THE UNIVERSITY OF NEWCASTLE
AUSTRALIA

NUbots: Enhancements to Vision Processing, and Debugging Software for Robocup Soccer

Kenny Hong

A thesis submitted in partial fulfilment of the requirements for the degree of Bachelor of Engineering in Computer Engineering at The University of Newcastle, Australia.

2 November 2005

ABSTRACT

'Enhancement to Vision Processing and Debugging Software for Robocup' is a project dedicated to the interesting field of robotic intelligence that is focused on competitive soccer held by Robocup. Robocup is an international robotic soccer competition held yearly to promote innovative research and application in robotic intelligence. NUBots is a team of developers at the 'University Of Newcastle' that have competed in the Four-Legged League of Robocup since 2002.

The proposed enhancements to vision processing are object recognition and the prediction algorithm.

Object Recognition is the recognition of field objects such as the ball, the goals, the beacons and other robots on the field. Beacons are the posts around the edges of the soccer field which allows the robots to determine its position, and the robots are the soccer players.

Prediction intends to evaluate future positions of moving field objects such as the ball and robots. The prediction algorithm utilises curve interpolation rather than linear interpolation for calculating the predicted position.

To date, beacon recognition has been implemented with good results; robot recognition gave good performance on detecting red robots, with added feature of determining their orientation. The recognition of blue robots requires further testing and collaborative work with colour classification. Rotation and filtering algorithms have been developed to ease the recognition code. The horizon line has been derived with offset variables. Horizon line is the line created at infinity where the soccer field seems to disappear from the camera's perspective based on the height of the camera. The horizon is important for rotation and filtering algorithm to work. On the other hand, Prediction is performing as required with curve interpolation giving favourable results than linear interpolation.

The proposed debugging software is the 'World Model Debugger'.

The 'World Model Debugger' (aka WMD) visually shows the current object location from localisation, predicted positions of moving field objects, and currently transmitted behaviour. An external camera which oversees the field provides the actual locations for comparison.

The WMD has being completed and tested. It is capable of drawing the ball and robots, interpreting different packet formats, and has integrated well with the external camera and the prediction algorithm.

ACKNOWLEDGEMENTS

I would like to give thanks for the following people who gave me joy in working on this project through good times and rough times:

Steven Nicklin
Naomi Henderson
Phavanna Douangboupha
Michael Quinlan
Timothy Moore
Stephen Young
Robin Fisher
Dr. Stephan Chalup

And of course my supervisors

Prof. Rick Middleton
Dr. Lu Gan

It has been fun.

Cheers

Kenny Hong

LIST OF CONTRIBUTIONS

The key contributions which I made towards this project are as follows:

- Derived, implemented and tested the Horizon Line.
- Proposed, design, implemented and tested the rotation algorithm. Includes modifying NUBot's Vision Debugger called 'EOTN' to display the rotated blobs.
- Proposed, design, implemented and tested the filtering algorithm.
- Design, implemented and tested the beacons recognition code.
- Design, implemented and tested the robots recognition code. Includes proposing, designing, implementing and testing orientation recognition; and modifying NUBot's Vision code for scanning of classified image using the rotation algorithm for the beacons and the robots recognition code.
- Proposed, design, implemented and tested the prediction algorithm.
- Proposed, design, implemented and tested a new 'World Model Debugger' application. Includes modifying NUBot's UDP Packet code to send out WMD Packets. Excludes the design and implementation of interfacing with the external camera and the camera feed dialog.

Signed by:

Student: _____

Kenny Hong

Supervisor: _____

Dr. Lu Gan:

TABLE OF CONTENTS

1. INTRODUCTION 1

1.1. About Robocup 1
1.2. The Sony AIBO ERS-7 Robot 1
1.3. The NUbots 3
1.4. Project Goal 4

2. SCOPE 5

3. DESIGN & IMPLEMENTATION 6

4. HORIZON LINE 7

4.1. The P-Plane and the H-Plane 7
4.2. Derivation of the P-Plane 8
4.3. Derivation of the H-Plane 13
4.4. Performance 15

5. ROTATION ALGORITHM 16

5.1. Rotation Maths 16
5.2. Inverse-Rotation Maths 17
5.3. Benefits 18
5.4. Performance 19

6. FILTERING ALGORITHM 20

6.1. Supported Scenarios 20
6.2. Performance 21

7. BEACON RECOGNITION 22

7.1. Initial Code 22
7.2. Evolution I 23
7.3. Evolution II 23
7.4. Evolution III: Current 24
7.5. Forming Beacon Field Objects 24
7.6. Non-Ideal Beacon Blobs 25
7.7. Colour Classification Affects Distance 26
7.8. False Beacons 26
7.9. Performance 27

8. ROBOT RECOGNITION 28

8.1. ROBOT BLOB CLUSTERS 28
8.2. ASSIGNING ROBOT BLOBS TO ROBOT CONTAINERS 28
8.3. ORIENTATION RECOGNITION 30
8.4. FORMING ROBOT FIELD OBJECTS 34
8.5. PERFORMANCE 34

9. PREDICTION ALGORITHM 35

9.1. KALMAN FILTER 35
9.2. LINEAR INTERPOLATION 36
9.3. CURVE INTERPOLATION 37
9.4. PERFORMANCE 39
9.5. FUTURE WORK 39

10. WORLD MODEL DEBUGGER 40

10.1. GRAPHICAL USER INTERFACE 41
10.2. OPENGL 45
10.3. UDP PACKET 45
10.4. EXTERNAL CAMERA 46
10.5. PERFORMANCE 47
10.6. ENCOUNTERED PROBLEMS 47
10.7. FUTURE WORK 47

11. REFERENCES 49

12. APPENDIX A A-1

12.1. WMD PACKET FORMAT A-1

13. APPENDIX B B-1

13.1. TEAM MEMBERS B-1
13.2. SUPERVISION B-1
13.3. COMMERCIAL CONFIDENTIALITY AND INTELLECTUAL PROPERTY B-1
13.4. EQUIPMENT AND CONSUMABLES B-1

LIST OF FIGURES

Figure 1-1: Specification of Sony AIBO ERS-7. [3] 2
Figure 1-2: Vision is processed from images given by sensors and the camera.s 3
Figure 1-3: Sequence of the vision component. 4
Figure 4-1: Picture Plane intersecting with Horizon Plane 7
Figure 4-2: Four Corner Points within Camera Frame 8
Figure 4-3: Calculating f on cF axis 9
Figure 4-4: Introduction of Snout, Camera angle offset, and Snout angle offset 10
Figure 5-1: Blob Rotations 17
Figure 5-2: Benefits of Rotation Algorithm 18
Figure 6-1: Purpose of the filtering algorithm. 20
Figure 7-1: Location of beacons, and their colour combinations. 22
Figure 7-2: Further away Beacons 25
Figure 7-3: Projection Of Beacon Blobs 26
Figure 8-1: Scanning of Classified image for robots 29
Figure 8-2: Robot Red Recognition and Orientation. 31
Figure 8-3: Robot Blue Recognition and Orientation. 33
Figure 9-1: Linear Interpolate versus Curve Interpolate. 36
Figure 9-2: Curve Interpolation Process ‘Angle difference calculation’ 37
Figure 10-1: WMD Graphical User Interface. 41

1. INTRODUCTION

1.1. About Robocup

Robocup is an international robotic soccer competition held yearly to promote innovative research and application in robotic intelligence. The initiative of Robocup was “By 2050, develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer.” [1]

The Four Legged League is a sub division of Robocup that uses Sony AIBO’s as soccer players to compete in the competition.

1.2. The Sony AIBO ERS-7 Robot

At present, the competition uses Sony’s AIBO ERS-7 Entertainment Robots, which are equipped with the necessary hardware to compete. Their specifications are as follows:

CPU	64-bit RISC Processor @ 576 MHz
RAM	64 MB
Media	Sony ‘Memory Stick™,
Image Input	350,000 – pixel CMOS image sensor
Audio Input	Stereo microphones
Audio Output	Speaker 20.8mm, 500mW
Integrated Sensors	Infrared distance sensors Í 2 Acceleration sensor Vibration sensor
Input Sensors	Head sensor, Back sensor, Chin sensor, Paw sensor
Wireless LAN function	Wireless LAN module (Wi-Fi certified) Internal standard compatibility: IEEE 802.11b/IEEE 802.11 Frequency band: 2.4 GHz Wireless channels: 1 – 11 Modulation : DS-SS (IEEE 802.11 – compliant) Encryption : WEP 64 (40 bits), WEP 128 (104 bits)
Movable Parts	Head – 3 degrees of freedom Mouth – 1 degree of freedom Legs – 3 degrees of freedom Í 4 Ears – 1 degree of freedom Í 2 Tail – 2 degrees of freedom

* Sub-table of [2]

