate the distance and heading relative to the robot for output of World Model.

We have already mentioned that several confidence factors are used in the localisation algorithm. A beacons confidence factor is computed as follows:

$$BeaconCF = \frac{1000 \times mass}{2 \times d^2}$$

where *mass* is the total number of pixels in both blobs and the beacon confidence is not allowed to exceed 1000.

The confidence in the location of the ball is given by:

$$BallCF = \frac{1000 \times mass}{\pi \times \frac{w}{2} \times \frac{h}{2}}$$

where *mass* is the total number of pixels and *w* and *h* are, respectively, the width and height of the bounding box.

The confidence in the location of the goal is computed as:

$$GoalCF = \frac{1000 \times mass}{w \times h}$$

where *mass* is the total number of pixels and *w* and *h* are, respectively, the width and height of the bounding box.

Since the goal may be obstructed, we make some guesses about *w* and *h*. If the goal is unobstructed, we expect roughly the height of the goal to be half the width. If this is not the case, we assume some obstruction. If the height is smaller than half the width then we extend the height. Otherwise, we extend the width: if the blue blob abuts an edge of the image, it is extended in the direction of that edge.

6 Action/Locomotion

The objective of the action module is to move the head and legs in response to commands from the behaviour module. For example we may wish to tilt the head up 1 degree, pan it right 2 degrees while simultaneously walking forward taking 4 cm sized steps. All head and leg commands are executed by invoking the makeParaWalk method, which was designed to activate all actions by passing parameters to class PWalk. Variables included in the method call are the parameters that control the movement and stance of the robot. To illustrate,

```
makeParaWalk(0,4,0,0,65,73,97,16,19,33,20,-35, 20,1,1,2,0);
```

would initiate a forward cantered trot walk taking 4cm steps each of 0.52 seconds duration, lifting the front legs 16mm off the ground and the back legs 19mm off the ground. The robot s shoulder joints would be 73mm from the ground and hip joints 97mm, etc. The head will move 1 degree up and pan right 2 degrees from its current position. (The parameters are explained in appendix PWalk Documentation)

Head and leg commands are given concurrently and execute concurrently. We will now explain them one at a time.

6.1 Head Movements

Head movements include tilt (up/down), pan (left/right), roll or sideways tilt (left/right) and mouth position. The roll of the head was not used and the motor value always set to zero. The head motors can be moved relative to their current position or to an absolute location. The last 4 parameters in makeParaWalk specify these head parameters. (See appendix PWalk Documentation)

In the above method call, the last 4 parameters are 1,1,2,0 and signify moving the head relative to its current position by tilting 1 degree up and panning 2 degrees right. The mouth is closed.

6.2 Leg Movements

6.2.1 Quadruped Gaits

We experimented with the crawl, trot and pace gaits. We decided against the pace gait as it was difficult to synchronise the weight transfer with the leg movements and it easily became unstable. The crawl gait was stable but proved to be rather slow in comparison with the trot gait. We therefore decided on the trot gait for the competition.

6.2.2 Design Objectives

The three primary design objectives were to:

1. Drive the robot as if controlled by a joystick with three degrees of freedom, namely: forward or backward, sideways left or right and to turn on the spot clockwise or counterclockwise. This requirement suggested itself because of the need to continuously react to environmental changes. Previous fixed walking

styles were not responsive enough and required unstable transitions. Having 3 key parameters controlling the motion allows complex movements to be derived geometrically, which greatly simplifies strategy development.
2. Move the robot over the ground at a constant speed. A constant velocity would reduce the strain on the robot motors by not accelerating and decelerating the body. It would allow a faster walking pace to be achieved after gaining momentum and assist with the third design objective.
3. Keep the camera as steady as possible. The camera is located in the head of the robot. We observed that images from the robot's camera showed wildly erratic movements due to the robots head and leg motions using previous walks. We postulated that a steadier stream of images would assist in object tracking. When approaching a ball, for example, it is desirable not lose visual contact with it.

The solution adopted was to move the paws of the robot's feet around a rectangular locus. 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 to the locus describe the path used to lift the paw back ready for it to take the next step. The legs that are touching the ground exert the forces that move the robot. In the trot gait 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 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 looses 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.

6.2.3 Control

We will 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 inclination of this plane relative to the sides of the robot we can determine whether the robot moves forward/backwards or sideways. For example if the locus plane is parallel to the robot sides, the robot will move forwards or backwards. Whether the robot moves forward or backward is determined by the direction in which the paw moves around the locus. It helps to image that the paw moving around the locus acts similarly to a rotating wheel. If we look at the robot side on and it is facing left, then if the paws move in an anti-clockwise direction around the rectangular locus the robot will move forward. If the paws are made to move clockwise the robot will walk backwards.

If we incline the locus plane perpendicular to the sides of the robot it will move either left or right in a similar fashion to the way it moves forward/backward. An inclination of the locus plan at another angle will cause the robot to move both forward or backward and left or right calculated by adding the vectors of both components.

So far we have assumed that all the locus planes are kept parallel to each other and all the paws move either clockwise or counter clockwise. Before we move on to explain how the robot is made to turn, we should also note that the width of the rectangular locus and the speed at which it is traversed by the paw determines the speed at which the robot moves. There are limitations to these parameters and we will see later how they were tuned to achieve acceptable performance.

How can the robot be made to turn? If we take a plan view of the robot and describe a circle that passes through each of the upper leg joints, then inclining the locus planes tangentially at each joint will turn the robot clockwise or anti-clockwise.

Again, it is possible to combine components of each of the three movement dimensions creating complex waltz like movements in which the robot moves forward, sideways and turns all simultaneously.

6.2.4 Parameterised Walk

The behaviour module activates leg movements by calling method makeParaWalk. Parameters are passed to PWalk that specify control and stance information. The robot stance is determined by the spread of the legs and the height of the body both at the front of the robot and at the back. PWalk first calculates the paw locus for each leg and converts this to leg joint positions using inverse kinematics. The geometric equations to calculate the locus and perform the inverese kinematics are derived as shown in appendix PWalk Documentation.

Each time the makeParaWalk method call is accepted it commits the robot to move each paw half of the rectangular locus from the middle of the top (home) to the middle of the bottom (home) position, or vice versa, depending on which diagonally opposite legs are raised. This half step typically has duration of the order of 0.5 seconds. The advantage is that the speed and direction of movement can be changed at the strategy level 2 times per second making the robot very responsive. Because of the common underlying mechanism, gradual changes to control parameters cause smooth transitions between walk directions.

There is one parameter that determines the speed at which the paws move around the rectangular locus. This corresponds to a faster rotation of a wheel in our analogy. Together with the control parameters setting the stride length, this will determine the speed at which the robot moves over the ground. The power limitations of the leg motors and the physical dimensions of the robot limit trot speed and stride length respectively. The question then arises as to which control, speed and stance parameter values give the best performance. For example, how to walk forward in a straight line at the fastest speed.

6.2.5 Search in parameter space

There are three control parameters, one speed parameter and 8 stance parameters that influence the leg movements for a particular walk style. (Described in appendix PWalk Documentation). A machine learning solution to finding the best speed and stance parameter setting suggested itself. We could perform a gradient ascent on the performance measure (for example forward speed) incrementally adjusting the parameters over many test runs. The problem with automating this approach was that it would take considerable time and resources to set up the learning environment. Of concern was the wear and tear on the robot motors and leg joints given the long periods of training required. Hornby, et al, report continuous 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. These considerations include:

1. The home position of the legs needs to be adjusted so that the robot will dynamically balance on the two legs that touch the ground.
2. A longer stride required the robot body to be closer to the ground to allow the legs to reach further.
3. The front and back legs do not interfere with each other during the cycle of the gait.
4. The legs needed some sideways spread to allow the robot to move its legs laterally so it could walk sideways.

Besides the ground speed, we would also judge the reliability of the walk and the deviation from the intended path as a part of the performance measure. In this way we found that there was a trade off between the frequency of the gait and the length of the stride. Maximum ground speed was achieved with a longer stride and a slower gait frequency. This manual approach used about 10 minutes of robot running time and only about 12 iterations of parameter adjustments to find a good performance setting resulting in the characteristic low forward leaning stance of the UNSW robots.

A further refinement that increased ground speed considerably was the introduction of a canter action. The canter action sinusoidally raises and lowers the robot s body by 10mm synchronised with the trot cycle. The parameters could then be manually tuned so that the robot was able to reach speeds of 1200cm/min. This compares to 900cm/min achieved using a genetic algorithm approach (Hornby et al 2000). The camera is not as steady in this type of walk because of the additional canter movement.

6.3 Application

Head and Walk parameters are varied dynamically allowing interesting skills to be developed in the behaviour module such as ball tracking, ball circling and walking towards an object. To track the ball, for example, we would take the horizontal and vertical displacement of the ball from the centre of the camera image and use parameters proportional to these measures to drive the head movement relatively. In this way, the head can be made to move to keep the ball in the centre to the image. As another example, to make the robot walk towards the ball in a straight line, we first track the ball as above and then use the degree of twist in the neck from straight ahead to feed the turn parameter of the walk. This will ensure that the robot will turn towards the ball if it should veer to either side as it walks towards it. These types of applications of the action module form the skills that are more fully described in the section on behaviours.

6.4 Kicking

The kick was developed as a part of the action module where it logically fits in the overall architecture. It also allowed for easy integration with the walk. The design objectives were for an accurate, easy to line up and powerful kick. 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 ball best met the objectives.

In order for the kicking action 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 manually by conducting many trials and adjusting them. The final kick was a compromise between power and reliability as the more powerful kicks had lower success rates. The kick was hard coded as walk type 2.

The skills to position the ball for kicking were developed in the behaviour module. The robot would walk up to the ball at a fast speed keeping the head above the top of the ball. It would then trap the ball under the head, producing repeatable ball positioning. The kick could then be triggered with the ball in a controlled starting position. Because of the stochastic nature of the real robot environment the kick was effective about 80% of the time and less in play when opponents effected ball positioning or blocked the ball from being kicked at all.

6.5 Interface to Effectors (Double Buffering)

All the motors positions are sent to the effectors using the Sony double buffering scheme. (See Sony Open-R Software Tutorial Chapter 5 Servo and Example 3.2). The robot motors accept 125 motor positions from the effector per second. PWalk sends 5 effector frames per buffer. This number of frames was calculated to commit an action for the minimum duration consistent with the camera frame rate. Since we are able to process slightly more than 25 camera frames and hence generate 25 action decisions per second, we do not want to commit actions for any longer than 1/25 or 0.04 seconds. As each effector frame is processed and executed by the robot in 0.008 seconds, each buffer needs to be filled with 0.04/0.008 frames (ie 5). All motor positions are re-calculated for each 0.008 seconds of movement.

If the camera frame rate drops below 25 per second, which is possible when presented with very complex images, the buffers may both be emptied and slight delays may be introduced in the actions. This is more prevalent in head movements that rely solely on makeParaWalk method calls. Leg movements are committed for half a locus cycle as described above. While executing this cycle PWalk is not subject to camera delays because it effectively calls itself until completion of the half step.

7 Shared Skills

This chapter describes skills that are shared by the forward and the goalie.

7.1 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. The target and rotation directions are decided elsewhere at the strategy level.

This circling technique is unique because it involves no aggressive transitions in the robots movement, maintains constant visual contact with the ball, and keeps the robot s body pointing as close as possible to the ball.

The skill only ever evaluates the parameters for the next step and never forces the robot to perform a sequence of pre-calculated steps. Because of this the robot can constantly re-evaluate its path and maintain correct positioning.

Ball circling is based upon two geometrically derived points calculated using the field and ball positions in the world model. The target point is the final intended destination and the circling point deflects the path around the ball if required. See Figure 10.

To perform the skill, the robot is simply driven towards the closer of the circle and target point (using forward and left parameters), while the body is oriented towards the ball (using turn parameter).

