|  |
| --- |
| QUT – queensland university of technology |
| Khepera Robot Soccer |
| INB860 – Computational Intelligence, Control and Embedded Systems |
|  |
| ***Contributors***   |  |  |  | | --- | --- | --- | | Justin Eyles | justin.eyles@gmail.com | 05706343 | | Magne Gasland | magnemg@gmail.com | 06954596 | | Chris Savini | sav\_chris13@hotmail.com | 05741467 | | Stephen Vidas | s.vidas@student.qut.edu.au | 05751187 | |
| **5/18/2009** |

|  |
| --- |
| This report details the design and testing of code to control a Khepera robot, with the purpose of making it score a goal in various circumstances in a Khepera Robot Soccer Environment. The methodology used to achieve the group’s goals is outlined, as well as justification for decisions made throughout the process. Limitations of the group’s solution are discussed at the conclusion of the document. |

Table of Contents

[1 Introduction 3](#_Toc230420788)

[2 Achievement Summary 4](#_Toc230420789)

[2.1 Robot Performance 4](#_Toc230420790)

[2.2 Teamwork 4](#_Toc230420791)

[3 Team Contributions 5](#_Toc230420792)

[3.1 Justin Eyles 5](#_Toc230420793)

[3.2 Magne Gasland 5](#_Toc230420794)

[3.3 Chris Savini 5](#_Toc230420795)

[3.4 Stephen Vidas 5](#_Toc230420796)

[4 Methodology 6](#_Toc230420797)

[4.1 Overall Program Flow 6](#_Toc230420798)

[4.2 Image Capture and Processing 7](#_Toc230420799)

[4.3 Orientation of Robot and Direction of Goal 9](#_Toc230420800)

[4.4 Searching for Ball 10](#_Toc230420801)

[4.5 Estimate of Ball Direction and Distance 11](#_Toc230420802)

[4.6 Fuzzy Logic and Controller 12](#_Toc230420803)

[4.7 Chasing and Kicking the Ball 18](#_Toc230420804)

[4.8 Object Avoidance 19](#_Toc230420805)

[5 Performance Analysis 20](#_Toc230420806)

[5.1 Modular Testing 20](#_Toc230420807)

[5.2 Final Testing 23](#_Toc230420808)

[5.3 Competition Results 24](#_Toc230420809)

[5.4 Limitations 24](#_Toc230420810)

[6 Conclusion 25](#_Toc230420811)

# Introduction

Khepera robots are small, relatively inexpensive devices used primarily as an educational tool to teach students about robotics and Artificial Intelligence (AI). The most common use of these robots are in Khepera Robot Soccer. This project involved the design and implementation of a soccer-playing strategy utilizing principles such as Fuzzy logic and state machines.

A Khepera Soccer simulation package called KIKS was used for testing the effectiveness of code as it was written. This package was run via MATLAB, a high-level mathematical programming environment. The nature of MATLAB meant that although there were limitations such as in the use of interrupts, in general the programming syntax was very simple. This was consistent with the emphasis of the project being in the design of appropriate algorithms themselves, not their implementation.

Some of the major problems faced during the project included:

* Keeping track of the robot’s position in relation to the goal
* Locating the ball
* Moving to the ball
* Kicking the ball in the correct direction and
* Avoiding obstacles

The solutions developed to these problems are discussed in section 4 of the report.

# Achievement Summary

## Robot Performance

In the initial demonstration on Thursday 14 May 2009, the group’s robot passed 2 of the 3 tasks – one of the best results in the class. This was a very good result, given that the specific details of the tasks were unknown beforehand.

Aspects of the robot’s behavior that were considered great successes were the ball-finding and chasing behavior. The realignment of the robot in order to setup its shot at the goal was also performed well. Keeping track of the orientation of the robot and position of the target goal functioned excellently for the most part, but began to breakdown when the robot became jammed at a wall. This pointed to the main weakness of the robot’s programming – its inability to effectively avoid collisions with the wall and to push the ball out of a corner when the opponent or an obstacle was present. The main reason for this failure was that wall-avoidance behavior was only implemented late in the development process, and as a result there was insufficient time for testing and refining it.

Overall the group was very pleased with their achievements relating to the performance of the robot in undertaking these tasks, and were confident that they understood the reasons behind the shortcomings of the robot, and how to rectify these weaknesses.

## Teamwork

By utilizing an online data repository and versioning system, group members were easily able to communicate their individual progress, and access up-to-date code. This allowed much of the work to be done independently. However, regular group meetings were undertaken in which design decisions and aims were discussed and agreed upon. The only major difficulty encountered in terms of teamwork was finding an appropriate meeting time, given that all four members of the group had significantly different timetables associated with different courses and majors.

Again, all members of the group were very happy with how the team functioned, and useful experience in project management, group work and the use of repositories was gained.

# Team Contributions

## Justin Eyles

Justin’s main responsibility was writing the function to determine the ball’s direction and distance from the robot. He also worked on obstacle avoidance techniques and helped with some of the fuzzy logic and spinning functions.