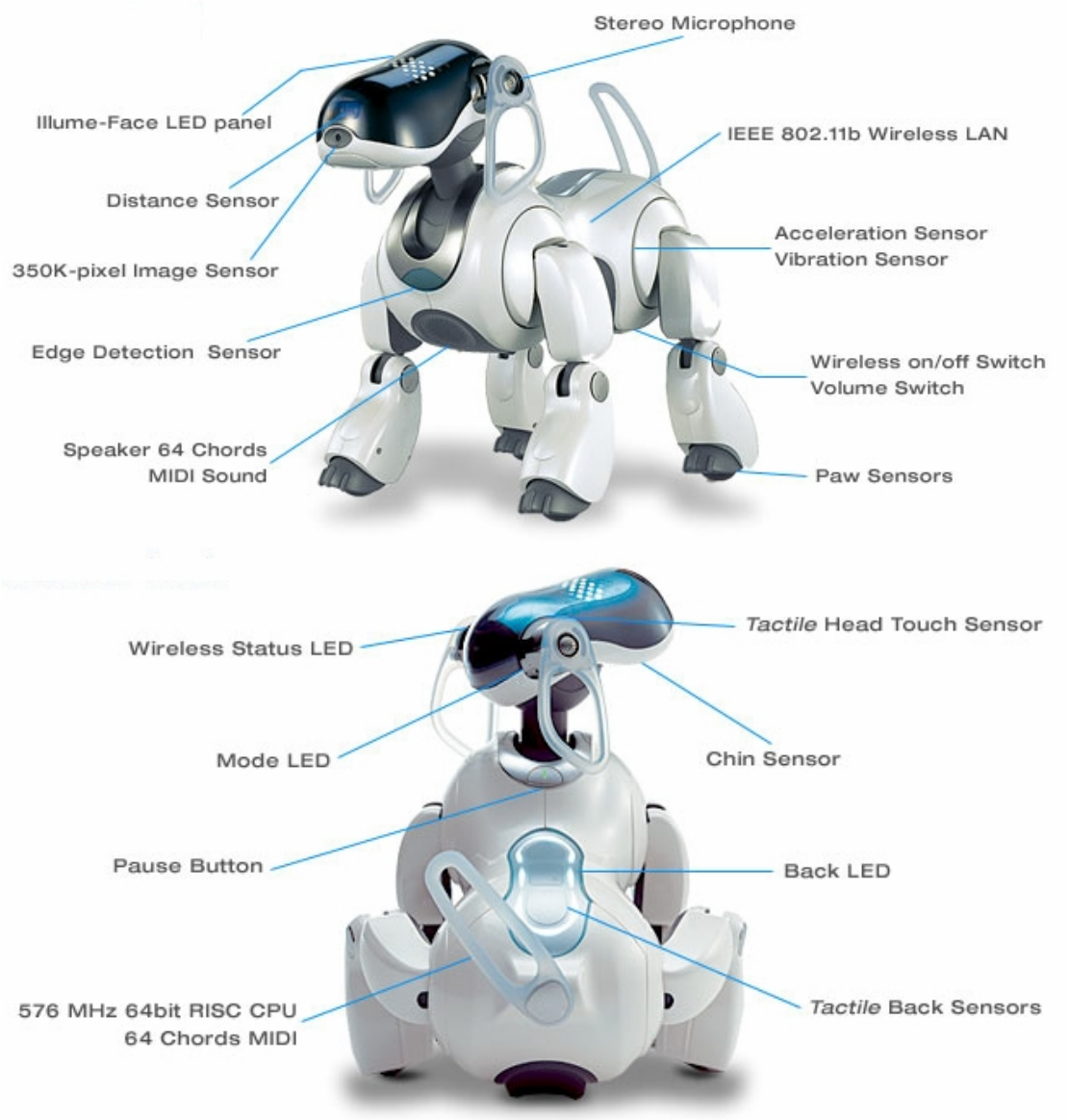


Figure 1-1: Specification of Sony AIBO ERS-7. [3]

The programming side is the fundamental core where applications of robotic intelligence is implemented, and always under constant development.

1.3. The NUbots

NUbots is a team of developers at University Of Newcastle that have competed in the Four-Legged League of Robocup since 2002. The name NUbots stands for '**N**ewcastle **U**niversity ro**bots**'. [4]

Vision is one major programming component that feeds information of the environment by the camera located at the snout of the robot. The environment in this case is the soccer field, and the information is field objects such as the ball, the goal, the beacons and other robots on the field.

Vision works along side with localisation and behaviour. Localisation is the component responsible in using vision data to calculate an object's location. The location data is then updated in a mathematical representation of the soccer field called the World Model. Behaviour uses information within the World Model to determine its next set of possible actions to achieve the best outcome. If one such action requires movement of the robot, then behaviour would call the locomotion engine. The component of vision is illustrated in Figure 1-2.

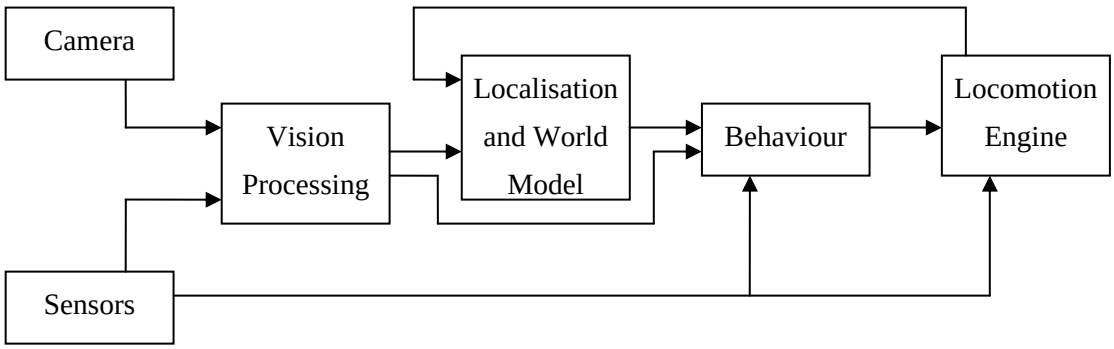


Figure 1-2: Vision is processed from images given by sensors and the camera. The output of vision provides data regarding the object location and then changes states in the world model. Behaviour uses object locations to determine next set of actions, which may call the locomotion engine to coordinate the appropriate movements. This figure is taken from [5]

The details of vision requires outside images to be streamed from the camera, then classified using colour classification [6]. Blobs are formed around patches (or clusters) of classified images, and then they are rotated and filtered before reaching object recognition code. This is illustrated in Figure 1-3.

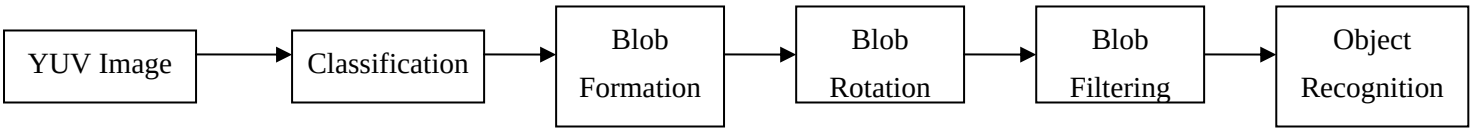


Figure 1-3: Sequence of how an output image reaches object recognition within the vision component.

1.4. Project Goal

The goal of this final year project is to design and implement code for recognising field objects such as the ball, the goal, the beacons and the robots; provide some prediction, and create a debugger application that aids in the recognition and prediction process.

2. SCOPE

The need for ball and goal recognition is clear, as the aim of the game is to drive the ball towards the goal using clever strategies. Recognition of beacons is important for the robots to know where its own position is on the soccer field; relating to localisation. Recognition of robots is important to avoid collision, and it permits better game play using new behaviours that utilise other robot positions.

Prediction is having the ability to know the future location of an object, given sufficient data gathered at the present time. Prediction is useful as it enforces better game play, which once again, relates to behaviour.

The purpose of a ‘World Model Debugger’ is to aid in object recognition and prediction. This is achieved by drawing the recognised objects given their positions and their predicted positions. The gathered information is logged to be used by software developers to help debug problems in vision, behaviour, localisation and world modelling.

In all cases, particularly for recognition and prediction, the code should consume minimal computation as this is a real time application. Excessive computation would slow down the robot’s next set of actions, which would affect game play.

3. DESIGN & IMPLEMENTATION

The order of design and implementation discussed from this point forth is determined closely by the history of the project and the best flow of presenting the approach in tackling object recognition, prediction and the debugger application. They are as follows:

SECTION	COMMENT
Horizon Line	Explored during the development of beacon and robot recognition.
Rotation Algorithm	
Filtering Algorithm	Explored during robot recognition.
Beacon Recognition	First assigned objective.
Robot Recognition	Second assigned objective.
Prediction Algorithm	Explored during the development of the WMD.
WMD aka 'World Model Debugger'	Explored during robot recognition.

The methods derived in Horizon Line, Rotation Algorithm, Filtering Algorithm, Beacon Recognition and Robot Recognition are verified using NUbots' own visual debugger application, called EOTN, which stands for '**E**ye **o**f **t**he **N**Ubot'. This application allows the end user to debug their logic by visually presenting screens that process frames taken from the robot's camera, and performs operations such as drawing the horizon line, allowing users to create a look up table for colour classification, draw blobs based on the classified image, and aid in debugging code for distinguishing which blobs are the ball, the goals, the beacons and the robots.

The effectiveness of the proposed prediction algorithm is verified through the WMD debugger application.

At present, the majority of low level code is written in C++ using Microsoft Visual C++ Version 6. Codes that require constant tuning of parameters are written in Python. Python is an interpreted, interactive and object-oriented language that doesn't require compiling like C++. Using python reduced development time, as python code can be transmitted to a robot using wireless connection and debugged, rather than turning off a robot, take out the memory stick, and recompile the code on the desktop before inserting the memory stick back into a robot to observe the change.

4. Horizon Line

Horizon line is the line created at infinity where the soccer field seems to disappear from the camera's perspective based on the height of the camera. The horizon line aids in vision code as it is an indication that the head of the robot has been tilted, resulting in images that are not in parallel to the ground. Calculation of the Horizon Line has been previously tackled by the German Team. However, we observed inaccuracies in our implementation of their calculations on the ERS-7 robots and therefore we had to re-derive the calculations involved to resolve these problems.

4.1. The P-Plane and the H-Plane

The horizon line is calculated by finding the intersection of the left and right edges in the Picture Plane (P-Plane) with the Horizon Plane (H-Plane). The P-Plane is the plane of the image taken by the camera limited in length and width by the camera's horizontal and vertical field of view. The H-Plane is a levelled plane parallel to the ground with height given by the height of the camera. This is illustrated in Figure 4-1.

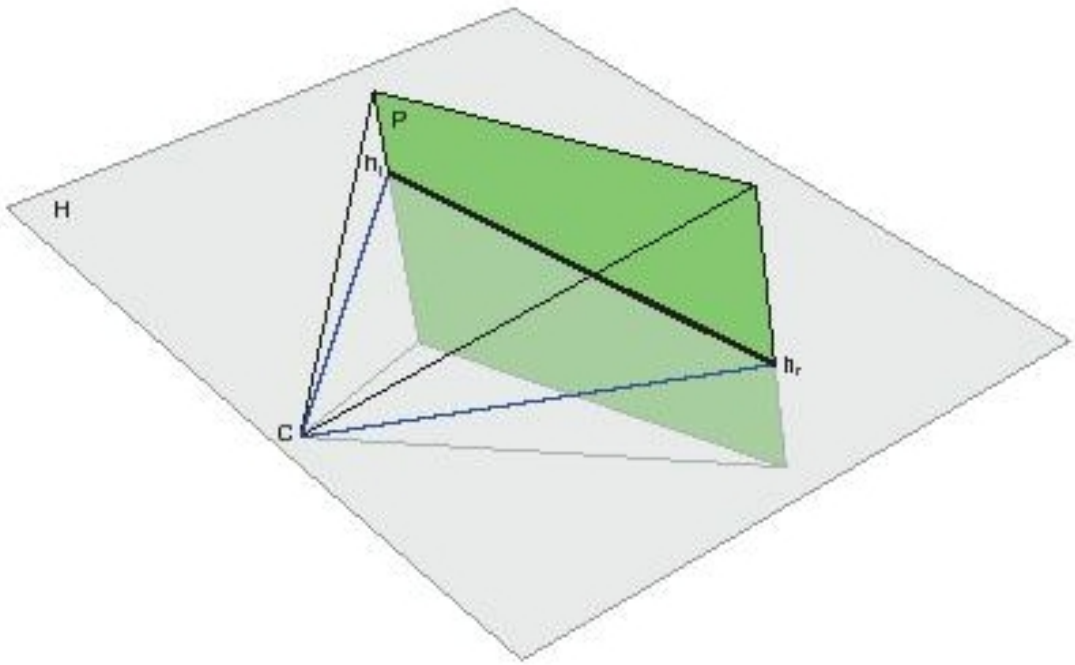


Figure 4-1: Picture Plane intersecting with Horizon Plane. Horizon Line is found by finding left and right intersection point, hL and hR respectively. This image is taken from [7].