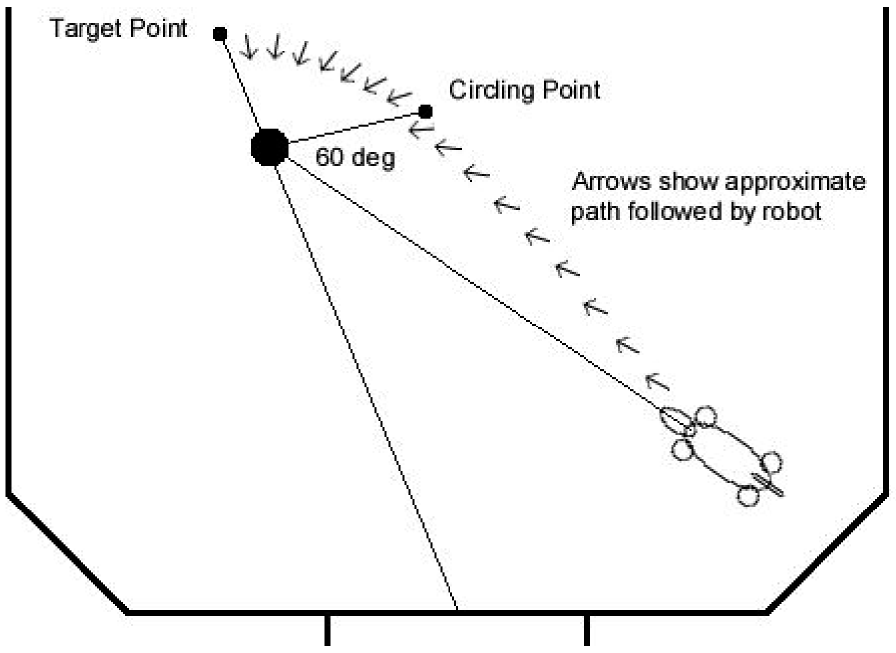


Figure 10 GetBehindBall Skill

7.2 KickBall

When setting up to kick the ball, the robot will approach at approximately 80% of maximum speed and must maintain a heading to the ball between –15ß. The robot will only track the ball s movement with the head pan while the head tilt is held constant such that the head will just clear the ball.

Upon losing vision of the ball, the robot will transition into a very low stance with the head 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, the kick will be aborted.

The robot can then take a limited number of rotational steps to align the goal (set at 2 complete steps for Robocup 2000) before triggering the kicking motion. The goalie and striker will also use these rotational steps to try and kick the ball away from obstacles.

7.3 DribbleBall

The DribbleBall skill is designed to move the ball forward by bumping it between the front legs of the robot as it walks. This creates a fairly controlled movement of the ball but the robot does have to periodically lose vision of the ball, making it sometimes risky.

With the dribble, the robot approaches the ball with the same constraints as the kickball skill. A minor difference is that the mouth is held partially open such that the ball will just bump it upwards a small fraction.

Once the robot loses sight of the ball, the robot continues to walk forward blindly. A counter is started which will trigger the finish of the skill, and a knock on the mouth (hopefully from the ball) will delay the counter slightly.

Once the counter triggers the end of the dribble, the dribble is deemed finished if the robot can see the ball. The controlling strategy may then chose to re-initialise the dribbleBall skill, and the dribble will continue seamlessly.

If the ball cannot be seen the robot will stop and take a few steps backward. The assumption here it that the ball is either directly under to head or has rolled slightly to the side, whereby backing up will bring it into view as quickly as possible. Once the ball is found or the backward steps completed, the controlling strategy must decide the next skill to be executed.

7.4 BuntBall

The Bunting Strategy is very simple in that the robot essentially walks directly at the ball with full range head tracking enabled. The focus of this skill is speed and it is not designed for precision.

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). For example, if the target (generally the goal) is 45 degrees to the right, a small component of left movement is induced onto the robot as it approaches the ball. This is to help achieve the ideal alignment situation when the robot finally approaches the ball and hits it. An added benefit of the sidewards component is that the robot will also circle around the ball to a small extent as it approaches, helping to get the robot directly behind the ball before it is reached.

7.5 TrackBall

One of the most critical skills for any soccer-playing robot is to keep visual contact with the ball. Losing track of the ball wastes significant amounts of time and can put the team at a significant speed disadvantage.

The two primary control objectives are: -

1. Keeping the head as steady as possible.
2. Making the head react quickly to fast transitions in the balls movement.

The TrackBall skill only controls the head of the robot. As such, this skill is always run concurrently with another skill that controls the robots walking.

The head-tracking formula takes as input the heading, elevation and distance of the ball (relative to the camera), as well as the sensor readings for the pan and tilt motors in the robots neck.

Using this data the formula resolves the correction components for the tilt and pan motors that will align the ball at the direct centre of the visual field, or as close as possible if beyond the heads range of motion.

When the ball is a long way from the centre of the visual field the head is moved by 80% — 120% of the correction component. The absolute motor values are set and the head moves very aggressively to the target.

When the ball is close to the centre of the visual field the head is only moved 20-40% of the correction component but this is set as a relative movement from the last motor target position. This generates a slow and smooth movement of the head.

The formula solves the correction components as follows:-

1. Uses heading, elevation and distance of the ball to generate a position ÒBÓ(x,y,z) in 3D space relative to the camera. X is horizontal in camera, Y is vertical in camera, Z is depth.
2. Translates this position ÒBÓforwards and upwards by the length of the face and neck respectively.
3. Rotates this position ÒBÓin the XZ plane to correct for the heads current pan motor position.
4. Rotates this position ÒBÓin the YZ plane to correct for the motors tilt motor position.
5. Clips this position ÒBÓto allow for limitations in the heads range of motion. The ball has to be at least 4cm(length of neck) in front of the robot to position it directly in the centre of the camera image.
6. Solves the tilt component required in the YZ plane to align ÒBÓ,compensating for the effect of the neck length.
7. Rotate in the YZ plane to align ÒBÓon the XZ plane.
8. Solves the pan component required in the XZ plane.

7.6 ScanForBall

The ScanForBall skill drives the head in a rectangular path searching for the ball. The path consists of one upper scan and one lower scan. The head is moved in a direction such that if it hits the ball in the lower scan, the ball will roll in the direction of the target goal. For example, if the target goal is on the robot s left the head will move clockwise, as this generates a leftward movement of the head in the lower scan.

The controlling strategy sets up two data structures that designate the coordinates of the corners of the rectangle, as well as the transition speeds along the sides of the rectangle.

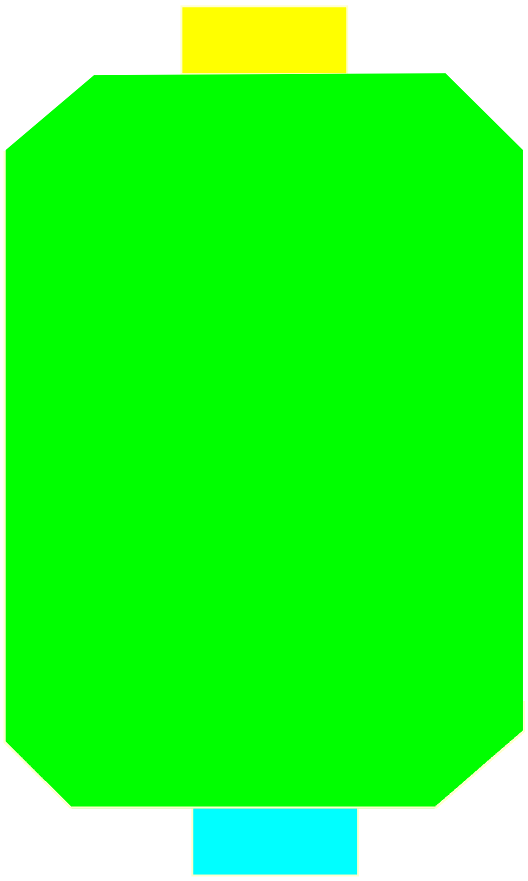
8 Strategies

Each team is allowed three players on the field. The UNSW team plays two Forwards and a Goalkeeper. Each role has its own set of strategies, which are described in this section. Over the course of the year, many strategies were implemented by the members of the team then later discarded. Only the final strategies used at Robocup 2000 will be described.

8.1 The Forward

The basic strategy is simple: the robot tries to align itself behind the ball, and then selects an appropriate attacking skill to move the ball towards the target goal. However, the details of the strategy are modified by the robot s location on the field. The field is divided into regions as shown in Figure 11.

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*. Each time the Forward module is invoked, only one skill is chosen and only one action is returned. *Backup*, *findBall*, *getBehindBall* and *chargeBall* are specific to the Forward strategy. We will now explain each of these skills.

8.1.1 Backup

When a robot sees one of its teammates nearby, it is a good idea to avoid bumping it. This strategy reduces the chances of our robots interfering with each other. One example is when robot 1 is dribbling the ball, its teammate, robot 2, sees the ball and starts running at the ball. But when robot 2 gets close to robot 1 it backs up, giving way to robot 1. This behaviour also reduces the occurrence of rugby-type ÒscrumsÓ when a pack of robots all fight for the ball, but generally just get in each other s way. The backup behaviour tends to keep one robot on the ÒwingÓ of its teammate, which effectively makes one robot wait outside a scrum for the ball to pop out. Once that happens, the ÒwingÓ can attack the ball in clear space.

One problem that may occur is when two robots go for the ball, see each other and start backing up at the same time, leaving the ball untouched. To prevent this situation, when a robot is less than 20 centimetres from the ball it ignores other robots.

The backup skill works by taking a few steps in the opposite direction to an excessively close team mate. For example, if the team mate is on the right, it walks sideways to the left.

The backup skill has a significant importance to the UNSW strikers because their wide walking stance and aggressive nature can cause huge amounts of interference. Extensive interference can cause the robots to overlap their legs and lock together, or can result in a robot falling over forwards onto its head. Locking of the legs greatly reduces the effectiveness of the strikers and they can take a long time to separate.

8.1.2 FindBall

The findBall skill iteratively performs the *scanForBall* skill, followed by a 45ß turn of the robot s body. The direction of the turn is chosen so that if the robot accidentally hits the ball, it will hit the ball towards the target goal. If the target goal heading relative to the robot is positive then the direction of the turn is clockwise otherwise it is anti-clockwise. This decision is made once for the first turn. Subsequent turns in the execution of the same skill are all in the same direction.

The robot continues alternately turning and scanning the head until it finds the ball or it has made six turning moves. When it has turned 45ß six times without seeing the ball, it is likely that the ball is obstructed or outside its range of vision. The robot then starts going to a defensive position defined to be at coordinates (50,100). At this position the robot should be able to see an unobstructed ball anywhere on the field. The robot then spins on the spot constantly at the defensive position. It exits the *findBall* skill once it sees the ball.

8.1.3 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 opponents 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 6 before it runs at the ball. There are two skills that the robot can use to charge the ball namely *dribbleBall* and *buntBall*.

Dribbling is invoked when the robot is facing the opponents half, the ball is close and the ball heading is less than 15ß off target. If the ball is not in position to dribble, the robot will bunt the ball with the head. Even though bunting is not accurate, we only need to knock the ball to the other half.

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

8.1.4 GetBehindBall

The Forward decides target point and the direction to turn before calling the shared getBehindBall skill described earlier.

When getBehindBall is invoked, the robot tries to go to the point on the line drawn between the target goal and the ball and 20cm behind the ball. If the ball is not near the edge (i.e. in region 4 or 6) the robot will turn in a direction so that it can get to the target position quickest (figure 12).

When the ball is near the edge (in region 3 or 5), the robot will turn in a direction