## Magne Gasland

Magne was responsible for getting the robot to search for the ball, aligning the ball with the goal, and designing pseudo code for the fuzzy controller.

## Chris Savini

Chris was responsible for creating the reusable fuzzy controller, working on the state machine designs, activity diagrams and UML. He also worked on implementing the ball chasing functionality and setting up the team's online subversion repository.

## Stephen Vidas

Stephen’s principle focus was the development of the main function governing the total behavior of the robot. However, he also had the responsibility of developing functions to determine the orientation of the robot and the general location of the target goal. In addition Stephen was responsible for the image processing associated with the image capture of the robot. Small contributions to the chasing of the ball and the alignment of the ball with the goal were also made.

# Methodology

## Overall Program Flow

Several options were explored for designing an effective and versatile program flow that would ensure the regular scoring of goals by the Khepera robot. The most significant difference between these alternative strategies was the nature of the ‘grand loop’ which would be run upon initiating the program. One option was to have the loop run continuously, regardless of the state of the robot, however this was found to be inflexible. The final solution was to model the loop on the different stages of scoring a goal, only resetting when one of the stages failed.

In all explored designs, the first stage involved the robot locating the ball. Afterwards, the two main options were to either move towards the ball, or to reposition in order to align the ball better with the goal. The first option was preferred because in a competitive environment it was considered important to get close to the ball as quickly as possible, in order to prevent the opponent from gaining control.

Once the ball was reached, there were several options presented. Option 1 was to simply continue pushing into the ball and attempt to ‘dribble’ the ball into the goal. Option 2 was to encircle the ball until the alignment with the goal was correct, and then kick the ball into the goal. Option 3 was to encircle the ball and then dribble it into the goal. Option 4 would involve limiting the amount of encircling, and then attempting to kick the ball at an angle so that it could get into the goal without the robot having to reposition itself.

Although Option 4 was probably the fastest option, it was perceived to have much greater implementation problems associated with calculating the correct approach angle in order to deflect the ball into the goal. Option 2 was selected because it seemed to be a good compromise between speed and simplicity.

After kicking the ball towards the goal, the robot would then reenter the main loop, in order to ensure that if the shot missed, or was blocked, the robot would attempt to score again.

Apart from this main flow, several other features had to be built in to the main program. These included object avoidance, and wall avoidance. Both these features were included throughout the program – constant checks were in place to ensure that if an object or wall interfered with an action (such as encircling or shooting), it was taken into account.

If there was more time, the implementation of more advanced behaviors such as defending against opposition shots, stealing the ball, and avoiding the opponent would have been attempted.

The main function which controlled the overall robot behavior was called ManUnited(). The basic operation of this function (as outlined above) is illustrated in the Flow Diagram shown in Figure 1 below:

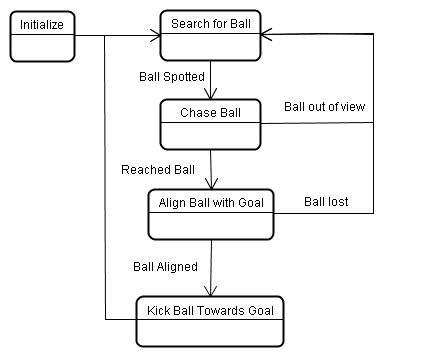


Figure : Main Function Flow Diagram

## Image Capture and Processing

The function UpdateGlobals() (outlined in more detail in section 4.3), is called regularly, and includes the capturing of an image from the Khepera’s camera. The image is 1 x 64 pixels resolution grayscale, and because each of the objects and surfaces in the environment has a unique shade, it is possible to determine whether the robot is viewing either the goal, the wall, the ball or an obstacle or opponent, or any combination of these.

Several algorithms were explored in order to categorize each part of the image as either goal, wall, obstacle, or goal. Given that the boundaries between the objects are difficult to determine, some inaccuracy was to be expected regardless of the algorithm. However, it was considered very possible to achieve full accuracy in determining the presence of any of the four entities.

Each candidate algorithm took a pixel by pixel approach, and the two main options involved analyzing the intensity of pixels in the immediate neighborhood as well. The first option required all 4 pixels surrounding any pixel (two to the left and two to the right) to also be within the predefined limits of a category, before the centre pixel would be categorized. The second approach only required the average of the 5 pixels to be within the predefined limits. In both cases the predefined limits were identical, as shown in Table 1 below:

Table : Image Classification limits

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | First Method | | Averaging Method | |
| Entity | Lower Limit | Upper Limit | Lower Limit | Upper Limit |
| Ball | 151 | - | 151 | - |
| Wall | 116 | 150 | 116 | 150 |
| Obstacle/Robot | 91 | 115 | 91 | 115 |
| Goal | - | 90 | - | 90 |

Both methods returned very similar results, so the averaging method was chosen because it was simpler to implement in code. It was also believed, although unproven, that it might be slightly more accurate in determining the boundaries between entities.