4.2. Derivation of the P-Plane

Prior to deriving the P-Plane, we will introduce two 3 dimensional co-ordinate systems. The first co-ordinate system should have dimensions labelled F,S,H for front, side and height axis respectively. The second co-ordinate system should have dimensions labelled cF, cS, cH for camera front, camera side, and camera height axis respectively. This prepares us for matrix transformation [8].

The first co-ordinate system should be called the reference frame, and this represents the actual world where (0,0,0) represents the neck base of the robot. The second co-ordinate system should be called the camera frame where (0,0,0) represents the location of the camera on the robot [8].

The purpose of matrix transformation is to map the location of the camera relative to the neck base of the robot, by projecting the camera frame onto the reference frame.
The initial step is to find the four corner points of the P-Plane on the camera frame. This is achieved by using the horizontal and vertical field of view angle of the camera, FOVx and FOVy respectively, which is a given constant. The calculation is as follows:

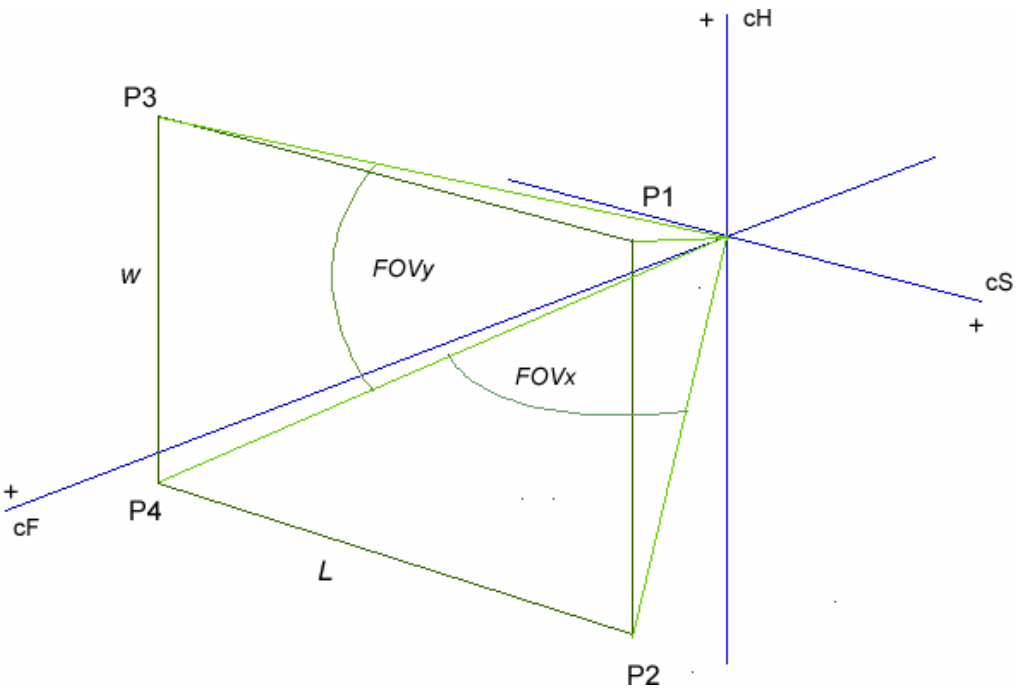
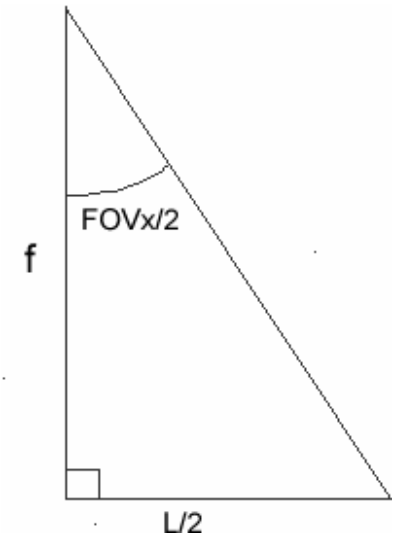


Figure 4-2: Four Corner Points within Camera Frame

We define f as the coordinate of the corner points on the cF axis, and the focal length. f can be found using the following trigonometry:



tan(FOVx/2) = (L/2) /f

f = L/2 **Í** cot(FOVx/2)

[This can be calculated using FOVy:
ie f = w/2 **Í** cot(FOVy/2)]

cS and cH coordinates can be easily obtained as follows:
cS = L/2 for P1, P2
cS = -L/2 for P3, P4
cH = w/2 for P1, P3
cH = -w/2 for P2, P4

Figure 4-3: Calculating f on cF axis

Using the calculated coordinates, the corner points are as follows: (cF,cS,cH)

$$P_1 = \left(\frac{L}{2} \times \cot\left(\frac{FOVx}{2} \right), \frac{L}{2}, \frac{w}{2} \right) \qquad P_2 = \left(\frac{L}{2} \times \cot\left(\frac{FOVx}{2} \right), \frac{L}{2}, -\frac{w}{2} \right)$$

$$P_3 = \left(\frac{L}{2} \times \cot\left(\frac{FOVx}{2} \right), -\frac{L}{2}, \frac{w}{2} \right) \qquad P_4 = \left(\frac{L}{2} \times \cot\left(\frac{FOVx}{2} \right), -\frac{L}{2}, -\frac{w}{2} \right)$$

Now suppose that we begin with the camera frame in the same stance as the reference frame. That is, the origin (0,0,0) of the camera frame is located at the origin (0,0,0) of the reference frame where the front, side and height axis are matched.

To simplify matrix transformation, introduction of snout distance, snout angle offset, and camera angle offset is necessary. Snout distance is the length between the camera and the pivot point of the head, which is calculated by trigonometry. Snout angle offset is the angle from the mouth axis to the snout. Camera angle offset is the angle between the perpendicular to the snout and the P-Plane, which is the surface of the camera. This is illustrated in Figure 4-4.

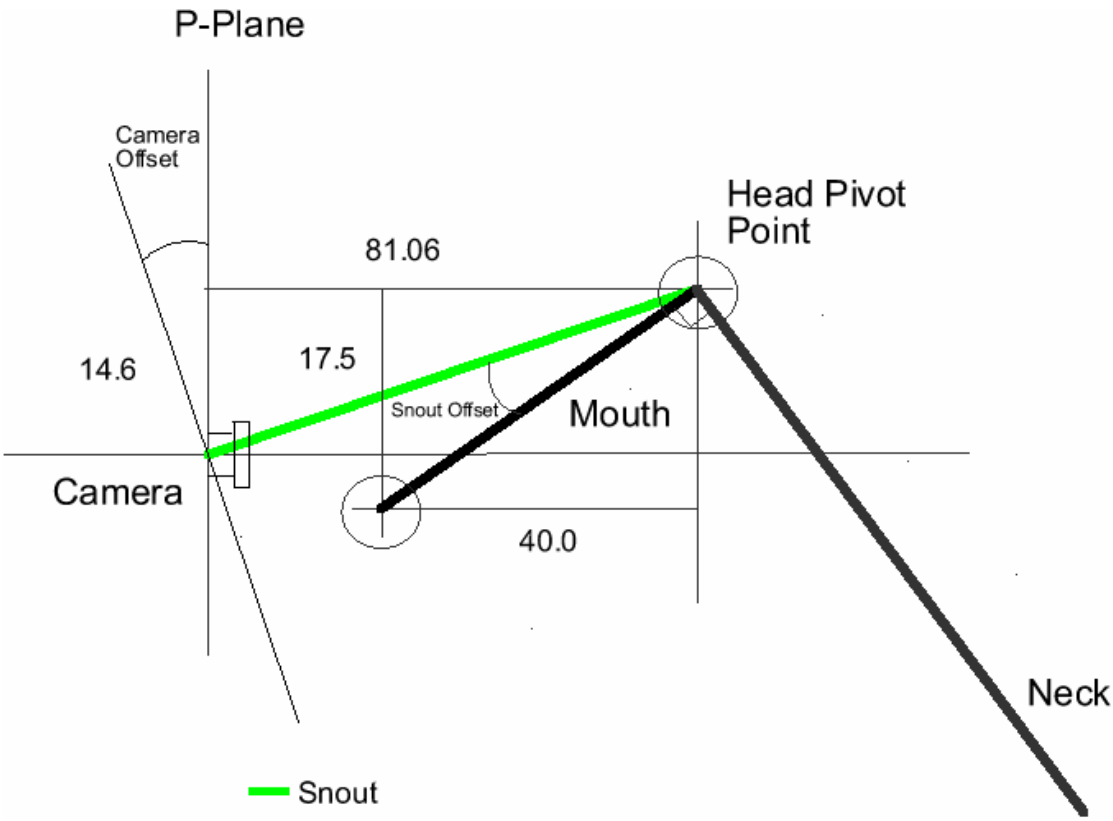


Figure 4-4: Introduction of Snout, Camera angle offset, and Snout angle offset to simplify matrix transformations.

By Introducing Snout Angle Offset, θ_S,Small tilt, θ_{ST} , must include Snout Angle Offset:
$\theta_{ST} = \theta_{ST} + \theta_S$

Now we use matrix transformation [8].
Steps

i) Rotation about F axis if there is body roll, θ_{BR}.
ii) Rotation about cS axis if there is body tilt, θ_{BT}.
iii) Translation about the neck length towards the head on cH axis, N_L.
iv) Rotation about cH axis if there is head pan, θ_P.
v) Rotation about cS axis if there is small tilt, θ_{ST}.

vi) Translation about the snout towards the camera on cF axis, S_L.

vii) Rotation about cS axis for camera offset, θ_C.