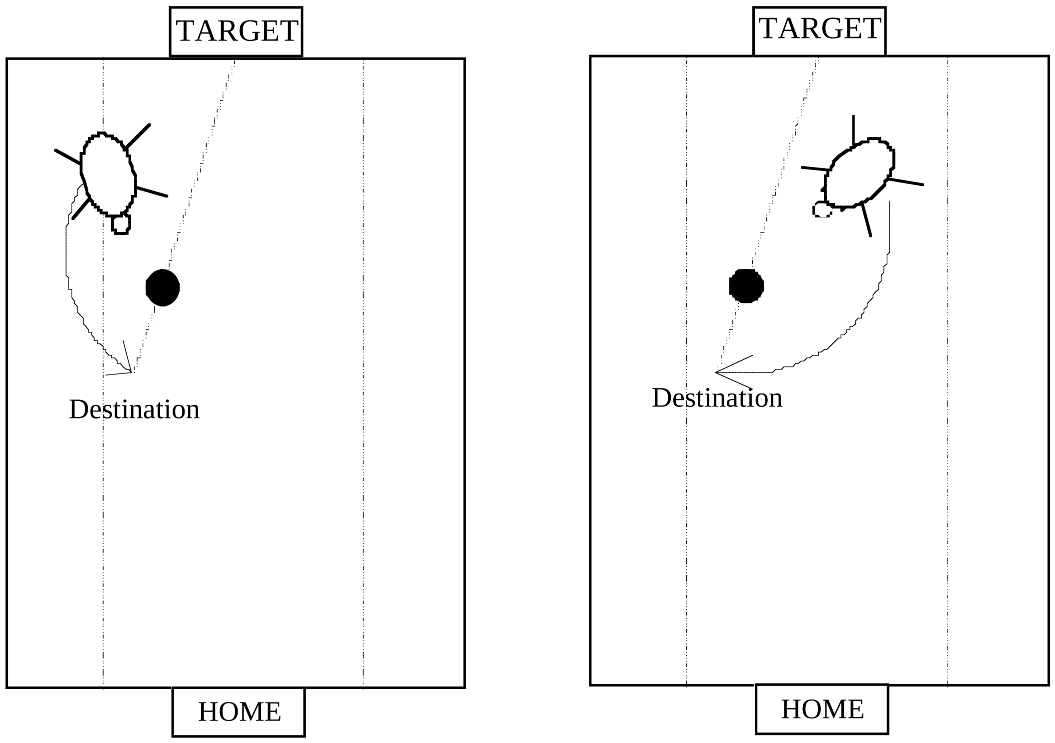


Figure 12 Turning direction when ball is not near the edge

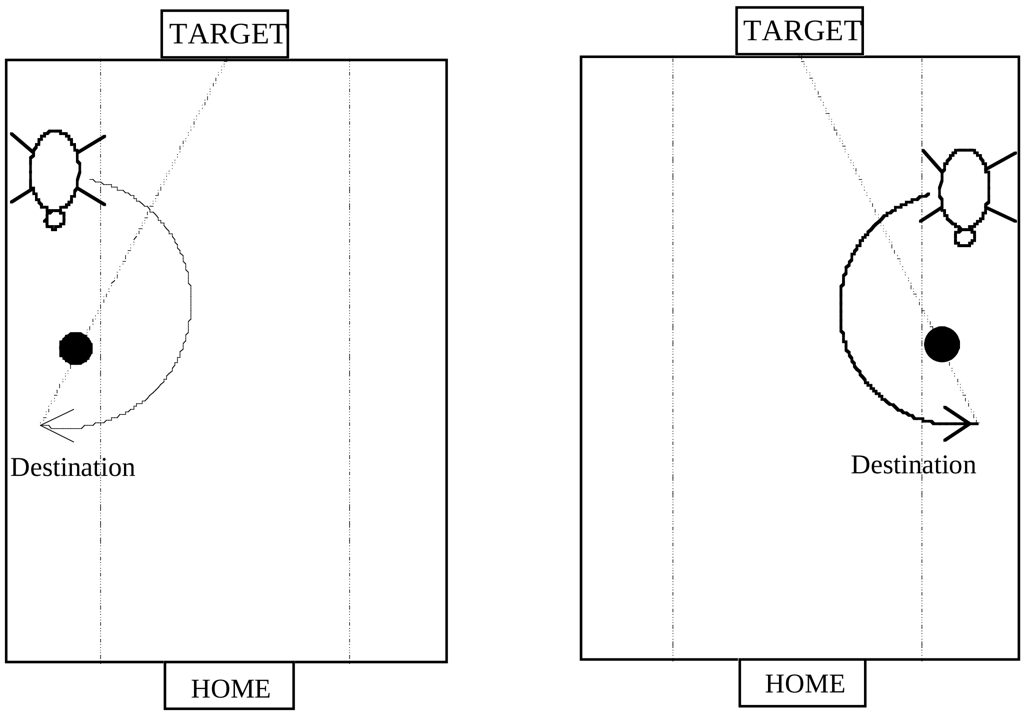


Figure 13 Turning direction when ball is near the edge

away from the edge avoiding colliding with the wall (figure 13).

8.2 The Goalie

At the highest level, the goalkeeper strategy employed at RoboCup 2000 utilises three key behaviours to defend its own goal. These behaviours are: -

1. Finding the ball
2. Tracking the ball and acquiring a defensive position
3. Clearing the ball

8.2.1 Finding the ball

Finding the ball begins with a 360ß rotation in the direction that would clear a ball stuck behind the robot away from the goal being defended. 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-range and long-range search.

If the ball was not found during the rotation, the *scanForBall* skill is initialised in a loop to control the head of the robot. At the same time, the robot will face away from the goal it is defending and walk backwards towards it. Once close to the goal, the goalie will acquire a heading of 90ß (towards target goal). Once positioned, this heading will oscillate between 45 — 135ß to allow the robot to see the ball at any location within the defended region (assuming it is not obscured by robots).

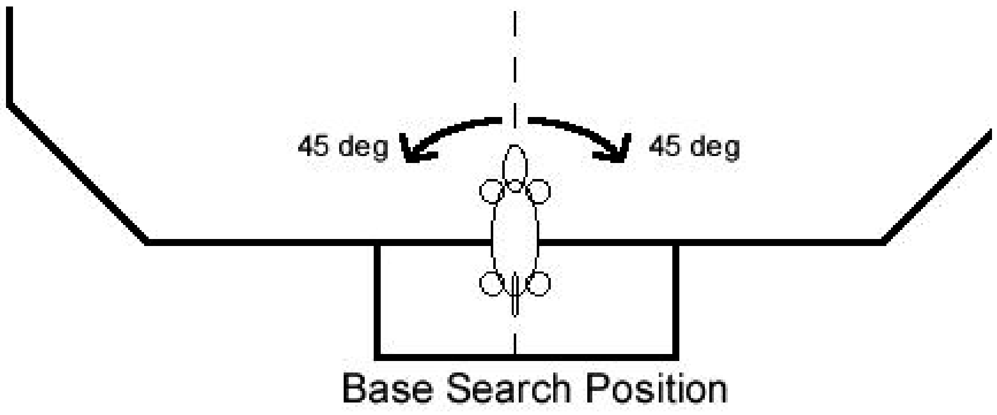


Figure 14. Home position of robot searching for the ball

8.2.2 Tracking the ball and acquiring a defensive position

Once the ball has been found, the robot will enter the 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 essentially tracks along a semicircle around the defended goal, keeping the body oriented towards the ball.

While tracking the ball, the robot will oscillate the head side to side as much as it can without losing the ball. The objective of this is to try and maximise the chances of seeing landmarks and help maintain localisation. The robot never stops watching the ball for the purposes of localisation.

8.2.3 Clearing the ball

Clearing the ball it activated when the ball gets closer than 80cm to the defended goal or the ball enters the designated penalty area. The mode finishes when the ball is kicked, lost, or moves more than 120cm from the defended goal. Upon deciding to clear the ball the robot will evaluate its position relative to the ball and determine whether it can directly attack the ball or should reposition behind it.

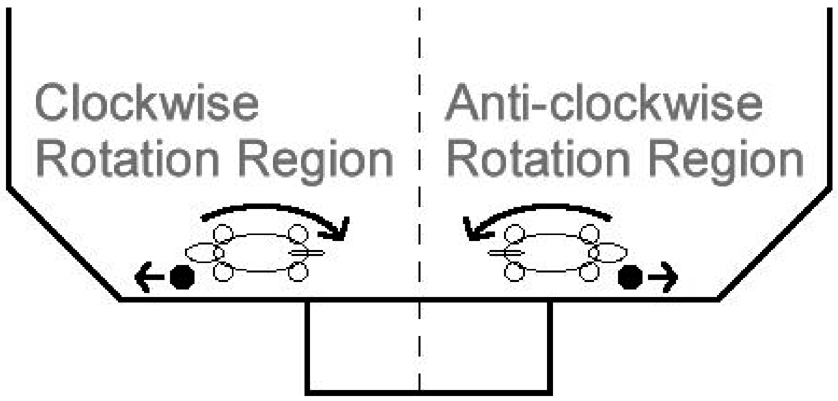
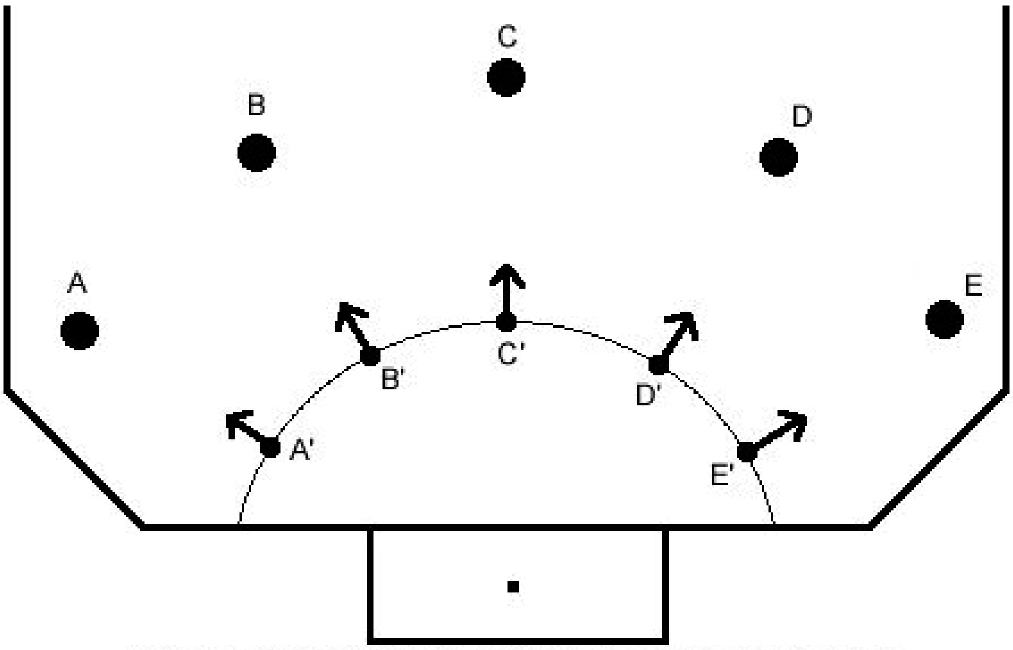


Figure 15 Rotation directions push ball out from behind robot.

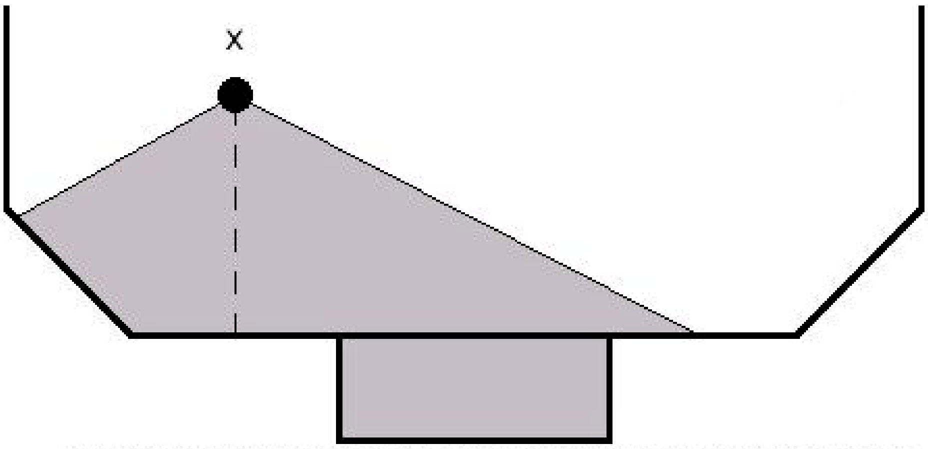


5 Ball positions (A,B,C,D,E) with the positions and headings acquired by the robot to defend them (A',B',C',D',E').

Figure 16 Defensive Positions Acquired by Goalie

The region in which the robot can attack the ball contains two often-overlapping components: -

1. The first component is a 120ß wedge based at the ball and extending off in the downfield direction. Within this region the robot has a good chance of knocking the ball upfield.
2. The second component is a 60ß wedge based at the ball and extending backwards towards the defended goal. Within this region the robot may not hit the ball upfield but should knock it away from the target goal.



Ball at position X can be attacked if the robot is within the shaded region

Figure 17 Attack Region 1

This combination of regions generally keeps the robot behind the ball yet allows the robot in front of the ball to effectively attack along the two field edges adjacent to the goal.

Once the robot is clear to attack the ball it will commence the *kickBall* skill. On approach to the ball, it the ball heading deviates outside of –15ß the robot aborts its kick and resorts to the *buntBall* skill. This combination of kicking and bunting strategies allows the robot to accurately move the ball when it has time to align, while still allowing the robot to make an aggressive, less accurate clear when the ball is being knocked about by opposition strikers.

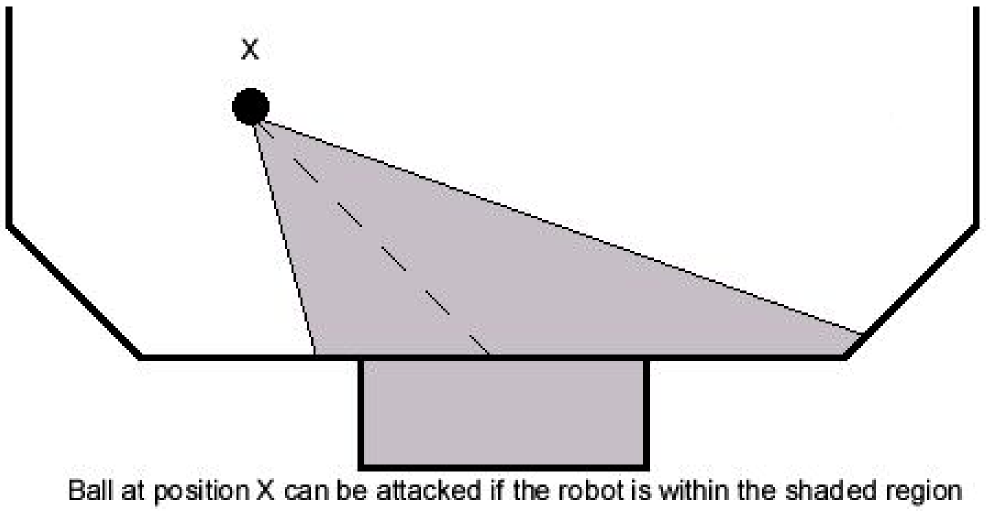

X
Ball at position X can be attacked if the robot is within the shaded region

Figure 18 Attack Region 2

9 Challenges

9.1 The RoboCup 2000 Challenge Competition

The Sony Legged league includes a Challenge competition in addition to the soccer matches. The three routines set for RoboCup 2000 were (1) a striker challenge, (2) a collaboration challenge, and (3) an obstacle avoidance challenge. The idea behind these challenges is to stretch technical research and development to ball handling and team skills that may also be useful during soccer matches. For the challenges each team completes a set task by itself. The team that scores a goal in the minimum time is the winner. If no goal is scored within the allotted time, then the team with the ball closest to the goal wins. The UNSW team won challenge 1 and 2 outright and came first overall in the challenge competition.