The figures below are two scenarios which are used to demonstrate the effectiveness of the group’s image processing algorithms:

|  |  |
| --- | --- |
| im1_setup.JPG  Figure : Image 1 (see ) | im2_setup.JPG  Figure : Image 2 (see Figure 4) |

The two left-hand subplots in Figure 4 below show what the raw image captured from the robot looks like in both scenarios. The yellow circles outline the presence of the ball in the image, while the red circles outline the presence of an obstacle or enemy robot. The green circle in the top-left image is the goal, in front of which the obstacle or opponent is positioned.

The right-hand images show the output of the EncodeImage() function, which successfully identified the presence and general position of all the entities in both scenarios. More extensive testing was undertaken, and it discussed briefly in Section 5.1.1.

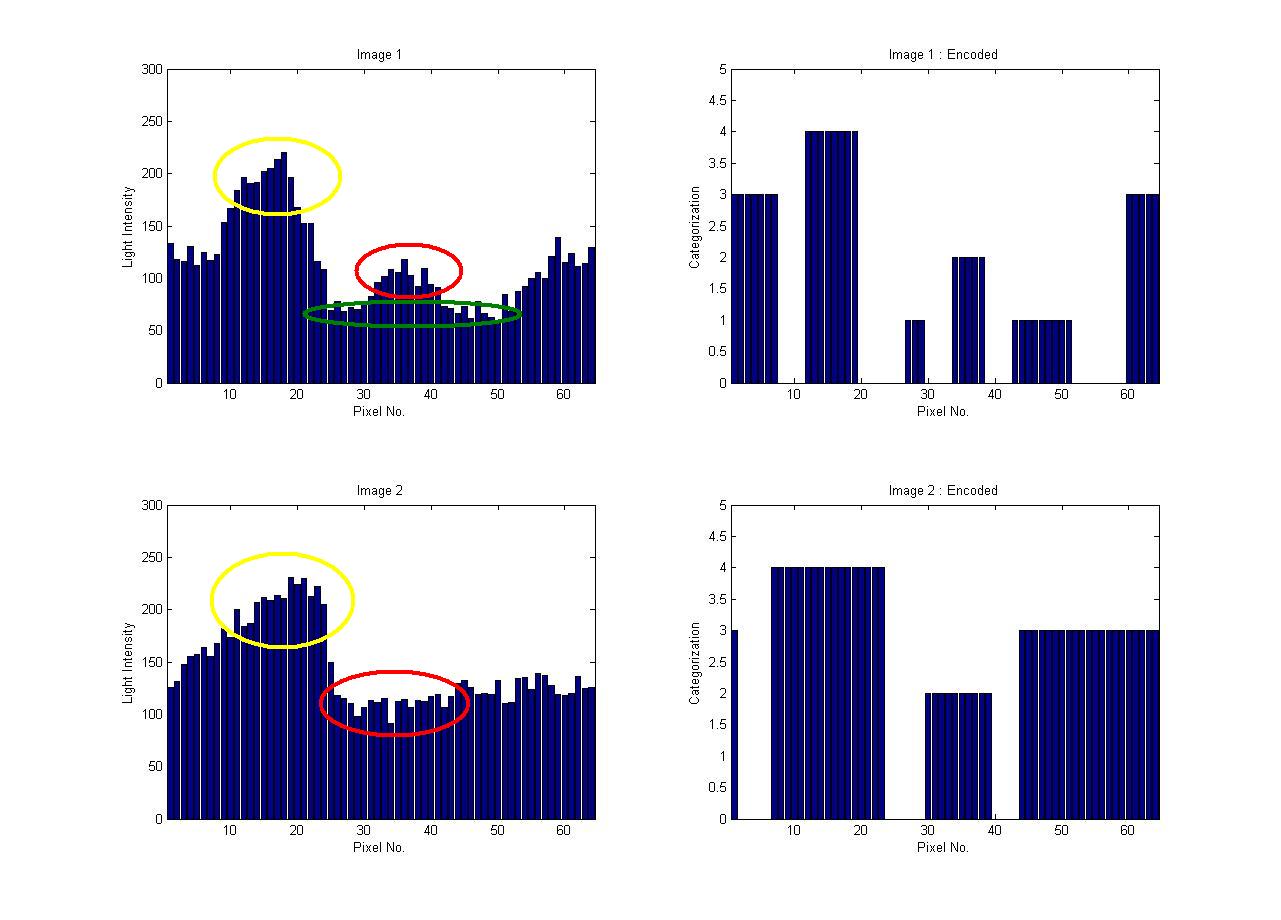


Figure : Effectiveness of Image Encoding Algorithm

## Orientation of Robot and Direction of Goal

The nature of MATLAB as a language and as a programming environment meant that the use of interrupts was impractical. As a result, other options had to be explored for ensuring that data from the robot’s actuators was read in regularly. The function ‘UpdateGlobals()’ was developed with the purpose of reading in data from the robot, and updating the global variables accordingly. In addition, the function would process the image from the robot’s camera, and encode it to allow other functions to access the most up-to-date image at all times. This function was called during any loop (occurring in any function) that was capable of lasting more than a few milliseconds. It effectively operated as an irregularly-called interrupt.