The arrived equation for the head frame becomes:

Head Frame = Rot(F,θ_{BR}) o Rot(cS, θ_{BT}) o Trans(0,0, N_L) o Rot(cH, θ_P) o
Rot(cS, θ_{ST}) o Trans(S_L,0,0) o Rot(cS, θ_C)

The mapping of the 4 corners points of the camera frame to the reference frame becomes:

Reference Frame = Head Frame **í** Points

Let the transformed points be PTi where i is from 1 to 4. Then

PTi = Head Frame **í** Pi (i is from 1 to 4)

Below are the solutions for PT1, PT2, PT3, and PT4.

$$P_{T1} = \begin{bmatrix} \frac{L}{2}\cot\left(\frac{FOVx}{2}\right)a_1 + \frac{L}{2}a_2 + \frac{w}{2}a_3 + a_4 \\ \frac{L}{2}\cot\left(\frac{FOVx}{2}\right)b_1 + \frac{L}{2}b_2 + \frac{w}{2}b_3 + b_4 \\ \frac{L}{2}\cot\left(\frac{FOVx}{2}\right)g_1 + \frac{L}{2}g_2 + \frac{w}{2}g_3 + g_4 \end{bmatrix} \qquad P_{T2} = \begin{bmatrix} \frac{L}{2}\cot\left(\frac{FOVx}{2}\right)a_1 + \frac{L}{2}a_2 - \frac{w}{2}a_3 + a_4 \\ \frac{L}{2}\cot\left(\frac{FOVx}{2}\right)b_1 + \frac{L}{2}b_2 - \frac{w}{2}b_3 + b_4 \\ \frac{L}{2}\cot\left(\frac{FOVx}{2}\right)g_1 + \frac{L}{2}g_2 - \frac{w}{2}g_3 + g_4 \end{bmatrix}$$

$$P_{T3} = \begin{bmatrix} \frac{L}{2}\cot\left(\frac{FOVx}{2}\right)a_1 - \frac{L}{2}a_2 + \frac{w}{2}a_3 + a_4 \\ \frac{L}{2}\cot\left(\frac{FOVx}{2}\right)b_1 - \frac{L}{2}b_2 + \frac{w}{2}b_3 + b_4 \\ \frac{L}{2}\cot\left(\frac{FOVx}{2}\right)g_1 - \frac{L}{2}g_2 + \frac{w}{2}g_3 + g_4 \end{bmatrix} \qquad P_{T4} = \begin{bmatrix} \frac{L}{2}\cot\left(\frac{FOVx}{2}\right)a_1 - \frac{L}{2}a_2 - \frac{w}{2}a_3 + a_4 \\ \frac{L}{2}\cot\left(\frac{FOVx}{2}\right)b_1 - \frac{L}{2}b_2 - \frac{w}{2}b_3 + b_4 \\ \frac{L}{2}\cot\left(\frac{FOVx}{2}\right)g_1 - \frac{L}{2}g_2 - \frac{w}{2}g_3 + g_4 \end{bmatrix}$$

The alphas, betas, and deltas are defined below.

To simplify expression let c = cos, s = sin:

$$a_1 = [cq_{BT}cq_P c(q_{ST}+q_C) - sq_{BT}s(q_{ST}+q_C)]$$
$$a_2 = [-cq_{BT}sq_P]$$
$$a_3 = [-cq_{BT}cq_P s(q_{ST}+q_C) - sq_{BT}c(q_{ST}+q_C)]$$
$$a_4 = [cq_{BT}cq_P cq_{ST}S_L - sq_{BT}(sq_{ST}S_L + N_L)]$$
$$b_1 = [cq_{BR}sq_P c(q_{ST}+q_C) + sq_{BR}(sq_{BT}cq_P c(q_{ST}+q_C) + cq_{BT}s(q_{ST}+q_C))]$$
$$b_2 = [cq_{BR}cq_P - sq_{BR}sq_{BT}sq_P]$$
$$b_3 = [-cq_{BR}sq_P s(q_{ST}+q_C) + sq_{BR}(-sq_{BT}cq_P s(q_{ST}+q_C) + cq_{BT}c(q_{ST}+q_C))]$$
$$b_4 = [cq_{BR}sq_P cq_{ST}S_L + sq_{BR}(sq_{BT}cq_P cq_{ST}S_L + cq_{BT}(sq_{ST}S_L + N_L))]$$
$$g_1 = [-sq_{BR}sq_P c(q_{ST}+q_C) + cq_{BR}(sq_{BT}cq_P c(q_{ST}+q_C) + cq_{BT}s(q_{ST}+q_C))]$$
$$g_2 = [-sq_{BR}cq_P - cq_{BR}sq_{BT}sq_P]$$
$$g_3 = [sq_{BR}sq_P s(q_{ST}+q_C) + cq_{BR}(-sq_{BT}cq_P s(q_{ST}+q_C) + cq_{BT}c(q_{ST}+q_C))]$$
$$g_4 = [-sq_{BR}sq_P cq_{ST}S_L + cq_{BR}(sq_{BT}cq_P cq_{ST}S_L + cq_{BT}(sq_{ST}S_L + N_L))]$$

The intersection of the P-Plane to the H-Plane can be obtained by first finding the line equation of the left and right edges of P-Plane which are $P_{T1}P_{T2}$ and $P_{T3}P_{T4}$ respectively:

Using the symmetry equation for deriving the line equation [9]:

$$\frac{F - P_{Ti[F]}}{P_{T(i+1)[F]} - P_{Ti[F]}} = \frac{S - S_{Ti[S]}}{S_{T(i+1)[F]} - S_{Ti[F]}} = \frac{H - H_{Ti[H]}}{H_{T(i+1)[F]} - H_{Ti[F]}}$$

The left edge line equation becomes:

$$\frac{F - P_{T1[F]}}{P_{T2[F]} - P_{T1[F]}} = \frac{S - S_{T1[S]}}{S_{T2[F]} - S_{T1[F]}} = \frac{H - H_{T1[H]}}{H_{T2[F]} - H_{T1[F]}}$$

The right edge line equation becomes:

$$\frac{F - P_{T3[F]}}{P_{T4[F]} - P_{T3[F]}} = \frac{S - S_{T3[S]}}{S_{T4[F]} - S_{T3[F]}} = \frac{H - H_{T3[H]}}{H_{T4[F]} - H_{T3[F]}}$$

Then we are required to find the H-Plane equation as this is needed for substituting in the variable H to find the points of intersection.

4.3. Derivation of the H-Plane

Similar methods are required for finding the H-Plane. Instead of using the camera frame, we introduce the horizon frame to work with the reference frame. The horizon frame is the camera frame without the four corner points.

Now we use matrix transformation [8].

Steps

i) Rotation about F axis if there is body roll, θ_{BR}.

ii) Rotation about cS axis if there is body tilt, θ_{BT}.

iii) Translation about the neck length towards the head on cH axis, N_L.

iv) Rotation about cH axis if there is head pan, θ_P.

v) Rotation about cS axis if there is small tilt, θ_{ST}.

vi) Translation about the snout towards the camera on cF axis, S_L.

The arrived equation for the horizon frame becomes:

Horizon Frame = Rot(F,θ_{BR}) o Rot(cS, θ_{BT}) o Trans(0,0, N_L) o Rot(cH, θ_P) o
Rot(cS, θ_{ST}) o Trans(S_L,0,0)

By observation this is similar to the Head Frame equation, but without the camera angle offset rotation.

The mapping of the horizon plane to the reference frame becomes:

Reference frame = Horizon Frame **í** Origin

Therefore the H-Plane with respect to the reference frame becomes:
To simplify expression let c = cos, s = sin:

$$H_{Plane} = -sq_{BR}sq_{P}cq_{ST}S_{L} + cq_{BR}\left(sq_{BT}cq_{P}cq_{ST}S_{L} + cq_{BT}\left(sq_{ST}S_{L} + N_{L}\right)\right)$$

By letting the variable H be the horizon plane in the symmetry equation, we are reducing 3 dimensions to 2 dimensions. This allows us to calculate the co-ordinates f_L,s_L and f_R,s_R, which represents the co-ordinates of the left and right edges of the P-Plane intersecting with the H-Plane.

Our left intersection points are:

$$F_{L} = F = P_{T1[F]} + \left(P_{T2[F]} - P_{T1[F]}\right)\left(\frac{H_{Plane} - H_{T1[H]}}{H_{T2[F]} - H_{T1[F]}}\right)$$

$$S_{L} = S = S_{T1[S]} + \left(S_{T2[F]} - S_{T1[F]}\right)\left(\frac{H - H_{T1[H]}}{H_{T2[F]} - H_{T1[F]}}\right)$$

Our right intersection points are:

$$F_{R} = F = P_{T3[F]} + \left(P_{T4[F]} - P_{T3[F]}\right)\left(\frac{H - H_{T3[H]}}{H_{T4[F]} - H_{T3[F]}}\right)$$

$$S_{R} = S = S_{T3[S]} + \left(S_{T4[F]} - S_{T3[F]}\right)\left(\frac{H - H_{T3[H]}}{H_{T4[F]} - H_{T3[F]}}\right)$$

The EOTN draws the horizon line by using h_L and h_R, which are scalar values that start from P1 to P2 for the left edge and P3 to P4 for the right edge:

$$h_L = \sqrt{\left(F_L - P_{T1[F]}\right)^2 + \left(S_L - P_{T1[S]}\right)^2 + \left(H_{Plane} - P_{T1[H]}\right)^2}$$

$$h_R = \sqrt{\left(F_R - P_{T3[F]}\right)^2 + \left(S_R - P_{T3[S]}\right)^2 + \left(H_{Plane} - P_{T3[H]}\right)^2}$$

The trouble with scalar values of h_L and h_R is that the intersection may be above P1 (for h_L) and P3 (for h_R) which are meant to be negative values; thus the horizon line is drawn inaccurately, even though the scalar values are correct.

To correct this, differences between the points F_L, S_L to P_{T1} and F_R, S_R to P_{T3} found that:

If F_L-$P_{T1[F]} > 0$ and $S_L - P_{T1[S]} > 0$, then negate h_L
If F_R-$P_{T3[F]} > 0$ and $S_R - P_{T3[S]} > 0$, then negate h_R

4.4. Performance

The equations have been implemented in python and the drawing of the horizon line showed good results.

5. Rotation Algorithm

The rotation algorithm rotates blobs so that they would appear as though the camera of an ERS-7 robot was flat, providing an ideal situation for object recognition. This has not been tackled in previous years, which means when the head was tilted objects like beacons were difficult to recognised or not recognised at all.

The approach was simple. Rotate the blobs about the centre of the image with an angle that is negative of that to the horizon line gradient.

5.1. Rotation Maths

The rotation calculations are as follows:

horizonLine.Angle = tan-1(horizonLine.gradient)
offsetX = IMAGE_HEIGHT/2
offsetY = horizonLine.gradient Í offsetX + horizonLine.offset
X = X – offsetX
Y = Y – offsetY

rotatedX = X Í Cos(-horizonLine.Angle) – Y Í Sin(-horizonLine.Angle)
rotatedY = X Í Sin (-horizonLine.Angle) + X Í Cos(-horizonLine.Angle)

rotatedX = rotatedX + offsetX
rotatedY = rotatedY + offsetY

The offsets are required to move the blob into the centre of rotation (which is the centre X of the image and height is where the horizon line intersects the centre X), and then they are moved back following the rotation.