The striker challenge is the simplest. The ball and one robot (the striker) are placed at random on the field so that the ball is between the robot and the goal. The objective is simply for the striker to kick a goal within 3 minutes.

In the collaboration challenge two robots are placed at either end of the field and required to stay within their respective halves. The ball is placed nearer the robot furthest from the goal. The objective is for this robot to pass the ball to its teammate who has to kick a goal. 4 minutes is allocated for this task.

The obstacle avoidance challenge is similar to the striker challenge, except that two other stationery robots, one red and one blue, are also placed on the field between the striker and the ball. The objective is for the striker to kick a goal in 4 minutes without touching the stationery robots.

For each challenge the software programs controlling the robots are handed in 30 minutes prior to the start of the challenge. The starting positions for the robot(s) and ball are then determined at random.

9.2 Challenge Behaviours

The software development approach for the challenges was similar to that used to program the striker and goalie roles in soccer matches. The overall role is broken down into smaller behavioural units called skills. The activation of skills is defined by a decision tree in which the internal nodes test for certain environmental preconditions. The leaf nodes are the skills to be executed. For some leaf nodes a number of skills are executed in series building a more complex skill response. This makes the tree look like it has icicles or decorations on its leaves. We hence referred to it as the Christmas tree.

9.2.1 Striker Challenge Decision Tree

The decision tree for the striker challenge behaviour is shown in Program 2.

In the first rule (line 1.) the robot decides whether or not it can see the ball. We know that in this challenge the ball will be placed in the front half of the field closer to the goal. With the robot initially in the back half there is a risk that it may not be able to see the ball over the distance. If it cannot see the ball, the robot determines which half of the field it is in (line 2.). If it is in the back half it moves to its defensive

```
 1. if(ballLost)
 2.   if(ry<150)       skill = POSDEFENSE
 3.   else             skill = FINDBU
 4.                          FINDTURN
 5.                          POSDEFENSE
 6. else
 7.   if(low confidence)skill = GAINCONFIDENCE
 8.   else
 9.     if(ballDist<30)
10.       if(aligned)   skill = REACHOUT
11.                          TOUCHCHEST
12.                          CHINWALK
13.                          SHOOT
14.       else          skill = ALIGN
15.     else
16.       if(ry>by)     skill = GOSIDE
17.       else          skill = GOBEHIND
```

Progam 2. Striker Challenge

position which is defined to be the middle of the front half of the field. The objective is to move closer to the ball to increase the chance of seeing it. If it is already in the front half of the field and cannot see the ball, the robot invokes a sequential set of skills designed to find the ball (lines 3., 4. & 5.)

Once it has located the ball (line 6.), it is important to know in which direction the goal is located. It checks to see how well it is localised and if necessary it looks around at some of the beacons to check its position (line 7.). Having localised (line 8.) it measures its distance to the ball (line 9.) If it is close it checks to see if it is facing the ball in the desired direction (line 10.) and then invokes a series of skills in which it reaches forward with its front legs (line 10.), slowly walks up to the ball until the ball touches its chest (line 11.), walks with the ball towards the goal sensing it with its chin (line 12.) and then shoots the ball at the goal (line 13.)

If the robot does not face the ball correctly (in line 10.) it aligns itself (line 14.). If it is far from the ball (in line 9.) then if it is between the goal and the ball it walks to one side of the ball otherwise it walks behind it.

The decision tree invokes skills that are designed to create the conditions necessary to eventually score a goal. The decision tree is traversed about 25 times a second and skills fired immediately unless a previous skill has not timed out.

9.2.2 Striker Challenge Skills

The skills used in the striker challenge above determine the head and leg movements necessary to make the robot perform a particular unit of behaviour. Many of these skills are shared with the other roles. These skills are:

- Move to the defensive position (skill POSDEFENSE). The objective of this skill is to move the robot from its current position to the defensive position on the soccer field. The defensive position is defined by coordinates on the soccer field. The forward and sideways step sizes are proportional to the components of the vector from the robots current location to the defensive position. They are used as parameters to drive the walk (see Action/Execution). Because the robot can take larger steps forward than sideways the step sizes are maximised using elliptical equations. The head is panned from side to side +/- 40 degrees during the walk in the hope of finding the ball.

- Find ball while backing up (skill FINDBU). This skill attempts to find the ball while walking backwards. When the ball is lost from view it may have just rolled to either side of the robot. A good initial strategy to find the ball is to back the robot up looking from side to side in the hope of locating the ball. This skill times out after the robot has taken 5 steps backward.
- Find ball by turning on the spot (skill FINDTURN). In this skill the robot turns on the spot tilting and panning the head in an effort to find the ball. This skill is invoked if the ball is not found while walking backwards. The robot turns clockwise or anti-clockwise depending on which side it believes the ball to be from the world model (see section 5 Localisation). This skill times out after the robot has turned at least 360 degrees.
- Localisation. (skill GAINCONFIDENCE). In this skill the robot takes its eyes off the ball and looks around for beacons. Seeing a beacon or beacons localises the robot (see section 5 Localisation) increasing the confidence measure of its location. In this skill the robot remembers the last position of the ball and switches its gaze back to this position after localising.
- Moving the arms forward (skill REACHOUT). In this skill the robot crouches down moving its front legs forward to such an extent its elbows touch the ground. The idea is to be able to guide the ball between the front legs. If this action is performed in one step the robot may twist and loose its alignment to the ball. It is therefore executed in 10 very fast but smaller increments ensuring the robot retains its orientation. Throughout this skill the head tracks the ball.
- Move towards the ball until it touches the chest (skill TOUCHCHEST). The robot walks towards the ball in the crouched posture. It keeps its head high enough for the ball to pass under the chin. The front legs are spread to help to guide the ball. As the robot approaches the ball, the ball disappears from the view of the camera as it passes under the chin. It counts four steps before moving the front legs together and brining its head down to hopefully push the ball against its chest.
- Walk with the ball towards the goal (skill CHINWALK). The robot walks towards the goal with the ball between is front legs. The walk parameters are calculated to turn the robot towards the goal and walk forward. The mouth of the robot is kept slightly open and resting on the ball as it walks. In this way the robot can sense if the ball is still there. The mouth is moved by the ball during the walking action. A varying mouth sensor reading means that the ball is present. The robot uses the world model to turn towards an obscured goal. In the event that its confidence in its localisation drops to a low level, it executes a sub-skill, similar to GAINCONFIDENCE, before proceeding. The CHINWALK skill terminates if the ball cannot be sensed or the goal is close and lined up ready for kicking.
- Kicking (skill SHOOT). The robot lifts both front legs and brings them down in parallel on the ball in such a way that the ball is pushed out directly in a forward direction. The front legs both accelerate and guide the ball. (see also Action/Execution)
- Facing the ball in a particular direction (skill ALIGN). In this skill the robot circles sideways around the ball always facing it until it positions itself in a particular direction in relation to the ball. The desired direction depends on where

the robot is located on the field. For example, when it is near the target goal, the direction can be specified to line up the ball with the goal.

- Walk to one side of the ball (skill GOSIDE). This skill is used when the robot find itself between the target goal and the ball. In this situation it needs to go behind the ball without knocking it towards its own goal. To do this it walks to one side of the ball first. On which side it chooses to pass the ball is determined by the relative location of the goals, the robot and the ball so as to minimise the effort required or the possibility of scoring an own goal. If the ball is near one of the edges it chooses to walk past the ball on the opposite side to avoid running into the edge of the field.
- Approach the ball from behind (skill GOBEHIND). Being behind the ball means that the ball is located between the robot and the target goal. The robot walks straight at the ball, correcting its direction if it veers of this course. The head keeps tracking the ball as described in section 7.3 Action/Execution Application.

Each of the above skills are programmed as cases and detailed explicitly in the source code.

9.2.3 Collaboration Challenge

The collaboration challenge involves two robots, a passer and a shooter. The first passes the ball from the back half of the field to its partner in the front half of the field who then shoots the goal. Each robot is programmed to execute a similar task to the striker in the first challenge with some minor modifications to the rules and skills. We will just describe these modifications.

The passer's defensive position is now defined to be the middle of the back half of the field. Instead of walking to the goal it chooses the closest beacon on either side of the goal and walks towards it with the ball. Just before it reaches the halfway line it shoots the ball towards the target beacon. This strategy is used to both pass the ball to the other robot and to minimise the possibility of shooting a goal from beyond the halfway line that would incur a penalty. The shooter then executes the same strategy as the striker in the first challenge.

One over-ridding rule was added to the decision tree to ensure that each robot would not stray into the other's half. If the position of the robot is estimated to be offside or the ball is seen in the other half of the soccer field the robots would return to their respective defensive positions.

9.2.4 Challenge 3 — Obstacle Avoidance Challenge

Challenge 3 involves having a robot score a goal while avoiding 2 obstacle robots that stand on the spot. One of the stationary robots has red team markings while the other has blue team markings. One of the robots stands on the halfway line and the other stands on the penalty line as shown inf Figure 19.

The primary focus for challenge 3 is collision avoidance, as striker capabilities are common to all challenges. The base infrastructure to support collision avoidance is the robot recognition algorithm. However, due to the inaccuracies in determining the distance to an identified robot, Avoidance decisions cannot be reliably made based off a single image.

In challenge 3, all detections of each colour obstacle can be combined, as only 1 of each exists (assuming no false detections). For every frame containing an identified robot the following is performed: -

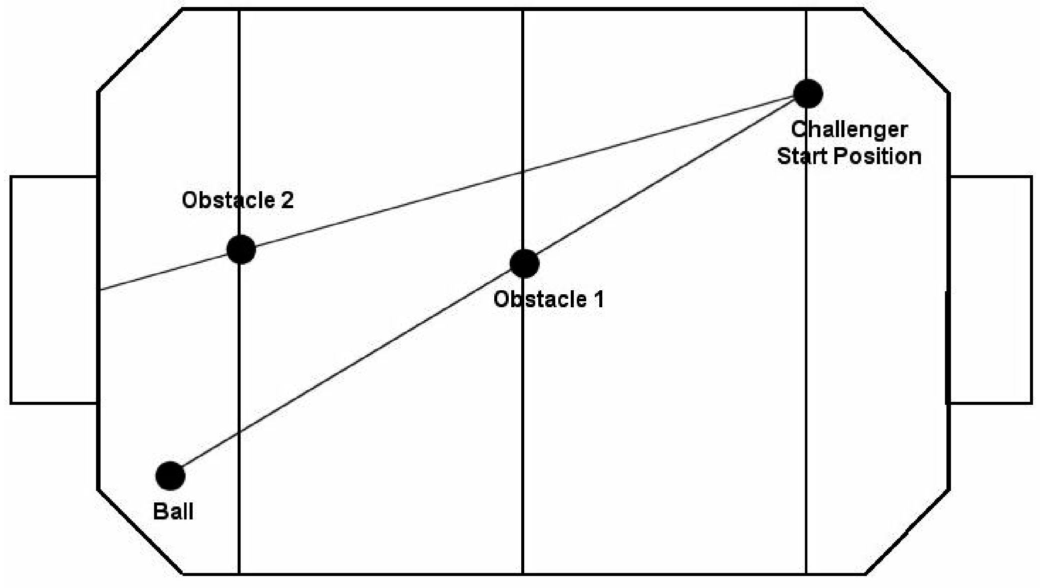


Figure 19 Example set up for challenge 3

1. Heading and distance from camera are resolved to heading and distance from the robot body.
2. Heading and distance from the robot are combined with the robots own localisation information to map the obstacle to a field position (x, y).
3. This identified field position for the obstacle is waited into the existing position for the obstacle using a learning rate that decays with every new observation.

Learning rate = 1/pow(observations, 0.5);
*obstacle.x = (1.0-alpha) * obstacle.x + alpha * curObs.x;*
*obstacle.y = (1.0-alpha) * obstacle.y + alpha * curObs.y;*

Source code to factor ÒcurObsÓ into the stored approximation ÒobstacleÓ

As such, with repeated observations the positions of the two obstacles will eventually stabilise at two distinct field positions. Once these positions have stabilised sufficiently the robot can plan a path that will hopefully guide the robot around the two obstacles.