The Wheel encoders were used (in the function EstimateOrientation()) to obtain an estimate of the angle of orientation of the robot, with zero degrees corresponding to facing due east (with the target goal north) and 90 degrees corresponding to due north. However, depending on the position of the robot, the angle itself may not indicate whether the goal is to the left or right of the direction the robot is facing. To achieve this knowledge, the updated image from the robot’s camera was used to track when the goal left the field of view, and in what direction that happened. Eight orientation states were defined, as shown below in Figure 5.

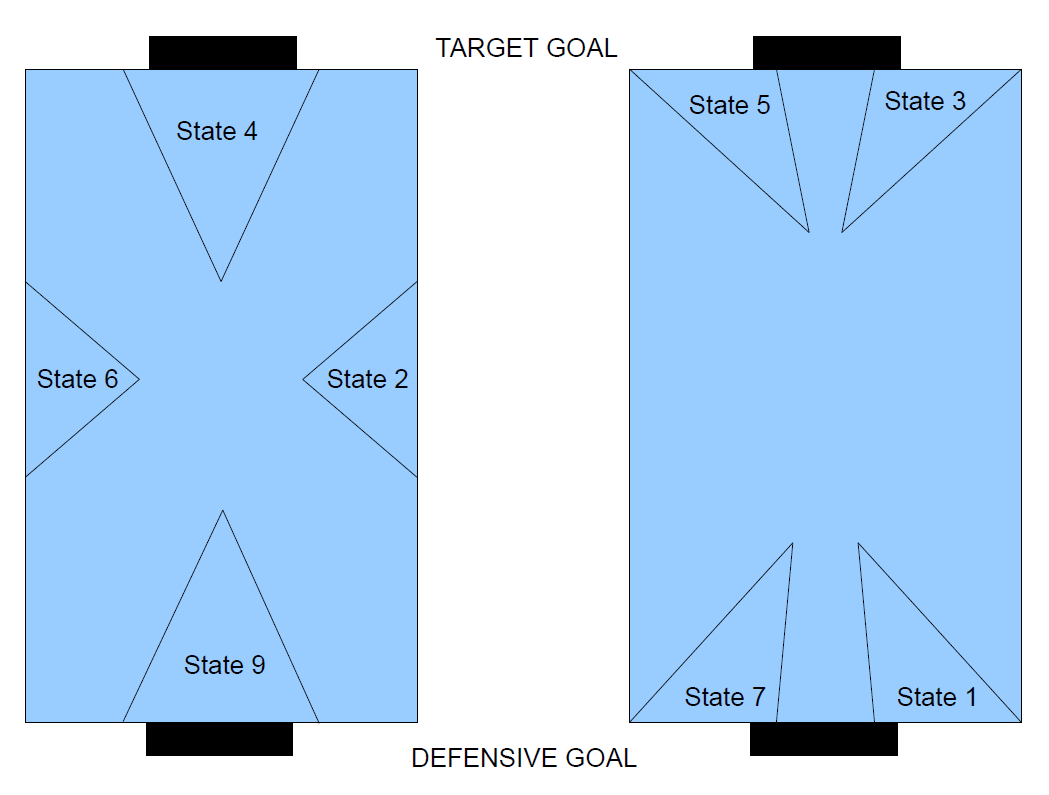


Figure : Robot Orientation States

These orientation states were important for knowing what way to approach the ball when the target goal was not in view. Additionally, if the wheel encoders began to get out of synchronization, they could be reset with some approximation when the robot was in a more restrictive state such as State 4 or State 9.

## Searching for Ball

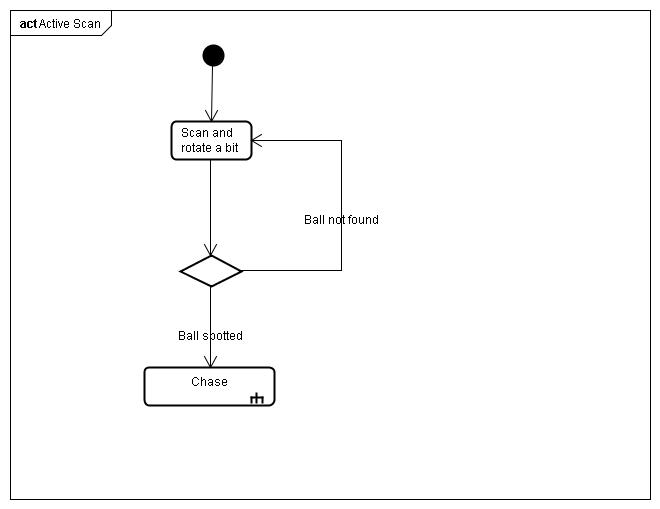


Figure 6

If a full rotation is made and the ball is not found, the robot moves in a random direction and scans again. This helps the robot locate the ball when it is hiding behind an obstacle.

## Estimate of Ball Direction and Distance

The BallDirection() function was written to calculate the direction and distance of the ball from the robot. The function also indicated whether the ball was actually in view or not. BallDirection() was used in locating the ball and moving to the ball

To calculate the direction, firstly the centre of the ball was located. This was done using the image processing functions. Once the centre of the ball was found, the direction of the ball was calculated (using the number of pixels the centre of the ball was away from the centre of view, 64 pixels = 36 degrees).