By not rotating every pixel and then reforming the blob we are saving ourselves a lot in terms of computation. The difficult part was realising not to simply take the corner points in the calculation, as straight forward rotation would cut out useful information, or introduce garbage inside the rotated blob. To understand this, Figure 5-1 shows the information stored about a blob when it is formed from the classified image.

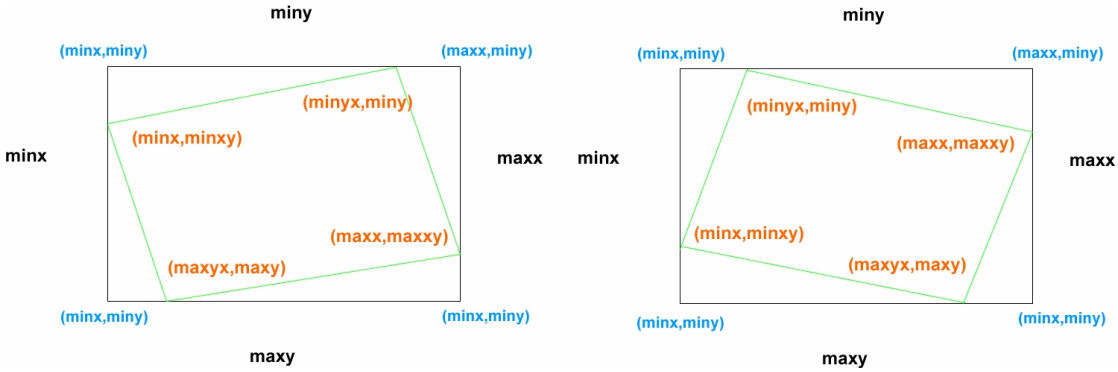


Figure 5-1: Orange text are the real co-ordinates, Blue text are the blob corners.

From the above illustration we would rotate the real coordinates of the blob (orange text) if the real coordinates are not the corners. By doing this the rotated blob would have maximum coverage of the real colours as if the classified pixels were rotated, and then subjected to blob formation. In cases where the angle of rotation is large, the code ensured that the rotated coordinates derived correctly the variables of minx, maxx, minyx, maxyx, minyx, maxy, minxy and maxxy. As transform position relied on the inverse-rotated coordinates, a function was created to allow the raw coordinates to be calculated.

5.2. Inverse-Rotation Maths

The inverse-rotation calculations are as follows:

horizonLine.Angle = tan-1(horizonLine.gradient)
offsetX = IMAGE_HEIGHT/2
offsetY = horizonLine.gradient Í offsetX + horizonLine.offset
rotatedX = rotatedX – offsetX
rotatedY = rotatedY – offsetY

X = X Í Cos(horizonLine.Angle) – Y Í Sin(horizonLine.Angle)
Y = X Í Sin (horizonLine.Angle) + X Í Cos(horizonLine.Angle)

X = X + offsetX
Y = Y + offsetY

5.3. Benefits

Figure 5-2 illustrates the benefits of the rotation algorithm.

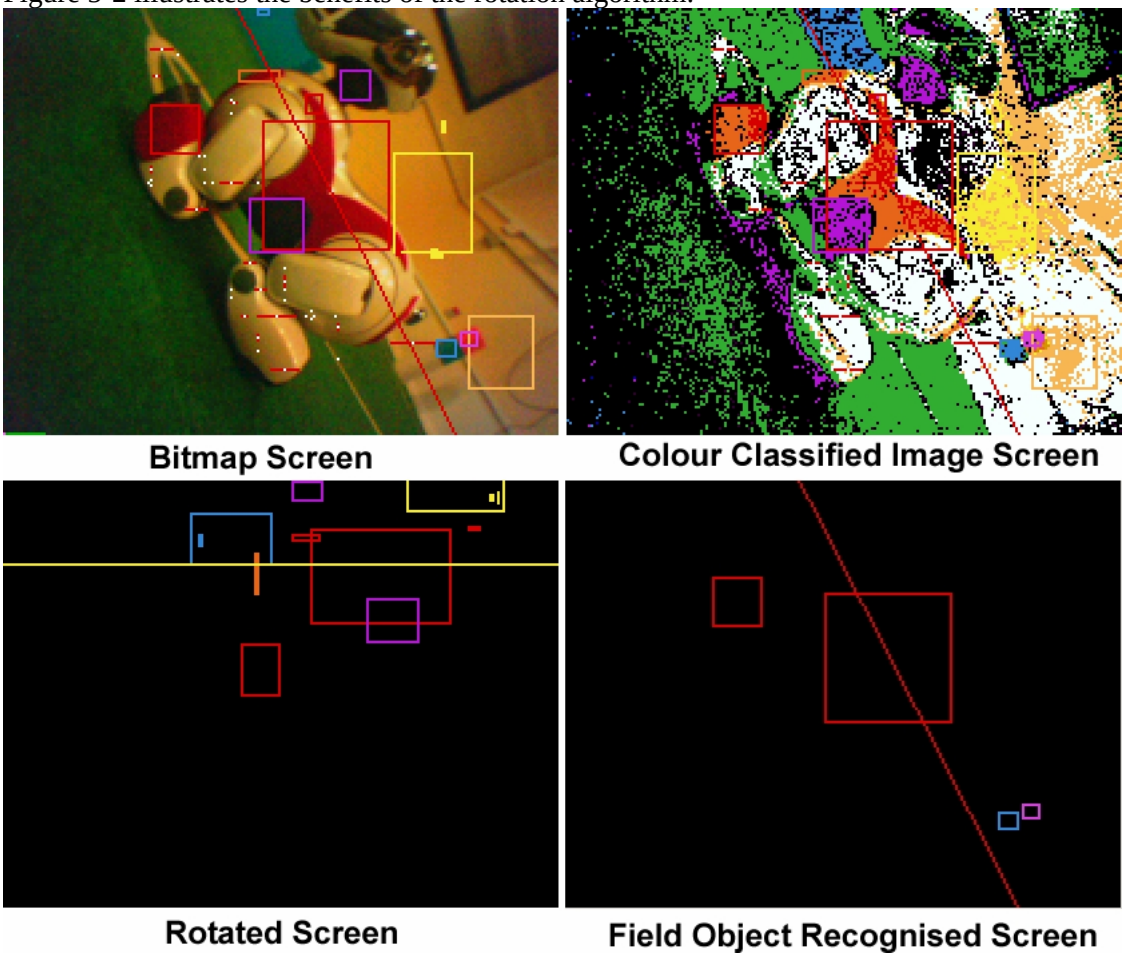


Figure 5-2: If rotation was not used, the robot recognition would result with a right orientation, not left. The case for beacon blobs being rotated out of screen was not a major issue, as the information of the rotated beacon blobs were retained to be recognised by the beacon recognition code.

Figure 5-2 is an example of how we have saved ourselves countless lines of coding and debugging as there is no need to cater recognition for every combination of blob patterns whenever a robot's head has been tilted. The rotated blobs are drawn in EOTN on a separate screen with the horizon line as a flat yellow line.

5.4. Performance

The performance of the rotation algorithm is very good. At one time, it was drawing rotated blobs incorrectly when the rotated coordinates went above or beyond the left of the screen. This was due to arriving at negative coordinates, and was resolved by casting the stored variables as integers rather than unsigned shorts. At present, the rotation algorithm has provided ideal scenarios for beacon recognition and robot recognition without slowing down frame rates or reducing execution speed.

6. Filtering Algorithm

The filtering algorithm aims to remove noisy blobs from vision. Previous years had some form of filtering, however not as extensive and filtering was performed within individual object recognition, which meant redundant code. This filtering algorithm was written in robot recognition to reduce strain on assigning robot blobs to robot containers. Eventually it was separated so that it could be used for filtering beacon blobs.

The filtering algorithm is called after the rotation algorithm. It examines each blob and decides whether it should be ignored or merged with another blob by using threshold values.

6.1. Supported Scenarios

Figure 6-1 illustrates the scenarios catered for in the filtering algorithm.

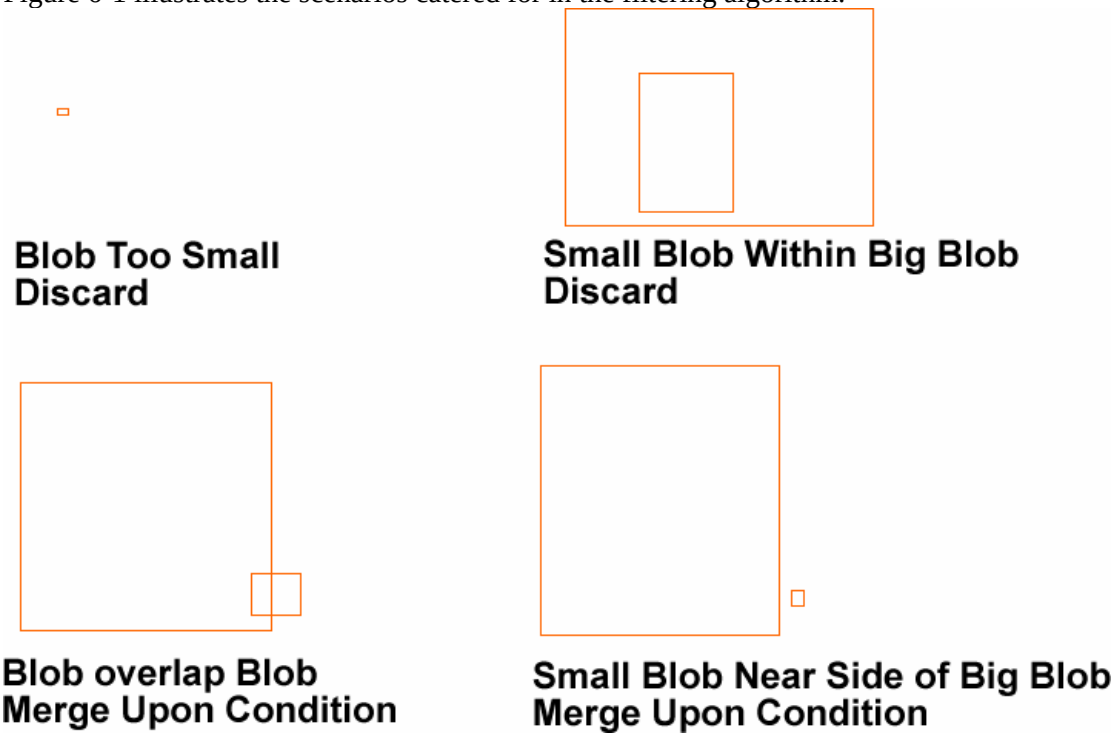


Figure 6-1: Outlines the purpose of the filtering algorithm.