Path planning

The path planner calculates the direction to drive the robot in to walk the robot to a destination while trying to stay more than X cm of the obstacles.

The inputs to the path planner are the current robot position (x, y, h), the destination (x, y) and the position of 1 or 2 known obstacles (x, y). The process of deciding on a path is as follows:-

1. A circle of radius X cm is drawn around any known obstacles.
2. A line segment is drawn between the current robot position and the destination.
3. If the line segment doesn't intersect any of the circles in part 1 then the robot is driven directly to the destination.

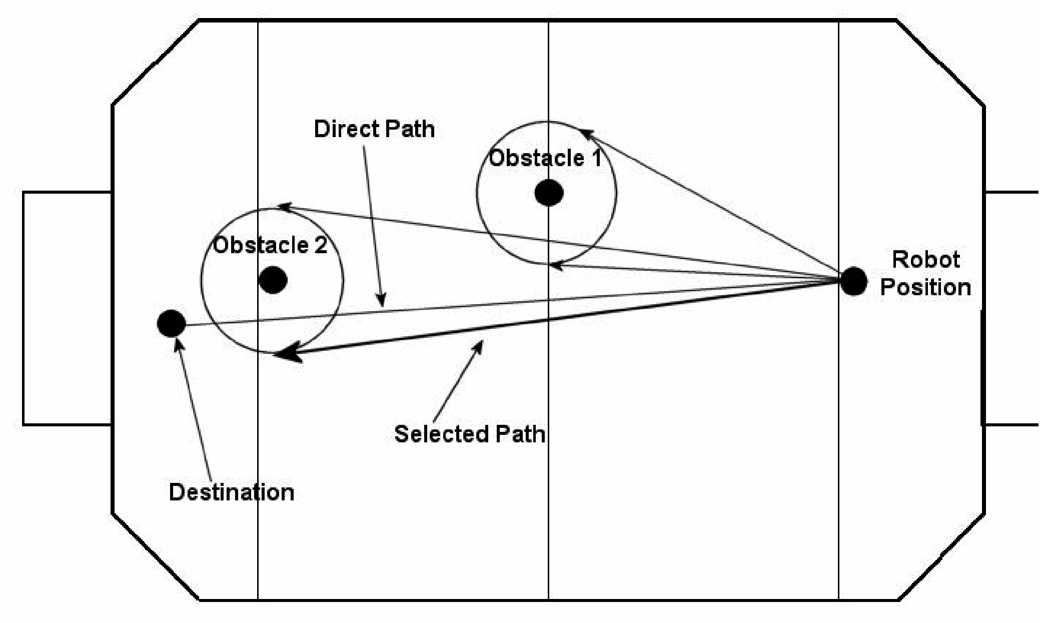


Figure 20 Four candidate paths. Direct paths go too close to obstacle 2.

4. If the direct path is rejected, 4 other paths are evaluated, each walking the robot to the left and right of each of the 2 obstacles. If the circles around the two obstacles intersect, paths heading between the two obstacles are removed. After this, any paths that force the robot within 20cm of the field edge are also deleted. Once a list of candidate paths is established, the path which deviates the robot the minimum from the direct path is chosen, under the assumption that it should be the quickest.

The lowest path is selected as it has the least heading deviation from the direct path.

Once the robot has walked a path around the two obstacles and is close to the ball, the strategy from challenge 1 is used to score the goal.

Problems

The field coordinates of the two obstacles are noisy and so is the robots localisation. As such the robot tends to walk erratically along the intended path, particularly when the robot is close to either of the obstacles. Sometimes the robot would still collide with the obstacle due to the noise within the system. As such, an emergency system was put in place so that the robot would immediately step away from an obstacle if the Infrared sensor measured it as being too close. This mechanism minimised problems related to noisy robot localisation but the problem still remains that obstacle positions could still converge inaccurately. The main cause of this problem has been the false identification of robots due to background noise, which generates field coordinates with large errors.

Performance

At the RoboCup 2000 competition, Problems recognising the blue robot uniforms unfortunately caused the above approach to be partially discarded. As such, the robot utilised robot recognition to plan a path around the red robot combined with a dead reckoning approximation of the two obstacle locations. Therefore, the robot plotted a wide path around the approximate locations of the robots and used the red robot avoidance in a supporting role.

Within the actual challenge, Sony incorrectly placed the robot on the field and the geometric assumptions that approximated the obstacles became incorrect. As such the robot collided with the first obstacle, then continued on to complete the task without touching the other obstacle. Incorrect placement occurred for over half of the teams, but wasn t a concern amongst the teams. UNSW placed one position behind Osaka, which was sufficient to capture the overall victory in the challenge

10 The Competition

The competition site for Robocup 2000 was the Melbourne Exhibition Centre located in Melbourne, Australia. The Sony legged league involved 12 international teams, namely: -

CM Pack'00	Carnegie Mellon University
Les 3 Mousquetaires	Laboratoire de Robotique de Paris
Baby Tigers	Osaka University
ARAIBO	The University of Tokyo
UNSW United	University of New South Wales
Upennalizers	University of Pennsylvania
McGill	McGill University
Humboldt Heroes	Humboldt University Berlin
Team Sweden	Orebro University and 2 universities
RoboMutts	University of Melbourne
SPQR	Universita' di Roma "La Sapienza"
Essex Rovers	University of Essex

The Draw at RoboCup 2000 split the 12 teams into 4 pools of 3 teams. Each of the 4 top placed teams were ensured to be in different pools, and each of the 3 new teams were also ensured of being in different pools.

Within each pool a team will play each of the other teams once, scoring 3 points for a win, 1 for a draw, and 0 for a loss. After all the round robin matches are completed, 1 team from each pool is eliminated, with the others progressing to quarterfinals.

In the quarterfinals, first from each pool plays second from a different pool. The loser from each match is eliminated.

In the semi finals, the four remaining teams play two separate matches. The winners progress to the final, with the other two teams playing off for third place.

10.1 UNSW United's path through the competition

10.1.1 Match 1: UNSW vs. McGill

This Match was a 14-0 victory to UNSW. The UNSW robots exhibited superior locomotion and reaction time to dominate the match. The McGill robots appeared to have good vision and localisation, as they could identify the ball and attempt to position correctly. We didn t notice any obvious localisation problems such as knocking the ball in the wrong direction, but the McGill robots did waste a significant amount of time looking for field landmarks. The primary weakness for McGill was the reaction time and speed of the locomotion they were using. McGill had significant difficulties reacting to the fast moving ball. Often the McGill robots would continue their current plan of action long after (3-4 seconds) the ball had been intercepted and was gone.

10.1.2 Match 2: UNSW vs. Essex

Essex were unable to compete in this match due to technical difficulties, so an exhibition match was held between 6 UNSW robots utilising varied strategies.

Round Robin Results for Pool C

UNSW	6
McGill	2
Essex	1

10.1.3 Match 3: UNSW vs. Humboldt (Quarter final)-

This match was a 11-0 victory to UNSW. The UNSW robots exhibited superior localisation and speed to aggressively attack the Humboldt goal. One particular weakness for Humboldt appeared to be their vision system. At the kick-off Humboldt would consistently manage to kick the ball deep into our half, yet their strikers failed to chase the ball to continue the attack. Our goalie had a tendency to get stuck on the mouth of the goal trying to clear the ball, meaning the ball was left untouched in the corner for a significant amount of time during the match.

The key strength of Humboldt was their powerful kicking motion. They could easily kick the ball the full length of the field but with very little directional control.

The advantage of the Humboldt kick was somewhat weakened by localisation problems. On several occasions during the match the Humboldt strikers became disoriented and knocked the ball in the wrong direction.

10.1.4 Match 4: UNSW vs. CMU (Semi final)

This match was a 12-1 victory to UNSW.

CMU utilised a powerful kick that consistently pushed the ball deep into the UNSW half at the start of play.

Both teams had accurate and reliable localisation except CMU had to invest a greater proportion of field time to acquire landmark information.

CMU utilised custom locomotion but with a different walking gait to UNSW. The CMU walk utilised a crawl gait whereby 3 legs are touching the ground at any given time, leaving them with a noticeable speed disadvantage. The stance of the CMU robots was fairly tall and narrow, making them unstable when contesting the ball.

During the second half of the match, the UNSW strikers appeared to suffer localisation problems, which was later diagnosed as an inaccuracy in the object recognition module.

10.1.5 Match 5: UNSW vs. LRP (final)

This match was a 10-0 victory to UNSW. UNSW dominated with superior stability, localisation and team play.

The LRP team utilised custom locomotion using a crawl style gait. The walk was not as fast as the UNSW walk and had an inherent instability, tending to falll backwards. This meant the LRP robots were easily bumped over when contesting the ball and would sometimes fall over after stepping on the field edge.

The primary strategy for LRP was to simply dribble the ball towards the UNSW goal. The team exhibited no kicking manoeuvrers and therefore struggled to clear the ball in open space. The UNSW strikers could chase down the ball even after LRP had made break.

During the match LRP appeared to have some vision problems, which led to the LRP robots confusing the red team jackets and the shadow in the yellow goal with the ball. The UNSW vision exhibited no major problems during the match.

11 Future development

While highly successful at the Robocup 2000 competition, all areas of the architecture employed by UNSW United have vast scope for improvement. Team co-operation will be an essential component in any successful team at RoboCup 2001. Communication will also be paramount to supporting this, and will either utilise the robots inbuilt speakers and microphone, or possibly wireless Ethernet (subject to supply from Sony).

Thus, we have identified several components of the system where we believe future development could be focused. We also suggest some new ideas that didn t reach development within the timeframe for RoboCup 2000. These ideas/components are as follows: -

Audio communication protocol - Allow Robots to communicate simple state information and negotiate roles on the field. For example, robots could "Call the ball" so that their teammates could move away into a supporting role.

Higher-level language for roles - Allows rapid strategy development and removes some of the complexities in writing stable, tuneable strategies.

Robot recognition based on pattern, shape and size of blobs - Should make distance and heading information more reliable and allow detection of multiple robots within a single frame.

Ball recognition (plus distance, heading, elevation, etc) using a neural network - Should make for more reliable ball information when the ball is on the frame edge or obscured.

Edge detection - Edge detection could play a strong role in supporting localisation, as field markings and sidelines could be recognised.

Object recognition other than by colour (i.e. shapes) - Should help prevent incorrect object recognitions due to background noise and colour mixing.

Development of other walk styles - Experimentation with other gaits and different locus geometries for increased speed, stability or manoeuvrability. A key area could be walks which help to control the ball and integrate well with kicks, dribbling and bunting.

Machine Learning of walk’s parameters - Find best parameters for straight-line speed, turning and sidestepping.

Better modelling of effect of claws on walk and in general better geometric model of robot for walking - Current model assumes that robot always walks on the ball of the foot, but a lot of the time the robot is standing on the claws, particularly with the rear legs.

Development of new kicks - Different variations of current kick, as well as single legged kicks and heading the ball.

Better ball dribbling using machine learning - Try to reduce the random component evident in the dribbling and bunting skills.

Ball tracking by anticipation - Rather than simply drive the head to the ball s current position, utilise information on the balls velocity and acceleration to predict the ball’s future position.

Adaptive use of camera resolutions - Devise a mechanism that utilises the most suitable resolution (Low, Medium or High) for a given situation. For example, low-resolution could be used to reduce CPU load when the ball is filling the major proportion of the camera image. Also, a localised scan in the high-resolution image could help identify objects that are questionable at the medium resolution (long range ball, beacons, patches).

Improved Beacon Recognition using 2D/3D transformation - When two half-beacons are identified. A 2D/3D transformation could be used to map camera orientation of the two half-beacons to their real world orientation. This way the object recognition module can check whether the two half beacons are aligned in the vertical before connecting them to form a beacon. Note: This was an evident problem at RoboCup 2000; Yellow background noise was merged with a pink half-beacon even though they were oriented horizontally. The Yellow noise was chosen in preference to the actual blue half-beacon which was clearly visible on the captured camera image.

Active Localisation - At RoboCup 2000, the robot strategies took a passive role to acquiring information for localisation. This means the robot almost never decided to look around and acquire information about the environment. The acquisition of this information was purely a result of searching for the ball and catching an occasional glimpse of landmarks in the background of the vision. One possible idea would be for the robot to use its current localisation to continuously calculate the two beacons that have the least heading deviation from the current head position. Then, whenever the strategy level deems it "safe" to quickly look away from the ball the robot could seek some visual conformation that its localisation is correct.

Separation of head and leg control - The infrastructure used at RoboCup 2000 ties together head and leg control into a single operation, causing some complications.

Firstly, the head effectors inherit the buffering scheme designed to provide smooth operation of the legs (5 frames). This buffering scheme is good for some head tasks but not for "watching the ball", which should work better without buffering.