To estimate the distance the ball was from the robot, the relative size of the ball in the image was used. By experimentation using the inbuilt kiks function interrogate(), a relationship between the distance and the size of the ball in view (number of pixels it occupied), was developed.

Many screen shots were taken to develop this relationship. Below, Figure 6 shows one such image capture.



Pixel Location

Encoded Image

Figure

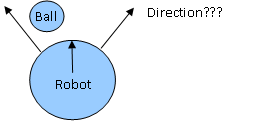
As seen from Figure 6, the number of pixels occupied by the ball was 23 in this case. Using interrogate(), the distance was found to be 270mm. This was repeated for many different distances.

By compiling all of the results, an estimated distance function was established whereby:

*Distance*=

## Fuzzy Logic and Controller

There are a number of problems in creating a fuzzy controller which can not be easily solved analytically. For example, to push the ball forward, the robot needs to be able to constantly adjust its speed and direction so that it does not lose the ball.



Figure

It is not easy to decide how to do this, so we used a fuzzy controller to solve the problem. The price that we pay for this is that more development time is needed to 'train' the controller to behave appropriately.

The controller classifies quantities in to categories with a certain fuzzy probability.

For example a ball distance can be classified as being a large distance away (LP), a moderate distance away (P), or close (Z). So distance is a member of the set {LP, P, Z}

Since these sets are fuzzy, a distance can have a certain probability of belonging to a particular set.

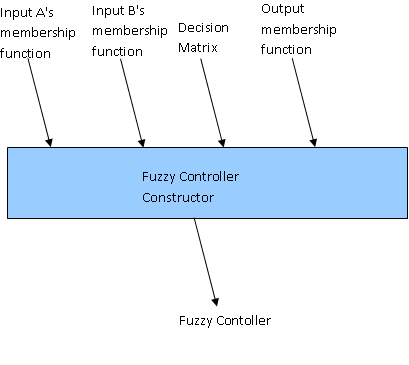
This is calculated from some arbitrary probability distribution which we can tweak as part of the training process.

I.e.: given a distance D

Pr(LP|D) = a value calculate from the probability distribution.

The success of using a fuzzy controller is highly dependent on how well we can train it. For this reason, we tried to make the fuzzy controller as general as reasonable. Our controller performs a general classification for two inputs (A and B). (For example, goal angle and ball angle).

The decision matrix is a lookup table used to map inputs to an output using Mandani's method (the values in the decision matrix can be altered as part of the training process).



Figure

Note that the fuzzy controller constructor expects FUNCTIONS as inputs (we use function currying to do this). The advantage of this is that it allows the user to define their own custom membership functions. This gave us greater flexibility in the training process.

To further speed up the training process we created function generators. Ie: function which we could used to easily create our membership functions. For example, to create a membership function that looks like the spike below, spike = fuzzyMember(h, w, d);

Where h: height w: width d: center point

y = spike(x)

% Spike

% y ^

% |

% |

% |

% |

% | 0

% | / | \

% | / | \

% | / |h \

% | / | \

% |-----------0-----X-----0---------------->

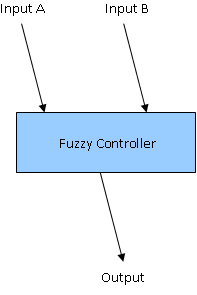
% |- w| -| x

% |

% |

% d

Now that we have created a fuzzy controller function for our particular purpose, we can reuse it.

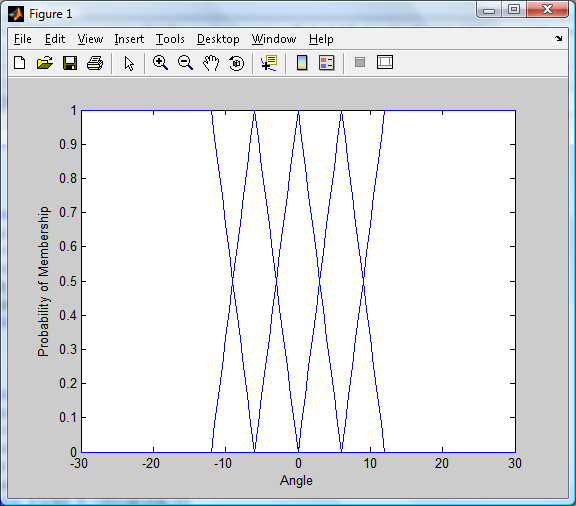


Figure

The fuzzy classifier for chasing the ball works in two stages. First, the robot looks at the image of the ball and estimates the distance using an exponential model which we fit from experimental data.

The robot uses this distance figure to classify the ball distance into one of three possible fuzzy values (Large distance, moderate distance, close). The robot constructs a fuzzy controller based on this estimate. There are three corresponding fuzzy classifiers which can result from this process.

The difference is in the decision matrix which is used.



Figure

The second step is to input the estimated angle to the ball and angle to the goal into the fuzzy controller. The centroid computed from the controller is then used to update the wheel rotation.