Merge upon condition require rules such as one blob should be sufficiently larger than the other for them to become one; otherwise you would end up with a very large blob where object recognition has no useful information to process.

There were discussions among the group of placing the rotation algorithm within the filtering algorithm. This did not go forward by considering that the current rotation works on-the-fly which is O(1) operations, while the filtering algorithm is $O(n^2)$ operations, where n is the number of blobs.

6.2. Performance

Similar to the rotation algorithm, the filtering algorithm has performed as required, and the results of the filtering algorithm can be seen in EOTN between the classified blobs screen and the rotated blobs screen. This is because filtering is performed before the rotated blobs are drawn.

7. Beacon Recognition

Beacons are the posts around the edges of the soccer field which allows the robot to determine its position. There are four beacons in total, each are distinguished by their combinations of pink with yellow, and pink with blue, as illustrated in Figure 7-1.

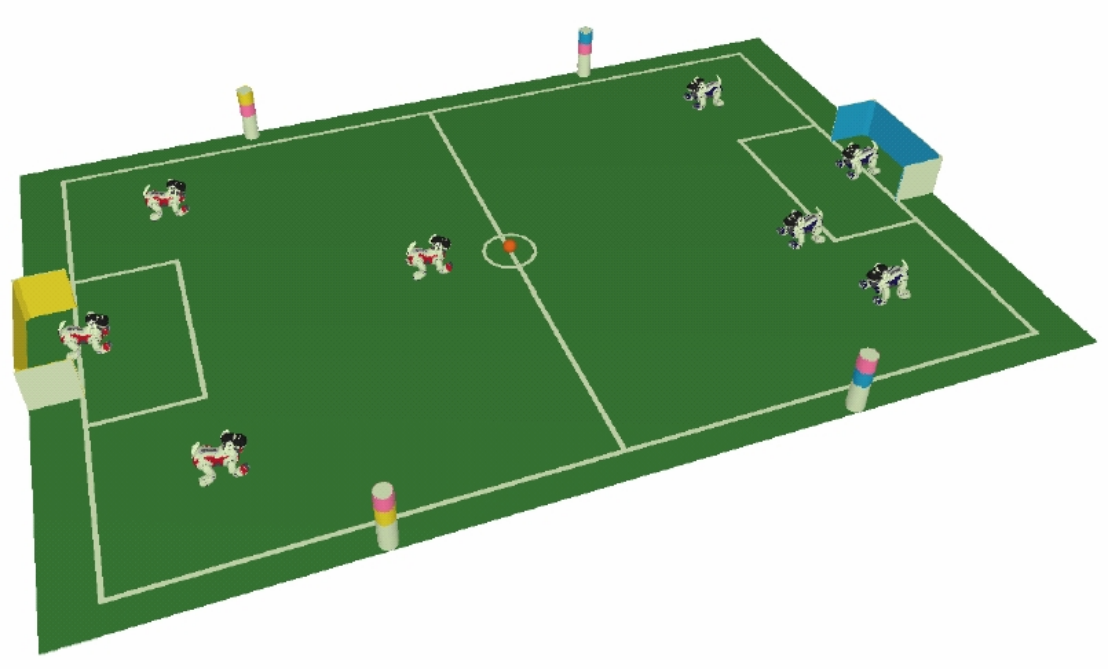


Figure 7-1: Location of beacons, and their colour combinations. This image is taken from [10]

Beacon recognition has been tackled in previous years of Robocup, however it was encouraged by the team that a rewrite was necessary. There have been quite a number of revisions to date with the new code.

7.1. Initial Code

The initial attempt to recognise beacons was written within python. The code looked for individual beacons by going through all pink and yellow blobs for pinkYellow (Pink = Top, Yellow = Bottom) and yellowPink (Yellow = Top, Pink = Bottom) beacons, then all the pink and blue blobs for pinkBlue and bluePink beacons. This essentially took $O(4n^2)$ operations where n is the number of blobs. The additional computations resulted in reduction to the frame rate to around 18 fps in cases when all beacons were visible. 18 fps is considered poor, as the recommended was 30 fps, otherwise the robot would exhibit a slow response.

7.2. Evolution I

The next evolution of beacon recognition began finding beacons by iterating through pink blobs. For each pink blob, we iterate through each blue and yellow blobs. This reduced the computation from $O(4n^2)$ to $O(2n^2)$ where n is the number of blobs. At the time, EOTN became capable of speed testing written codes. We noticed a speed improvement when recognition of goals was ported within beacon recognition. This means that whenever a yellow or blue blob is not regarded as a possible beacon blob, then it is a candidate to be a goal. A scan for white beneath the pink blobs allows the code to determine whether the pink blob is a top or bottom beacon blob. The choice of white was because colour classification of white was simple, and the colour beneath the bottom beacon blob was white. An additional speed improvement was made when it was decided that all beacon blobs be sorted from largest to smallest.

Even though the changes made in python allowed 30fps to be reached, however it was felt that by porting the merged beacons and goal code to C++, we have allowed further exploration of python code in ball and robot recognition without impact on the achievable frame rate. For awhile the beacons and goals remained merged, and then a separation was necessary to retain logical design structure and allow group members to work on their individual parts.

7.3. Evolution II

The next evolution of beacon recognition code was made after the 2005 Australian Robocup Open at Griffith University, Brisbane. It was noticed on the Australian Open that the previous evolution of beacon recognition missed beacons. This was due to the scanning for white on the classified image beneath a pink beacon blob. This was because the green soccer field at Griffith University is much lighter in colour, and made the classification of green on areas where it was meant to be white. Since the check for white determines that the pink blob is on the bottom, beacons like yellowPink and bluePink were difficult to recognise.

Before reaching the current evolution, a test bench was made in scanning for yellow and blue, above and below a pink beacon blob from the classified image. Out of 10 beacon frames, in good cases 6 were recognised, in average cases 4 was recognised. As you can see scanning on classified colours does not produce perfect results, even though beacon blobs was clearly present. This is due to colour classification either misclassifying the colours, or there is missing information, due to under classification.

7.4. Evolution III: Current

The current evolution takes concepts found in the initial code with those of previous evolutions. The initial code didn't require scanning of classified image, which is incorporated in the current evolution. Previous evolutions had $O(2n^2)$ operations where n is the number of blobs; this is what the current evolution achieves in worst case scenario.

The current evolution iterates through pink blobs, taking the largest first. It loops through the sorted yellow blobs then sorted blue blobs, which are sorted by size of largest to smallest, then passes them to the recognition code. The recognition code determines which blobs are top and which are bottom beacon blobs, and this is performed quickly. As two blobs are recognised as beacons, it would be ignored. When both yellowPink and pinkYellow beacons are recognised, the code would no longer deal with the remaining yellow blobs, same goes when bluePink and pinkBlue beacons have been recognised.

7.5. Forming Beacon Field Objects

Once two beacon blobs satisfy the checks to be regarded as a beacon, the two blobs are used to create the beacon field object. Parameters such as centreX, centreY of the beacon field object are determined by using the information from both blobs; for instance, the centreX, centreY of the beacon field object is calculated using the two blobs.

The beacon field object is then passed into transform position to determine its elevation, bearing, and distance. Transform position consists of matrix calculations that involves translations and rotations [8], and it requires tuning a calibration constant. The calibration constant is a mathematically derived ratio in transform position that requires tuning due to inconsistencies in the camera distance, colour classification errors, and slack in the sensor inputs. For beacons, the calibration constant is a number divided by the distance between centres of the beacon blobs.

Since elevation relies on centreY of the beacon field object, bearing relies on centreX of the beacon field object, and distance relies on distance between the two blob centres, their calculations are consistent if ideal beacon blobs are used.

7.6. Non-Ideal Beacon Blobs

Non-ideal beacon blobs can offset distance, bearing and elevation, which affects localisation and hinges on behaviour as misleading information is fed into the ERS-7 robot's camera. This is illustrated in Figure 7-2.

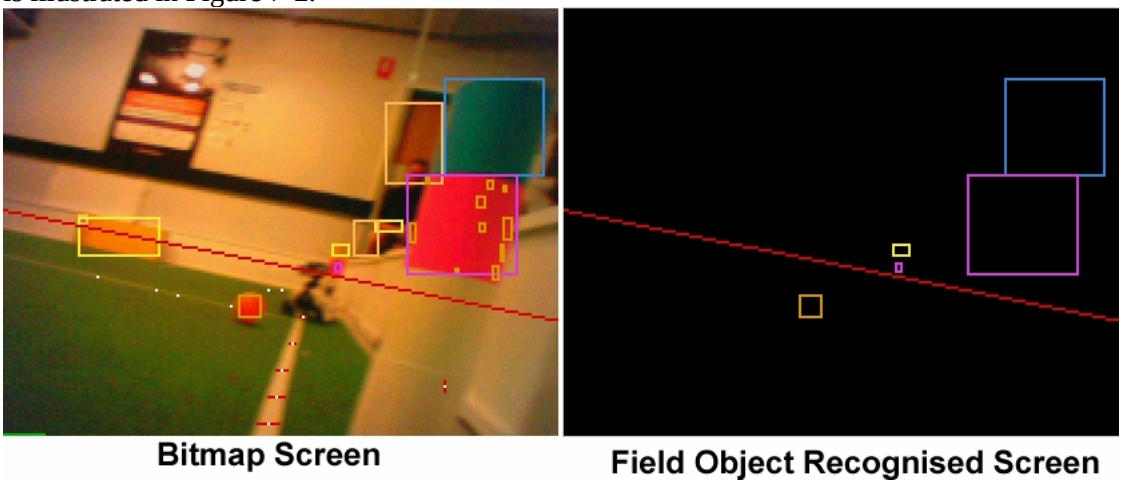


Figure 7-2: The further away the beacon blobs are, the more non-ideal it becomes. The images here are continued in the next figure.

Non-ideal beacon blobs stems from colour classification, and is a hard case to tune, as mapping more raw colours may allow more useless blobs to be formed elsewhere in the image. The solution was to make the beacon recognition code clever in taking bad beacon blobs and turning them into ideal beacon blobs. Take the instance of a small beacon blob matched with a big beacon blob, formed by colour classification. If these two blobs pass sanity checks to become a beacon field object, then project the small blob to the size of the big blob. This will improve distance, bearing and elevation calculation and provide consistency in the recognition. This is illustrated in Figure 7-3.