Secondly, smooth frame rate independent head movements require more detailed information relating to the use of the buffers, such as the number filled (0,1,2) and the latest motor values sent.

Using world model and odometry to detect locking state - The robot should be able to detect that it is stuck on something when odometry says that the robot is moving yet the world model says that it is still at the same spot.

Other robot detection using force on motors —When the robot collides with another robot, excessive torque output from the leg motors could be detected and signal that a collision occurred. Could also be used at goalmouth and field edge.

12 Conclusion

From the performance of this system at RoboCup 2000, it is clearly evident that we are very happy with the system that we have developed. We feel we have utilised our limited resources well to develop a solid stable system which will form a good basis for future development. During the development of the system, we have maintained a stable, evolving infrastructure consisting of our vision, object recognition, localisation and locomotion modules. For inclusion into the system infrastructure, new code had to displayed and shown to operate reliably.

The idea of an evolving infrastructure allowed concurrent development throughout the entire project. Modules within the system could be updated and plugged in with the benefits seamlessly being delivered to the rest of the system. Locomotion and localisation modules were continuously evolving, yet strategies would maintain consistent/improved behaviour as the supporting modules were updated.

One key part of the proposed architecture that didn t appear in the fullest sense in the final product was the separation of the strategy into control and skill components. The primary reason this separation didn’t really happen is that very few skills are absolutely identical between the striker and goalie. As such, skills have not been generalised between the two strategies but a lot of code is shared between the two implementations with trivial differences. Perhaps this could be solved with some form of skill parameterisation.

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

Based on the shock of competition experienced in 1999, weekly 2 on 2 games were played to identify and assess the improvements being made. The primary objective of any testing was to make the testing conditions as close as possible to competition conditions. This proving ground for development proved invaluable as many ideas that worked wonderfully in isolation really took a beating when placed in a match. Competition conditions stress the capabilities of all the modules within the system, and expose any problems alarmingly quickly.

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.

At the end of the day, we hope that we have developed a flexible, stable and understandable system that can form a strong basis for Robocup 2001 development.

References

Hornby, G. S., Fujita, M., Takamura, S., Yamamoto, T., Hanagata, O. *Evolving Robust Gaits with Aibo.* IEEE International Conference on Robotics and Automation. pp. 3040-3045. 2000.

Dalgliesh, J., Lawther, M. (1999). *Playing Soccer with Quadruped Robots*. Computer Engineering Thesis, Univesity of New South Wales.

Sony supplied Aperios and Open R references

Veloso M., Winner E. , Lenser S., Bruce J., Balch T. *Vision-Servoed Localization and Behavior-Based planning for an autonomous quadruped legged robot*. In Proceedings of AIPS-2000, 2000.

Fox D., Burgard W., Dellaert F., Thrun S. *Monte Carlo Localization: Efficient Position Estimation for Mobile robots*. In Proceedings of AAAI-99.

Hugel V., Bonnin P., Raulet L., Blazevic P., Duhaut D. *Vision based behavior strategy to play soccer with legged robots*. In Veloso M., Pagello E. and Kitano H. eds, Robocup-99: Robot Soccer World Cup III, Springer 2000.

Foley J., van Dam A., Feiner S., Hughes J. *In Computer Graphics: principles and practice*. Addison Wesley, 1997.

Kimura H., Shimoyama I., Miura H. *Dynamics in the dynamic walk of a quadruped robot*. University of Tohoku, 1998.

Appendix: ParaWalk Documentation

Overview

ParaWalk is a collection of effector macro-actions designed to make the Sony 4 legged robots walk, move the head, kick the ball, etc. These different macro-actions are collectively call called "walks" in this appendix.

This appendix covers the following three main topics:
1. Walk styles and parameters.
2. Geometry and equations used to implement the walking movements
3. Source code outline

Walk Styles and Parameters

The walks are activated by calling the `makeParaWalk or makeParaHead` method in Schema or its subclasses. For example: `makeParaWalk(0,FC,SC,TC,65,73,97,16,19,33,20,-35,20,HC,dtilt,dpan);`

The time to complete a walk macro depends on its type. The following walks are available, together with information about the 16 parameters.

A "step" in trot walks is defined to be the time taken to shift the weight symmetrically to the other pair of legs.

Methods

makeParaWalk(Forward, Left, turnCCW). Simple walk using trot (walkType 1) defaults and no head movement ie hType = null = 0.

makeParaHead(hType, tilt, pan). Head movement only. Robot continues current walk and leaves robot in last walk 0 or 1 stance if walk step has finished.

makeParaWalk(walkType, Forward, Left, turnCCW, hF,hB,hdF,hdB,ffO,fsO,bfO,bsO). Head is not used and set to hType null = 0.

makeParaWalk(walkType,Forward,Left,turnCCW,hF,hB,hdF,hdB,ffO,fsO,bfO,bsO ,hType,tilt,pan,mouth). Full parameterised walk.

Parameter Description

1. walkType (int). The walk macro-action to be activated.
2. Forward (double). The distance, in cm, the walk should move the robot forward per step. Negative values move the robot back.
3. Left (double). The distance, in cm, the walk should move the robot left per step. Negative values move the robot to the right.
4. turnCCW (double). The angle, in degrees, that the robot should turn counter clockwise per step. Negative values turn the robot clockwise.
5. PG (int). The time to complete a step in units of .008 seconds. eg if PG = 65, each walk step takes 0.52 seconds.
6. hF (double). The vertical height, in mm, from the shoulder joint to the ground. The point on the ground vertically below the shoulder is called "shoulder ground".
7. hB (double). The vertical height, in mm, form the hip joint to the ground. The point on the ground vertically bellow the hip is called "hip ground".
8. hdF (double). The height, in mm, that the robot lifts the front legs while walking.
9. hbF (double). The height, in mm, that the robot lifts the back legs while walking.
10. ffO (double). The home position of the front paws, in mm, forward from shoulder ground.
11. fsO (double). The home position of the front paws, in mm, spread left and right from shoulder ground.
12. bfO (double). The home position of the back paws, in mm, forward from hip ground.
13. bsO (double). The home position of the back paws, in mm, spread left and right from hip ground.
14. hType (int). The type of head movement ie none, relative, absolute.
15. tilt (double). The angle in degrees to move the head up/down depending on hType.
16. pan (double). The angle in degrees to move the head left/right depending on hType.
17. mouth (double). The angle, in degrees, to open the mouth to.

Fast Trot

A gait where diagonal legs are off the ground simultaneously. This walk is our fastest and has been clocked at 1200cm/min. It is low to the ground. It also appears to shake the camera more than the next walk because of its canter action. Generally, the Forward, Left and trunCCW parameters are used to control this walk. Other leg movement parameters can be left unchanged.

Parameter	**Comments**
walkType	must be 0
Forward	range -8.0 cm to 8.0 cm by itself, but generally depends on Left & turnCCW
Left	range -4.0 cm to 4.0 cm by itself, but generally depends on Forward and turnCCW
turnCCW	range -30 deg to 30 deg by itself, but generally deepends on Forward and Left
PG	use 65 as default. Can be any +ve multiple of 5
hF	use 73 mm as default. Limited by physical constraints.
hB	use 97 mm as default. Limited by physical constraints.
hdF	use 16 mm as default. Limited by physical constraints. Must be >0
hdB	use 19 mm as default. Limited by physical constraints. Must be >0
ffO	use 33 mm as default. Limited by physical constraints.
fsO	use 20 mm as default. Limited by physical constraints.
bfO	use -35 mm as default. Limited by physical constraints.
bsO	use 20 mm as default. Limited by physical constraints.
hType	0 - no action, 1 - relative, 2 - absolute
tilt	tilt head in degrees. Up +ve, down -ve
pan	pan head in degrees. Left +ve, right -ve.
mouth	open mouth by this many degrees, 0=closed. —ve down.

Trot

A gait where diagonal legs are off the ground simultaneously. This walk was the first parameterised walk and has been clocked at 850cm/min. It appears to shake the camera less that the above walk and stands a little taller. Generally, the Forward, Left and trunCCW parameters are used to control this walk. Other leg movement parameters can be left unchanged. Can use makeParaHead(hType, tilt, pan).

Parameter	**Comments**
walkType	must be 1
Forward	range -6.0 cm to 6.0 cm by itself, but generally depends on Left & turnCCW
Left	range -4.0 cm to 4.0 cm by itself, but generally depends on Forward and turnCCW
turnCCW	range -40 deg to 40 deg by itself, but generally deepends on Forward and Left
PG	use 80 as default. Can be any +ve multiple of 5
hF	use 100 mm as default. Limited by physical constraints.
hB	use 120 mm as default. Limited by physical constraints.
hdF	use 15 mm as default. Limited by physical constraints. Must be >0
hdB	use 18 mm as default. Limited by physical constraints. Must be >0
ffO	use 25 mm as default. Limited by physical constraints.
fsO	use 20 mm as default. Limited by physical constraints.
bfO	use -25 mm as default. Limited by physical constraints.
bsO	use 20 mm as default. Limited by physical constraints.
hType	0 - no action, 1 - relative, 2 - absolute
tilt	tilt head in degrees. Up +ve, down -ve
pan	pan head in degrees. Left +ve, right -ve.
mouth	open mouth by this many degrees, 0=closed. —ve down.

Kick

When the ball is touching the chest of the robot and the front body of the robot is close to the ground, this "walk" kicks the ball forward using a scissors lever action. (still being tuned). Note this walk uses customised settings for all effector values and hence most parameters have no effect.

Parameter	Comments
walkType	must be 2
other parameters have no effect and can be set to 0.	

Geometry and Equations

Overview

The walk is implemented by moving each paw in a rectangular locus. As the paw moves along the bottom edge of the locus it touches the ground and drives the robot. This motion has a similar effect to a rotating wheel. The locus plane is always perpendicular to the ground. Its inclination to the side of the robot and the direction the paw traverses the locus determines whether the robot moves forward backwards, sideways or turns. The locus planes of all the legs need to be co-ordinated to work together without conflict. To move forward all planes need to be parallel to the sides of the robot and all the paws need to move in the same direction. To turn on the spot diagonally opposite planes need to be parallel and the paws move in complementary directions to effect a turning motion.

The walk operates by passing control instructions via parameter variables in method makeParaWalk. The control parameters are used to work out the locus for each of the paws. The paws are stepped around this locus by calculating the paw co-ordinates for each step position. Each set of paw co-ordinates determines each of the limb angles using inverse kinematics. Kinematics is the process of calculating the position in space of the end of a linked structure, given the angles of all the joints. Inverse Kinematics does the reverse. Given the end point of the structure, what angles do the joints need to be in to achieve that end point. Finally PWalk sends the limb angle values to the motors to complete its job of activating the robot effectors.

The geometry and equations that follow build the above procedure up in stages. First we will describe the inverse kinematics from the 3 co-ordinates of the paw to the 3 motor positions of each leg. We then describe how the co-ordinates are calculated taking into consideration difference in the front and back height of the robot. Finally we show how the locus is calculated to provide the paw positions.

Inverse Kinematics

We set up a three dimensional co-ordinate system for each leg. The origin of this space is the point where the leg joins the body. The x direction is forward and parallel to the top of the robot body. The y direction is down and parallel the back of the robot body. The z direction is sideways away from and perpendicular to the robot's side. (See fig/slide Inverse Kinematics). Hence for example the x-y plane contains the side of the robot body.

We now construct a plane which contains the limbs of the leg. We call this the limb plane. The limb plane intersects the x-y plane through the point where the upper leg limb joins the body (see fig/slide Inverse Kinematics Limb Plane). The limb plane co-ordinates are designated u & v. The direction of u is parallel to the upper limb. (Don't confuse this with the u-v plane in the vision module. There is absolutely no connection.). It is now possible to precisely depict and describe each of the limb angles in terms of the x,y,z space and u,v plane.

A_1 = the angle the limb plane makes with body side
A_2 = rotation of the projection of the upper limb in the x-y plane
A_3 = angle between lower and upper limbs in the u-v plane

Unfortunately, the robot's leg has a geometry such that even though all motor angles are zero and the leg is straight, the leg does not reach down to its furthest possible position. This occurs because the legs are hinged at the knee on one side of the leg. (See fig/slide Inverse Kinematics A_3)

We first find A_3 = angle between lower and upper limbs in the u-v plane. This is calculated in fig/slide Inverse Kinematics A_3 from the limb sizes and x, y and z position of the paw. We next calculate A_1 = the angle the limb plane makes with the body side (See fig/slide Inverse Kinematics A_1). Because the direction of u is parallel to the upper limb, we can extend the upper limb on the u-v plane to the paw's u value and use this projection to find the angle between the limb plane and the side of the robot ie A_1. It remains now only to project the leg in the limb plane onto the x-y plane using A_1 and calculate A_2 = rotation of the projection of the upper limb in x-y plane. This is done in fig/slide Inverse Kinematics A_2 referencing the diagram in fig/slide Inverse Kinematics A_1.