These membership functions are used for both goal distance and ball distance.

**Output membership function**

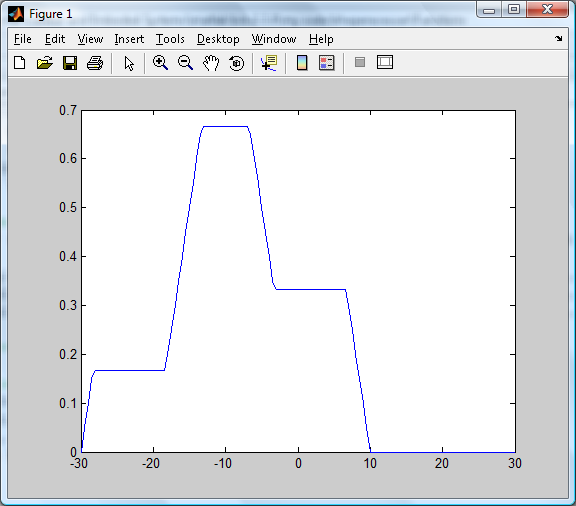
for example:

Using a membership function called largeP (for when the ball is a large positive distance away from the robot), a goal angle of 2 and a ball angle of 7, a centroid is calculated.

largeP(2, 7)

ans =

-8.4994



Figure

The output is the calculated as the centroid of the above graph.

The decision plane for large distances shown from a few different perspectives

|  |  |  |
| --- | --- | --- |
| Figure | Figure | Figure |

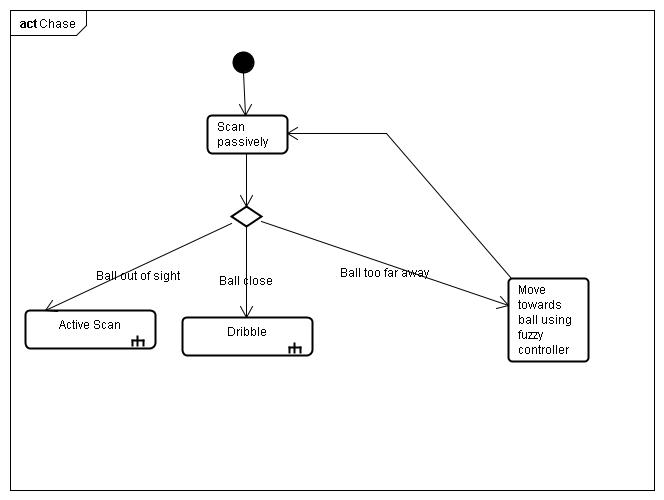
Decision plane for positive distances

|  |  |
| --- | --- |
| Figure | Figure |

Decision plane for small, near zero distances

|  |  |  |
| --- | --- | --- |
| Figure | Figure | Figure |

## Chasing and Kicking the Ball



Goal

Figure

The ball angle and goal angle are measured relative to the robot's orientation as shown in the above diagram. The robot uses the fuzzy controller to decide what action to take based on the ball angle, goal angle and ball distance.

The first step is to decide how far away the ball is and classify it as being a member of the fuzzy set {Zero distance, medium distance, large distance}. The robot will use a different decision matrix for each of these fuzzy distances. The decision matrices are used to map a goal angle and a ball angle to a direction.

Key:

LN = Large negative

N = Negative

Z = Zero

P = Positive

LP = Large Positive

For example, when the ball is a large distance away, the decision matrix is

Ball Angle

Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | LN | N | Z | P | LP |
| LN  Goal Angle | LP | Z | N | N | LN |
| N | LP | Z | N | LN | LN |
| Z | LP | P | Z | N | LN |
| P | LP | LP | P | Z | LN |
| LP | LP | P | P | Z | LN |

The output corresponds to the right wheel speed.

## Object Avoidance

In the soccer task, the robot was required to kick the ball into the goal without kicking the opposing robot. To do this effectively, obstacle avoidance techniques were required to miss the other robot. It was also desirable to have some techniques to prevent the robot from running into the wall.

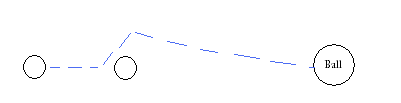
During the project, three scenarios were considered:

* 1. The opposing robot completely obstructed the view of the ball
  2. The opposing robot was partially in between the ball and our robot
  3. The ball was close to the wall

To compensate for the first scenario, code was written, so that if the robot could not see the ball in a full 360 degree spin, but could see the opponent, then the robot would move slightly and look for the ball again.

Unfortunately, due to time constraints, code was not implemented to overcome scenario 2 and 3. However, these scenarios were contemplated.

To overcome scenario 2 it was proposed that the robot should be able to deviate around the opponent when chasing the ball. In this case if the robot was chasing the ball, and came into close proximity to the opponent, it would simply turn and move out of the way of the opponent before chasing the ball again as seen in Figure 21.



Figure

For situation 3, one solution would be to simply follow the wall until the ball was close to the goal. This solution would prevent the robot from trying to move into the wall, making the operation more efficient and allowing the robot to steer the ball towards the goal.