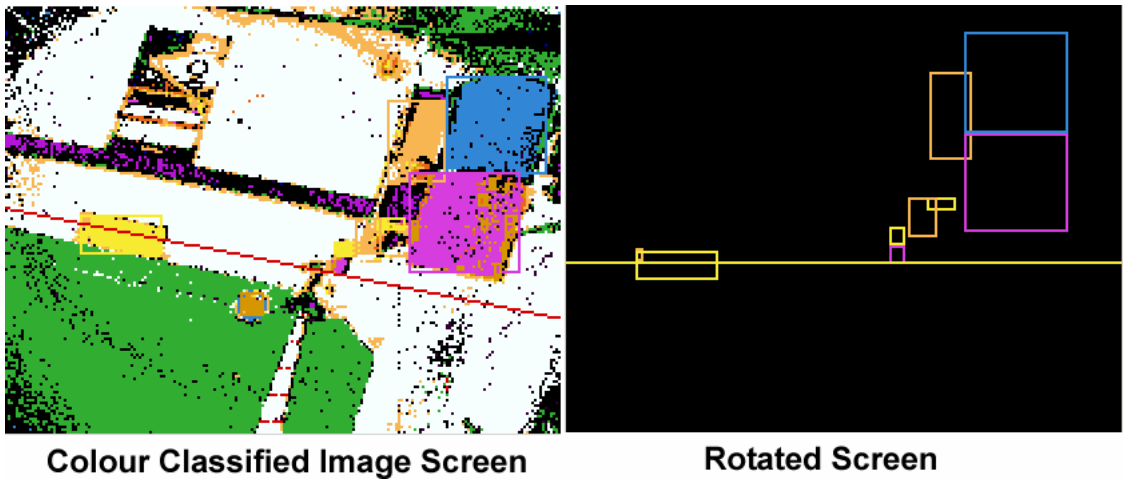


Figure 7-3: Small beacon blobs projected to be big beacon blobs. This creates an ideal scenario, and allows distance, bearing, elevations to be consistent and accurate. The projection is performed after rotation. The screen on the right is the rotated screen in EOTN. The bitmap image is in the previous figure. These images are taken with the colour classification look up table on the 22-05-2005.

7.7. Colour Classification Affects Distance

Besides non-ideal beacons affecting distance calculations, beacons that are far away provide poor distances. Close up beacons have consistent distances returned by transform position since the beacon blobs are large enough to calculate distance between the centres. Beacon blobs that are far away are prone to colour classification problems, resulting in transform position returning values that are noisy, and inconsistent. An improvement was by taking the distance from the top of the bottom beacon blob down to where the white of the beacon post meets the green field for far away beacon blobs. This makes the distance values larger for transform position to return consistent distances as good as close up beacons. This meant that there are two calibration constants for distance calculation of beacons; one for close up beacon field objects, and the other is for far away beacon field objects. This addition existed before the 2005 Australian Robocup Open at Griffith University, Brisbane.

7.8. False Beacons

Another example of colour classification problem in beacon recognition is that beacon field objects were being formed inside the ball. This wasn't an error with beacon recognition, as there was beacon colour blobs found in the classified image. This was a recent issue, and it was resolved by using elevation checks that rejects beacon field objects if they have an elevation level lower than -6 degrees. The elevation is the angle from the neck base of the ERS-7 robot towards a field object. A positive elevation indicates that the field object is above the neck base of an ERS-7 robot, while a negative elevation indicates that the field object is below the neck base of an ERS-7 robot.

7.9. Performance

The current evolution of beacon field object formation has picked up possible beacons from the blobs formed by colour classification in every frame, which is very good. In cases when the robot's head is tilted, the beacons can still be detected. This is achieved by passing the beacon blobs through the rotation algorithm, as discussed in section 4.

Furthermore, noisy beacon blobs are filtered using the filtering algorithm, as discussed in section 5.

8. Robot Recognition

Robots are the four-legged ERS-7s on the soccer field. There are a total of eight robots on the soccer field, four belong to our team, and the other four are our opponent's team. Each team is distinguished on the field by the uniform colours on the robots, which are either red or dark blue. Within a 20 minute match, the teams are required to swap uniform colours at half time.

Previous years attempt on robot recognition has not provided good results, and it was not used in the Robocup competition 2004. Likewise with beacon recognition, robot recognition was encouraged by the team to be rewritten.

This year, we have added orientation recognition as a new feature in the robot recognition code. Orientation recognition relies on having sufficient information of the blobs from the classified image to estimate the directional stance of the robots on the field, in particular the opponent. By knowing our opponents' directional stance, our team can implement better game play behaviours. For instance, our team would kick the ball past an opponent robot, whose current directional stance is facing away from the ball, rather than towards it.

8.1. Robot Blob Clusters

The general method of robot recognition begins by finding clusters of robot blobs on the field. Each identified cluster may represent an ERS-7 robot, and in the code the robot blobs inside the cluster are assigned to a robot container. The robot container is a data structure that stores at most three robot blobs, which is sufficient for the code to estimate a robot's orientation. The sufficient condition was derived by observing the possible number of patterns formed by the uniforms as an ERS-7 robot is viewed from different perspectives.

8.2. Assigning Robot Blobs to Robot Containers

The initial attempt of assigning robot blobs to robot containers worked on an arbitrary sorted order of robot blobs. This meant that the first robot blob may be located on the top left corner; the second robot blob may be located on the bottom right hand corner; the third robot blob may be located on the bottom left corner; and so on. The reason for the arbitrary sorted order came from the blob formation code that forms and merges blobs by scanning top-down, left to right [6].

An approach to assigning robot blobs to robot containers was using sanity factors, which switches on when there is at least one robot blob in a robot container. Sanity factors are values that are similar in respect to a person's confidence. The confidence increases when goals are achieved, and decrease when they are not. The goals are conveyed through threshold values in the robot recognition code. The sanity factors work by examining the current robot blob, then checking relationships, (such as ratios between the current robot blob area and distance to all the other robot blobs in all the robot containers) then assign the current robot blob to the appropriate robot container based on sanity factor values. This was tedious and time consuming as thresholds that work with sanity factors required constant tuning and testing.

An improvement in assigning robot blobs to robot containers was by sorting the robot blobs from left to right. This lessens the tuning of threshold values as results were favourable. The results were favourable because robot containers are created sequentially. That is, the current robot blob would be compared with only the current robot container, rather than with all the robot containers; and if the current robot blob appears to be another ERS-7 robot, then a new robot container is created. Further improvements began when scanning of classified image was used. The robot code scanned three areas between each robot blob for colours of green or orange; if there are colours of green or orange, a new robot container is formed. This is illustrated in Figure 8-1.

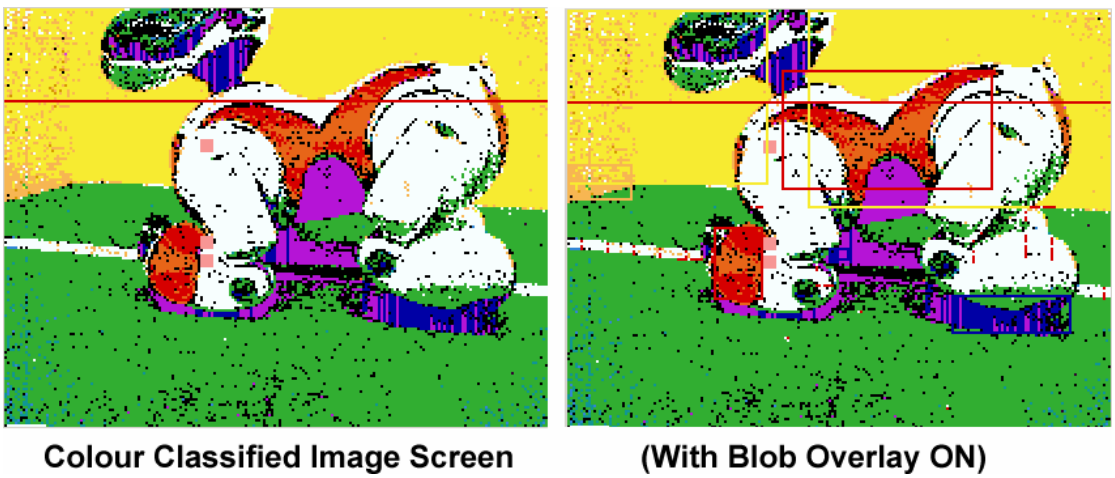


Figure 8-1: The scanned area is indicated by the solid pink squares. Since the colour green and orange are absent from the scanned area, the robot code would allocate the two red robot blobs to one robot container. These images are taken with the colour classification look up table on the 20-10-2005.

By making the code tighter using sorted blobs, the robot code has reduced by more than 170 lines.

8.3. Orientation Recognition

Orientation comes into play when the assigning code has finished. We take each robot container, and examine the information regarding the stored robot blobs. Information such as largest to smallest in area and highest to lowest in height are stored and calculated within the assigning code, ie performed on-the-fly. These information aids with the robot recognition code to determine their orientation. The robot recognition code attempts to distinguish the orientations: Left, Right, Front, Side, Back and Unknown using methods similarly described in Section 8.2. In particular scanning of the classified images contributes to the sanity factors that determine the robot's orientation.

Side orientation is when the ERS-7 robot has moved too closed to the side of another ERS-7 robot that it lacks insufficient information to determine whether the other ERS-7 robot is facing left or right. One reason is because the front leg patch is outside the image. Unknown orientation is necessary as not every pattern of robot blobs can be used to determine its orientation.