Co-ordinate Transformation for Robot front and back height

If the body of the robot is tilted because the front and back of the robot are at different heights above ground level then we need to perform a simple transformation to adjust for this effective rotation of the x-y plane. We also take the opportunity to introduce the f-s co-ordinate system for the ground plane. We take the origin of the ground plane to be that point below the x,y,z origin projected onto the ground plane. The f dimension direction is forward from the robot's viewpoint and the s dimension direction is sideways out from the body. The equations to transform a point f,s to x,y and z are given in fig/slide Projection on Ground Plane.

Locus of Paw Positions

To define the rectangular locus we first determine the start and end points on the ground plane of its bottom edge. This is the vector sum in the f-s plane of each of the control parameters which determine the forward, sideways and turning movements as show in fig/slide Start and End Positions of Locus on Ground. The parameters spreading the legs to their initial home position are passed to the PWalk and included in the calculation of the end points. Only the equations for one leg are shown, but others follow in a similar way. The turn control parameter is assumed to be in mm, but of course has the effect of turning the robot on the spot. The start and end f-s points on the ground plane together with the lift leg height parameter fully determines the rectangular locus. It is now just a matter of finding the discrete points along the locus to step the paw around this path.

It is important for the paw to traverse the ground stroke in the same period of time it takes to move the paw back to the start position by lifting it, reversing it and lowering it, to achieve a constant velocity of the robot body. The ground stroke is divided into the number of steps specified by the speed parameter (PG for points on the ground) in makeParaWalk. From PG the number of steps up (PGH), reverse (PGR) and down (PGH) is calculated as per fig/slide Positions Around Paw Locus.

Implementation

We are now ready to implement the walking action. The control, speed and stance parameters are passed from the behaviour module by the makeParaWalk method. We first calculate the start and end points of the locus. We then step the paw around this locus starting from a "home" position. The home position is either the middle top or bottom of the locus depending on where the leg is synchronised in the gait. At each step on the locus we have the f, s and height above ground level values. We convert these to x,y,z values using the co-ordinate transformation for robot front and back height. We then apply inverse kinematics to find the required motor positions which are sent to the effectors. The source code makes the implementation of the above procedure explicit. It will be outlined next.

Source Code Outline

The source code for robot actions has been isolated in class PWalk. The behaviour module class Schema and its sub-classes eg Forward control robot walking actions by passing control, speed and stance parameters using method makeParaWalk. Method makeParaWalk acts via class AtomicAction on class PWalk.

The variables names used in the source code follow the variable names in the above description and the figs/slides as much as possible. For example A_5 is theta5 and height at the front is hf.

PWalk has two main methods, continuePStep and pStep. continuePStep calls pStep until the legs have finished traversing half the paw locus from middle top/bottom to middle bottom/top respectively or in the case of the kick, until the kicking action completes.

pStep is the main driver advancing the paw around its locus and setting motor commands using double buffering.

`step_` is a counter that increments as the paw is stepped around the locus. It ranges from 1 to 2*PG. `step` is an analogous counter for the diagonally opposite leg in the trot gait.

Only if the paw is at a home position are new walk control and stance parameters are accepted, otherwise they remain at their previous values. In other words the step size and direction is committed for half a locus cycle. The control parameters are scaled using calibration equations for forward or sideways commands in cm or turn commands in degrees to try and achieve these values in practice.
Next mx, my and mh are calculated. They tell the localisation module to increment the orientation of the robot due to the walking action. This concludes the code activated at a home position.

The code that follows now applies to each paw step around the locus.

First the robots localisation is updated.

For walkType 0 the canter action is introduced by sinusoidally varying the front and back height of the robot.

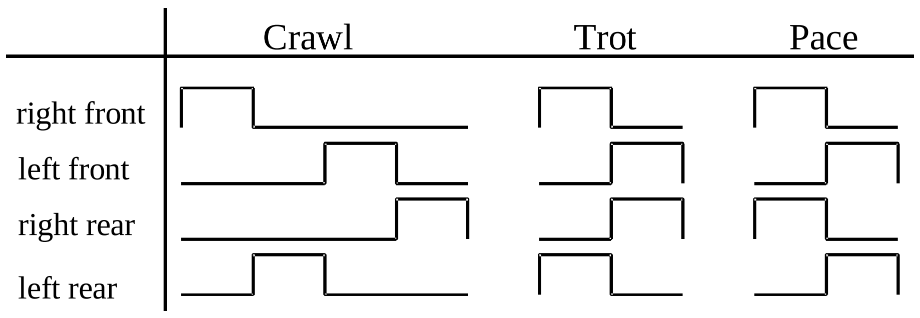
We then work out the leg motor positions for each leg in turn. The following calculations use the equations derived in the section Geometry and Equations. First the start and end points of the ground stroke is calculated. Next the number of steps (PGH & PGR) around the rest of the locus is determined. For each stage around the rectangular locus we determine the f, s and height value of the paw. We transform from f,s,h to x,y,z co-ordinates. Finally we determine, using inverse kinematics the three leg motor angles. Each leg is processed in turn.

Finally all head and leg motor values are sent to the effectors. The head movements are calculated either relatively or absolutely depending on parameter hType. In the case of relative movements the head is moved smoothly in the 5 effector frames to it new position. hType 3 also moves the head in equal increments to its new absolute position.

The kick action (walkType case 2) simply reads leg positions from and array `posa[set][numOfJoint]`. The two sets of values to effect the kick are specified in PWalk.h.

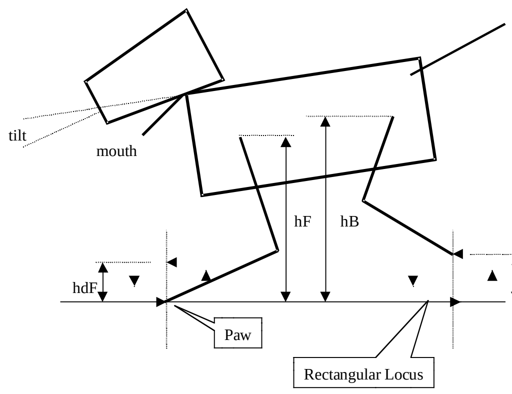
walkType case 1000 is used internally in the code where head movements alone are specified ie `makeParaHead`. It simply freezes the leg positions at their last value at one of the home positions while it moves the head as commanded.