# Performance Analysis

During the project each program was tested individually and then integrated into the entire system. This section gives a brief description about how each function was tested and then finally how the whole system was tested.

## Modular Testing

### Image Capture and processing

As the image processing was a core element of the overall system, extensive testing was carried out on the component EncodeImage(). The purpose of the testing was to ensure that the robot could differentiate between the wall, goal, ball and opposing robot.

Many different situations were set up in the arena (with different combinations of views – i.e wall, ball, opposing robot). Image screen shots were taken of all of these situations to ensure that the image was being encoded correctly.

During this process, the encoding algorithm was changed to improve the quality of the image signal. The final solution used an averaging process to classify what was in view. Figure 22 below demonstrates how the algorithm effectively categorized all entities in the image according to the following system:

GOAL = 1;

ROBOT/OBSTACLE = 2;

WALL = 3;

BALL = 4.

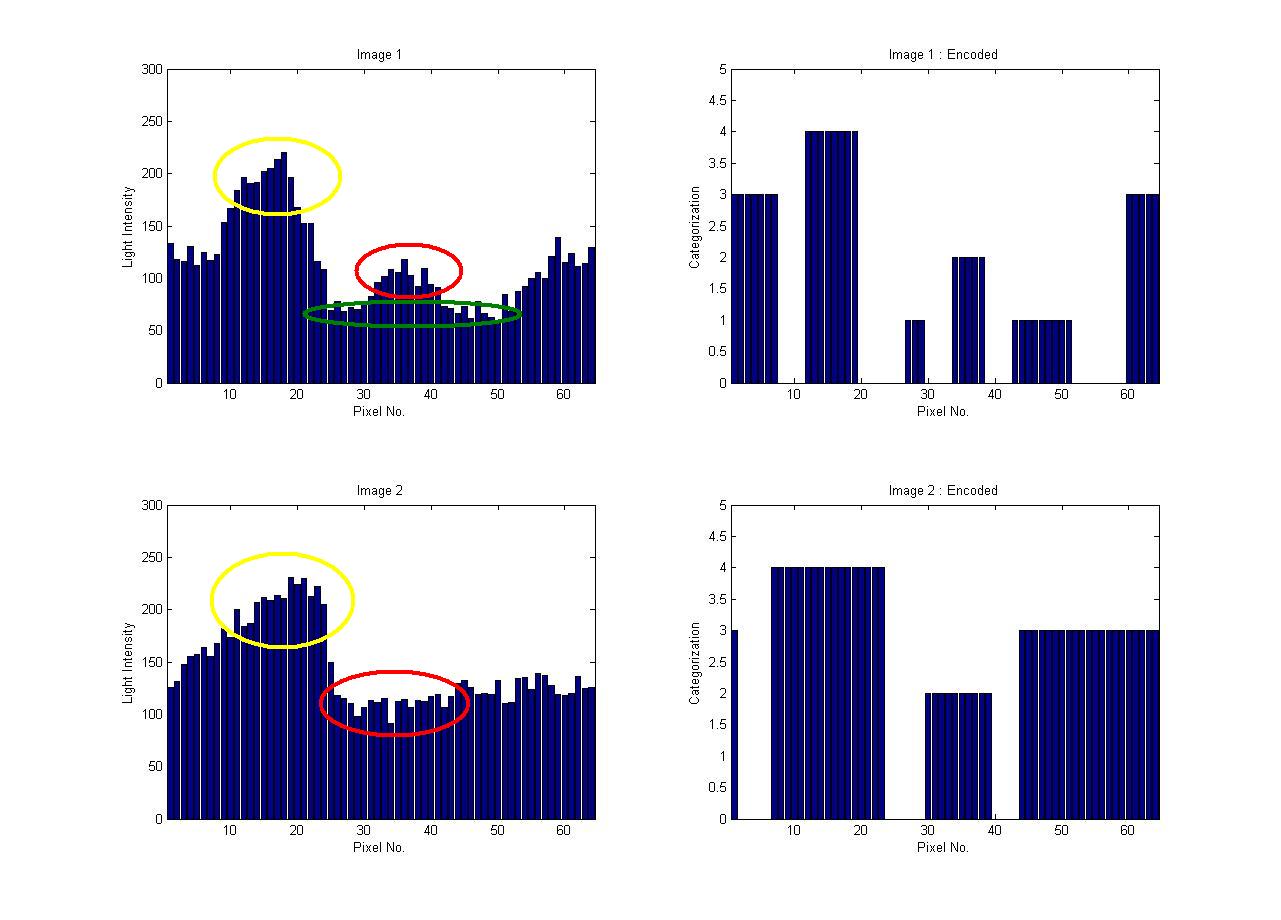


Figure : Effectiveness of Image Encoding Algorithm

### Orientation of Robot and Direction of Goal

In order to test that the estimate of orientation was accurate at all times, the disp() function was used to display the estimate while the function ManUnited() was running, and this was compared in real time with the true value read directly from the simulator. It was found that the estimate was accurate at all times, except when the robot became stuck and the wheels began spinning without any corresponding motion. This was seen as a non-issue because once the robot becomes stuck, the estimate of the orientation is worthless.

just a bit in here about how the disp() function was used to output constant orientation and orientation state values. Initially tested when just spinning around, later tested with more complex behavior. Time constraints meant that it was impractical to complete comprehensive testing before using this function in other functions.]