Quadruped Gaits

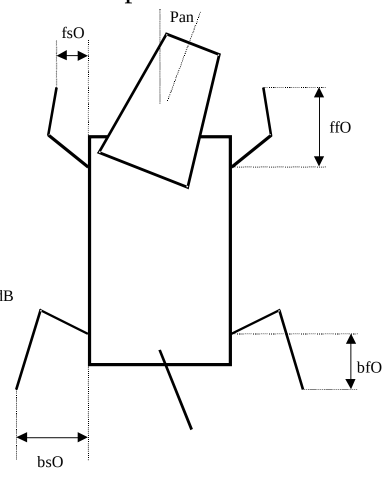


Robot Stance & Action Parameters

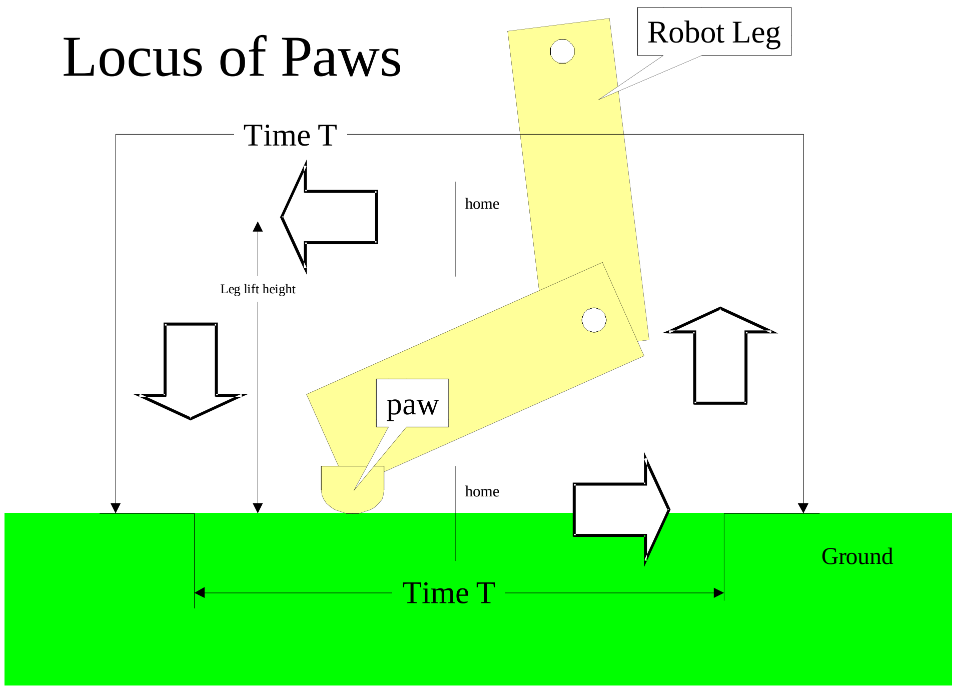
Side View



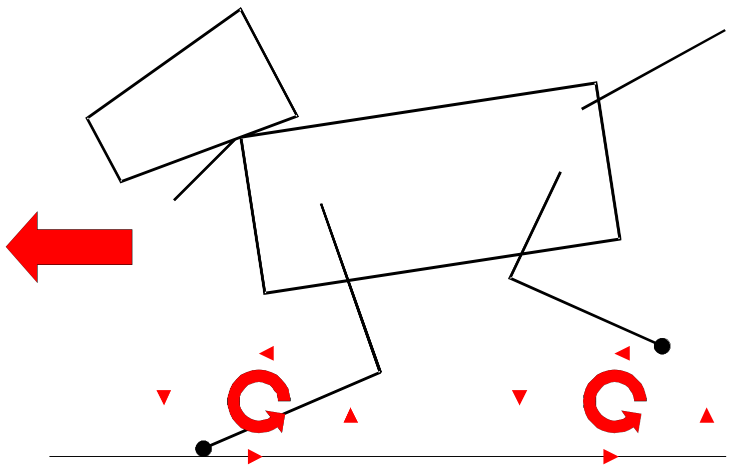
Top View



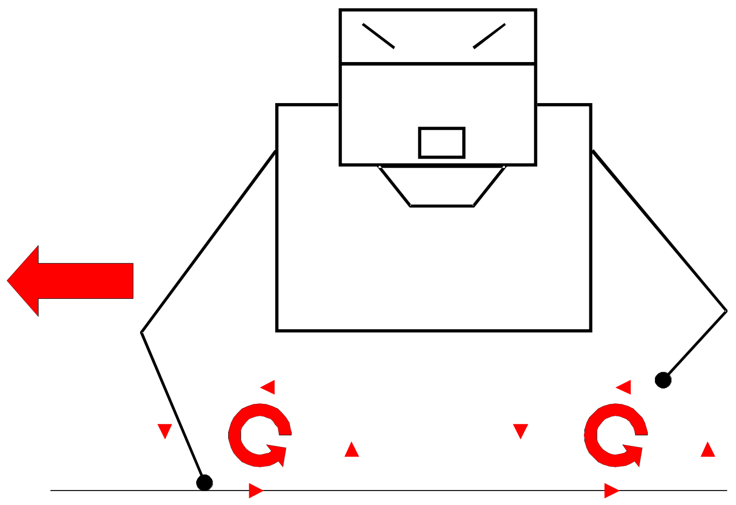
Locus of Paws



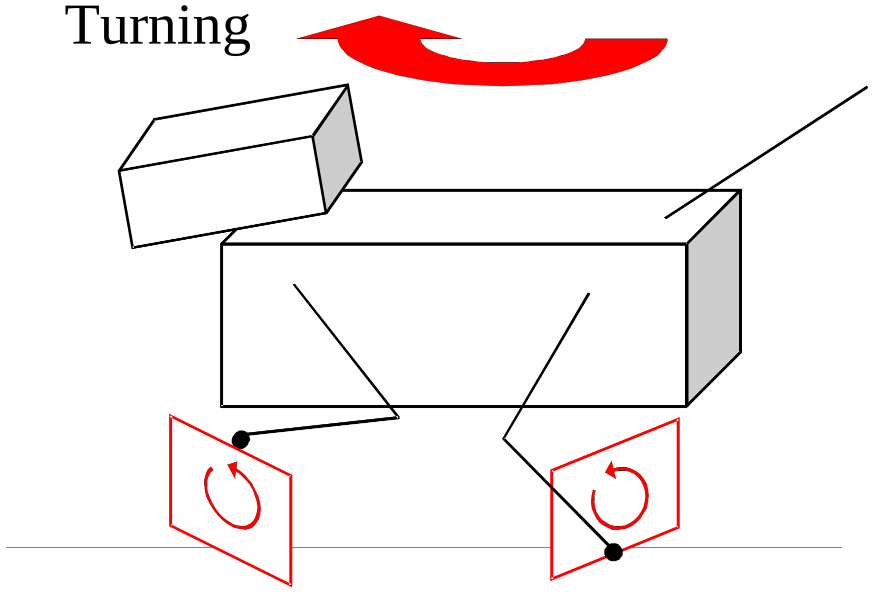
Forward Walk



Sideways Walk

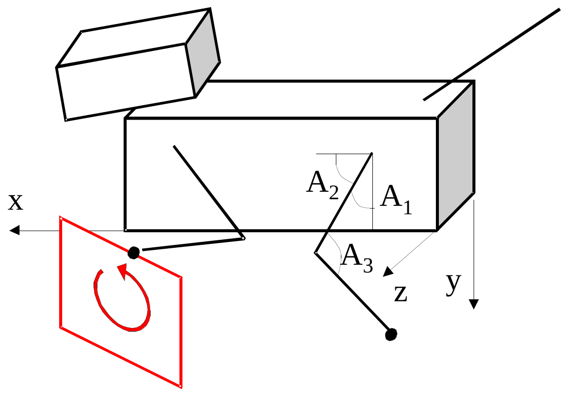


Turning

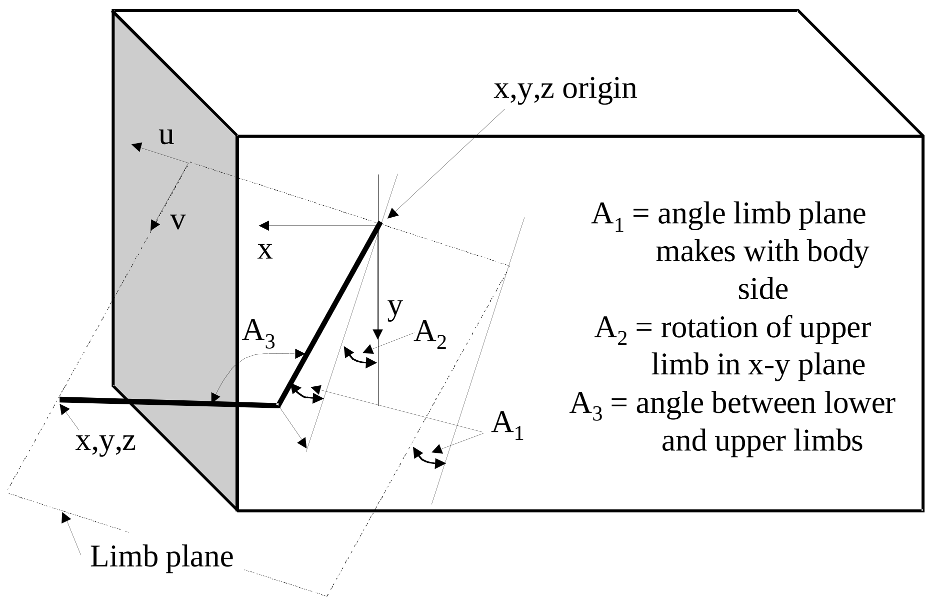


Inverse Kinematics

Kinematics is the process of calculating the position in space of the end of a linked structure, given the angles of all the joints. Inverse Kinematics does the reverse. Given the end point of the structure, what angles do the joints need to be in to achieve that end point.

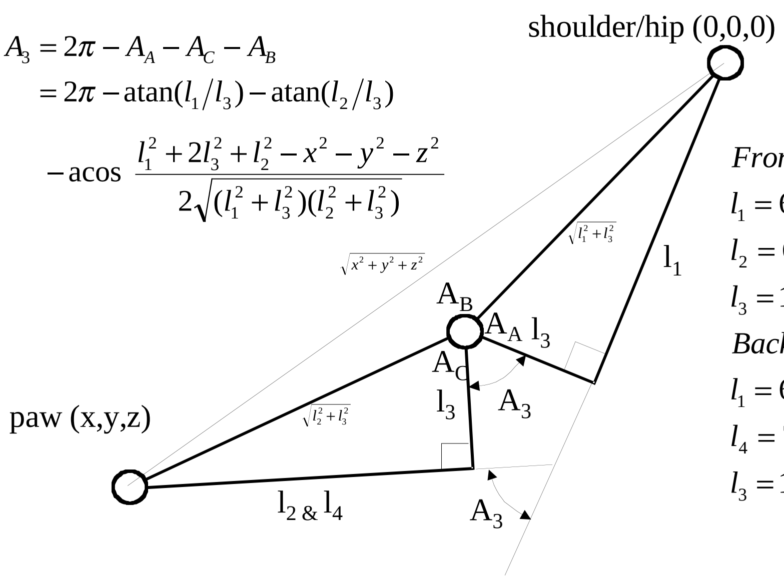


Inverse Kinematics Limb Plane


x,y,z origin
u
v
x
y
A_3
A_2
A_1
x,y,z
Limb plane
A_1 = angle limb plane makes with body side
A_2 = rotation of upper limb in x-y plane
A_3 = angle between lower and upper limbs

Inverse Kinematics A$_3$

$$A_3 = 2\pi - A_A - A_C - A_B$$
$$= 2\pi - \operatorname{atan}(l_1/l_3) - \operatorname{atan}(l_2/l_3)$$
$$- \operatorname{acos}\frac{l_1^2 + 2l_3^2 + l_2^2 - x^2 - y^2 - z^2}{2\sqrt{(l_1^2 + l_3^2)(l_2^2 + l_3^2)}}$$



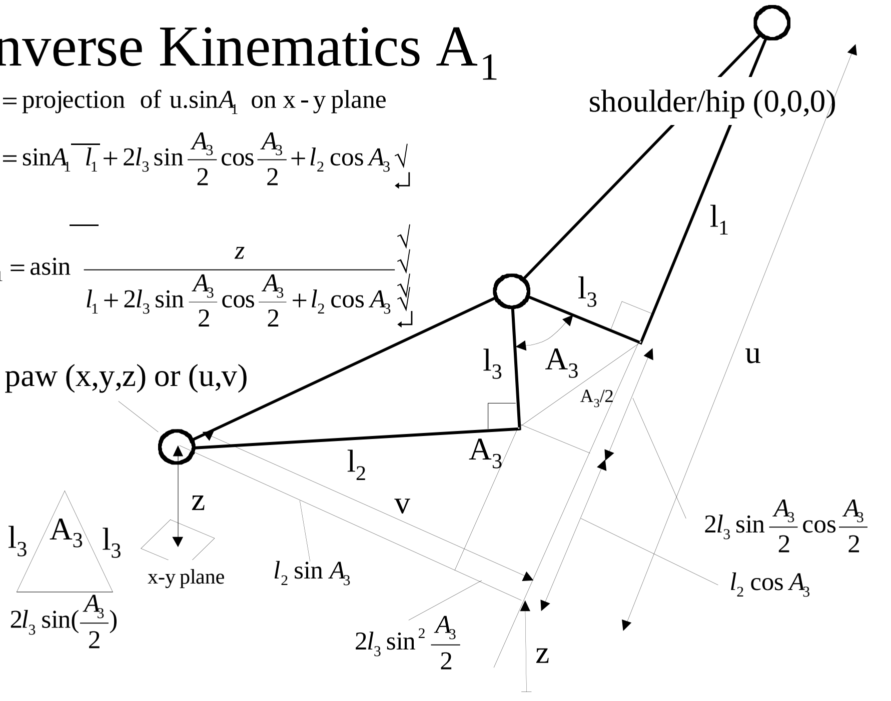
Front Legs
$l_1 = 60mm$
$l_2 = 65mm$
$l_3 = 15mm$
Back Legs
$l_1 = 60mm$
$l_4 = 75mm$
$l_3 = 15mm$

Inverse Kinematics A$_1$

$z =$ projection of u.sinA_1 on x - y plane

$$= \sin A_1 \left(l_1 + 2l_3 \sin\frac{A_3}{2}\cos\frac{A_3}{2} + l_2 \cos A_3 \right)$$

$$A_1 = \operatorname{asin}\left(\frac{z}{l_1 + 2l_3 \sin\frac{A_3}{2}\cos\frac{A_3}{2} + l_2 \cos A_3} \right)$$



Inverse Kinematics A_2

From limb plane

$$u = 2l_3 \sin^2 \frac{A_3}{2} + l_2 \sin A_3$$

$$v = l_1 + 2l_3 \sin \frac{A_3}{2} \cos \frac{A_3}{2} + l_2 \cos A_3$$

Projection on x - y plane

$$x = v \cos A_1 \sin A_2 + u \cos A_2$$

$$y = v \cos A_1 \cos A_2 - u \sin A_2$$

Solving for A_2

$$A_2 = \pm \text{acos} \frac{\text{yv}.\cos A_1 + ux}{u^2 + v^2 \cos^2 A_1}$$

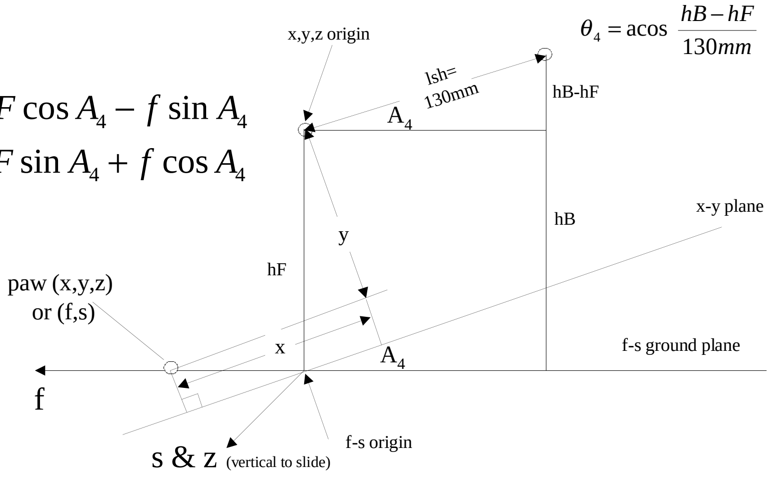
Projection on Ground Plane

$$\theta_4 = \text{acos} \frac{hB - hF}{130mm}$$

$$z = s$$

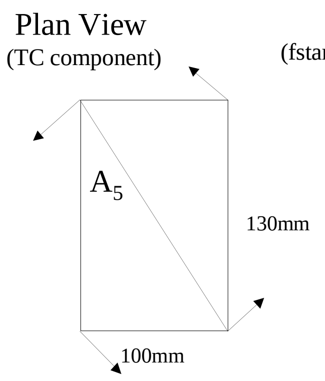
$$y = hF \cos A_4 - f \sin A_4$$

$$x = hF \sin A_4 + f \cos A_4$$

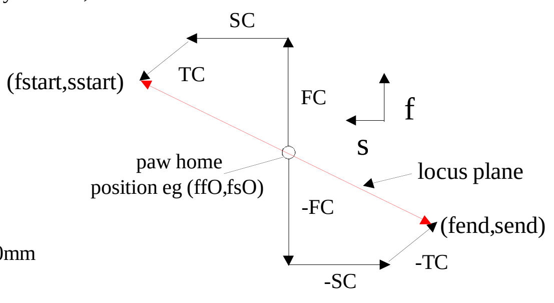


Start & End Positions of Locus on Ground

Control = forward FCmm, sideways SCmm, turn TCmm



$$A_5 = \operatorname{atan}\frac{100\,\text{mm}}{130\,\text{mm}}$$



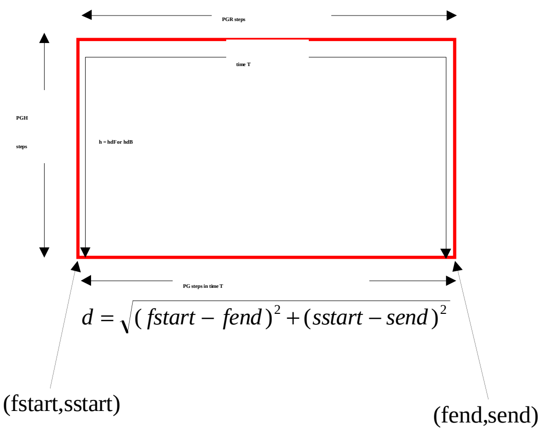
$$fstart = FC - TC\sin A_5 + ffO$$
$$fend = -FC + TC\sin A_5 + ffO$$
$$sstart = SC + TC\cos A_5 + fsO$$
$$send = -SC - TC\cos A_5 + fsO$$

Positions Around Paw Locus

$$\text{Distance per step around top} = \frac{2\text{h} + \text{d}}{\text{PG}}$$

$$PGH = \frac{PG.h}{2h + d}$$

$$PGR = \frac{PG.d}{2h + d}$$



$$d = \sqrt{(fstart - fend)^2 + (sstart - send)^2}$$