Both the estimate of the orientation and the orientation state functionality were tested first with simple behavior such as simply spinning in place. After their performance was determined to be effective, more complex behavior was undertaken with obstacles present in the environment. Both functions maintained their effectiveness under these tougher conditions.

### Searching for Ball

Within the search function, firstly the spin function was tested. After it was determined that the robot was spinning correctly, the search function was tested.

This test showed that the function was operating correctly, and that the robot would spin around until the ball was in completely view.

During testing it was determined that slowing the robot’s turning speed, increased the robot’s ability to see the ball correctly.

### Estimate of Ball Direction and Distance

To test the ballDirection() function, several setups were made in the arena, with the robot being placed a different distance from the ball. The distance indicated using the kiks interrogate() function was compared to the estimated distance. These results can be seen in Figure 22.

|  |  |
| --- | --- |
| Distance  Number of Pixels Ball Occupies  Figure | KEY:  ---------- Estimated Distance  ---------- Experimental Distance |

As can be seen from the figure, the estimated distance was quite accurate. Several tests were also done to ensure that the function was identifying that the ball was in view and a reasonable angle was being computed.

### Fuzzy Logic and Controller

The fuzzy controller was tested by comparing its output to numbers which we computed by hand. We also compared its output to the practical exercise on fuzzy controllers. The results were consistent.

### Chasing and Kicking the ball

During testing it was shown that the robot could move to the ball as anticipated.

However, it was found that once the robot reached the ball, the view of the goal was often obstructed. This caused problems with other parts of the code.

Hence, it was decided that when the robot reached the ball, it would kick the ball and then go and search for it again.

### Object Avoidance

To test the obstacle avoidance function, some simulations were carried out with the robot, ball and an opponent (cylinder) in the arena. Code was successfully implemented so that when the ball was completely blocked by the opponent, the robot would move a small distance, and then search for the ball again.

## Final Testing

Once the entire system was integrated, overall system performance was tested. The program was reasonably successful, and goals could be scored in many situations. Many of the main problems that were identified in the introduction were solved by the program. For instance, in most situations the robot could:

* Identify its position in relation to the goal
* Locate the ball
* Move to the ball
* Line the ball up with the goals
* Kick the ball
* Score a goal

|  |  |
| --- | --- |
| robot1.bmp  Figure : Robot Path without Obstacle | robot2.bmp  Figure : Robot Path with obstacle |

## Competition Results

At the time of writing, the competition has not yet been carried out. During the first testing session however, the group successfully completed two out of the three tasks, which was considered to be a very good result.

## Limitations

Although the program was quite successful there were some limitations to the program as described below.

1. **Vision of Ball**

If the program was required to run for a long period of time, sometimes the robot would slightly lose track of its position in relation to the goal. Hence, the robot would be trying to kick the ball in an incorrect direction. This only occurred however after the program had been running for more than approximately two minutes.

If more time was available, it would be desirable to solve this problem. This could possibly be done by closer calibration of the wheel encoders or by some means of resetting the encoder values at appropriate times.

1. **Vision of Ball**

Occasionally during testing, it was noticed that robot did not see the ball when it should have. The reasons for this were unsure.

The turning speed was decreased during testing, and this was found to improve the performance. However, further improvements could be made possibly by slowing the robot further.

1. **Time to Line Up Ball and Goal**

Lining the ball up with the goal occasionally proved to be quite a time consuming process. Possible improvements in this area could be made by adjusting the path that the robot moved towards the ball.

Efforts were made in the fuzzy controller to have the robot approach the ball from the appropriate side. The aim of this was to in essence, line the ball up with the goal before reaching the ball.

Difficulties arose however, when the robot approached the ball as the goal became obstructed from view due to the ball.

Efficiency could be enhanced if the path the robot took to reach the ball could be improved.

1. **Obstacle Avoidance**

Significant improvements could be made to the program by enhancing the obstacle avoidance techniques. Some desirable attributes of the obstacle avoidance would be to follow the wall when the ball was close to the wall, and to deviate around an opponent when chasing the ball.

1. **Practical applications**

Unfortunately, the group did not get to implement the program on the real Khepera robot. Practical obstacles such as wheel slip, and vision imperfections were not compensated for in the code.

Testing would be required to calibrate the program to account for these issues.

# Conclusion

The purpose of the project was to develop a program to operate a Khepera robot in a game of soccer.

Overall, the project was very successful. The program developed, successfully solved most of the issues that were identified including:

* Keeping track of the robot’s position in relation to the goal
* Locating the ball
* Moving to the ball
* Kicking the ball in the correct direction and
* Avoiding obstacles

The robot was found to be able to score a goal in most situations.

If further studies were to be pursued however, the efficiency of the operation could be improved. In particular, improvements would be desirable in obstacle avoidance and finding the optimal path for the robot to follow.