8.3.1. Robot RED

The orientation of robots with red uniforms is easier to tackle as red is a distinct colour and thus ideal red robot blobs are formed around patches of classified red pixels. This is illustrated in Figure 8-2

Figure 8-2a) Orientation: Left

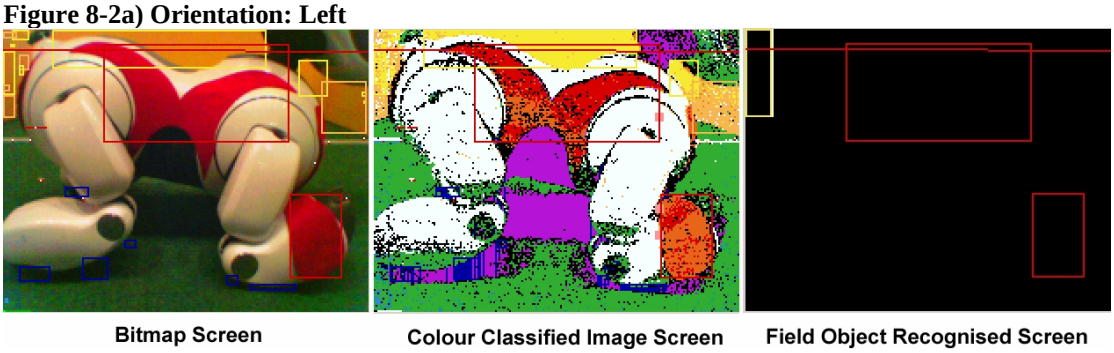


Figure 8-2b) Orientation: Right

Figure 8-2c) Orientation: Front

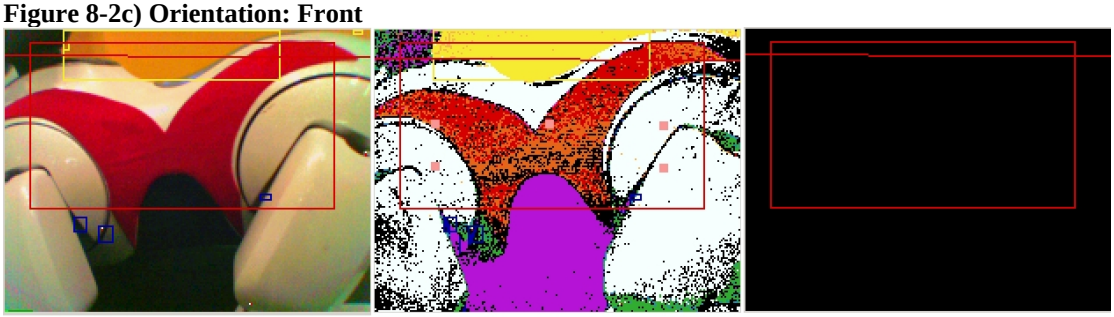


Figure 8-2d) Orientation: Side

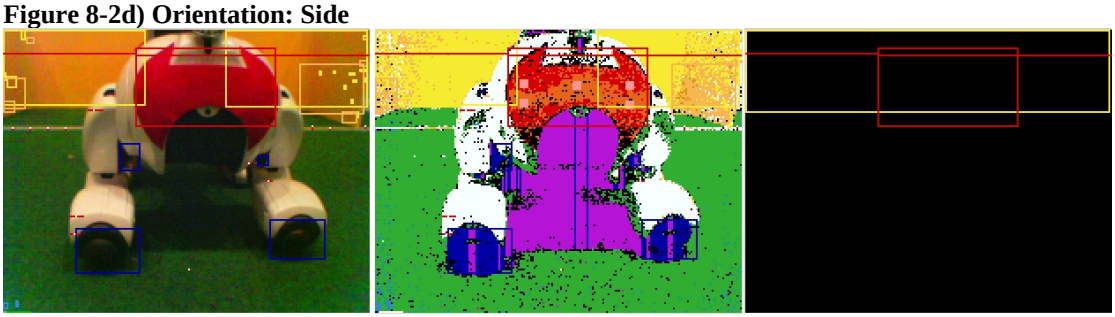


Figure 8-2e) Orientation: Back

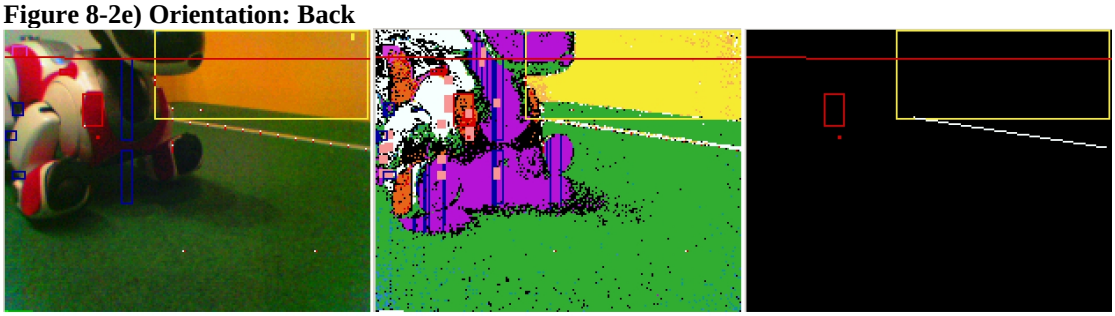


Figure 8-2f) Orientation: Unknown

Figure 8-2: Robot Red Recognition and Orientation. The scanned area is indicated by the solid pink squares. Robot blobs appearing in the objects screen means the robot has been recognised. As you can see each orientation has unique blob patterns which can be distinguished from each other. These images are taken with colour classification look up table on the 20-10-2005.

8.3.2. Robot BLUE

The orientation of robots with dark blue uniforms on the other hand doesn't produce ideal robot blobs, as the colour is hard to distinguish from other dark colours such as the grey parts on an ERS-7 robot and shadows on the green soccer field. This causes problems like classifying the blue uniform colours on the robots will cause the grey parts on the ERS-7 robot and shadows on the green soccer field to be regarded as the blue uniforms on the robot, and vice versa if shadow colours were used.

An improvement in the distinction is to build robot blue colours from shadow colours before passing through the blob formation stage. The way this is achieved is performed on the classified image, where we take a column of the classified image and run vertically downward. If there is a transition from white to shadow, from that point forth, replace each classified shadow pixel with that the blue robot pixel, til there is another transition from classified shadow colour to another classified colour, or end of the classified image [6]. This process is iterated through every column of the classified image. Afterwards, we run the robot recognition code, where the robot blue orientation component is an extension of the robot red orientation component. This is illustrated in Figure 8-3.

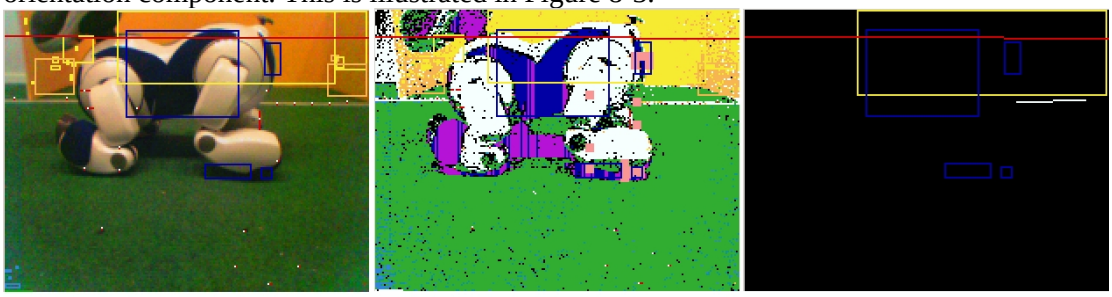


Figure 8-3a) Orientation: Left

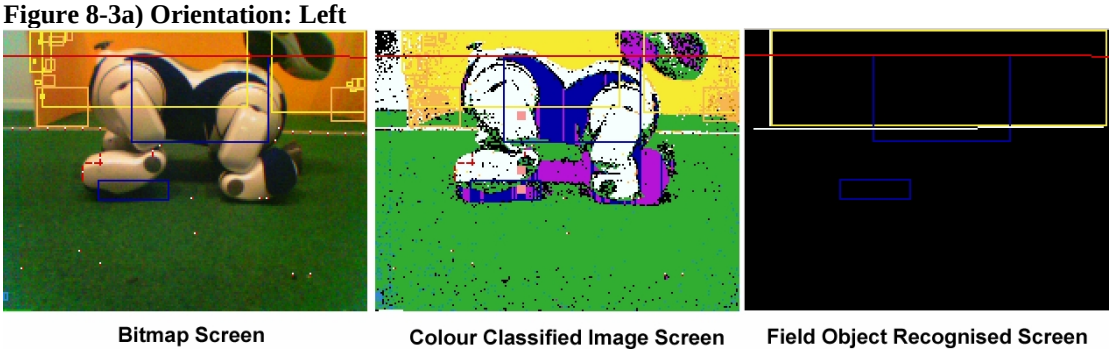


Figure 8-3b) Orientation: Right

Figure 8-3c) Orientation: Side

Figure 8-3d) Orientation: Front

Figure 8-3e) Orientation: Back

Figure 8-3: Robot Blue Recognition and Orientation. The scanned area is indicated by the solid pink squares. Robot blobs appearing in the objects screen means the robot has been recognised. As you can see each orientation has unique blob patterns which can be distinguished from each other. These images are taken with colour classification look up table on the 20-10-2005.

One of the most important aspects that the robot recognition code has to cater for is the situation that blue robot blobs being formed from shadow colours of red robots. Early versions of the robot recognition code did not cater for this, and robot blue was detected from the shadows of the robot red. Hence there was reporting of two robots on the field by the robot recognition code, where there was only one red robot.

8.4. Forming Robot Field Objects

Once an ERS-7 robot is recognised, a robot field object is constructed from the information of the stored robot blobs within a robot container. Similar to beacon field objects, the stored blobs are used to calculate the edges of the robot field object, and its centreX and centreY. Since we are introducing orientation as a new feature to robot recognition, we are required to have a calibration constant for each orientation for transform position to produce consistent distance calculations of robot field objects. (Refer to Section 7.5 on Transform Position and Calibration Constant)

8.5. Performance

So far the current evolution of robot recognition shows good results in recognising red robots. The orientation is slightly noisy. An implemented resolution was to average the orientation results by 3 frames, and choose the best orientation.

In the case of the blue robots, they are recognised by the robot recognition code; however the orientation requires more attention, in tuning and testing such that it can perform as well as with the red robots.

Similar to beacons, in cases when the robot's head is tilted, robots can still be detected, and its orientation determined. This is achieved by passing the robot blobs through the rotation algorithm, as is discussed in section 5.

Furthermore, noisy robot blobs are filtered using the filtering algorithm, as is discussed in section 6.

9. Prediction Algorithm

The prediction algorithm intends to evaluate future positions of moving field objects such as the ball and robots. This has been tackled by previous years using least squares method. The prediction algorithm this year traces out the curve movement of an object using past data, building on linear interpolation. The aim of the curve is to pin point a position which is closer to the actual object in the future than that which is given by linear interpolation.

The algorithm performance was evaluated and viewed in the 'World Model Debugger', discussed in Section 9.

9.1. Kalman Filter

The prediction algorithm this year relies on filtered velocities, comprising of two components. The filtered velocities are a result of being passed through the Kalman filter, implemented for localisation.

Definition:
"The **Kalman filter** is an efficient recursive filter which estimates the state of a dynamic system from a series of incomplete and noisy measurements. An example of an application would be to provide accurate continuously-updated information about the position and velocity of an object given only a sequence of observations about its position, each of which includes some error." [10]

9.2. Linear Interpolation

The linear interpolation of the prediction algorithm is a straightforward process. The input is given in seconds, and is multiplied with the velocities to obtain future distance offsets. The distance offsets are added to the current position of the object to obtain the predicted position. The predicted positions from the filtered velocities are not as good as one would hope it would be. This is illustrated in Figure 9-1.

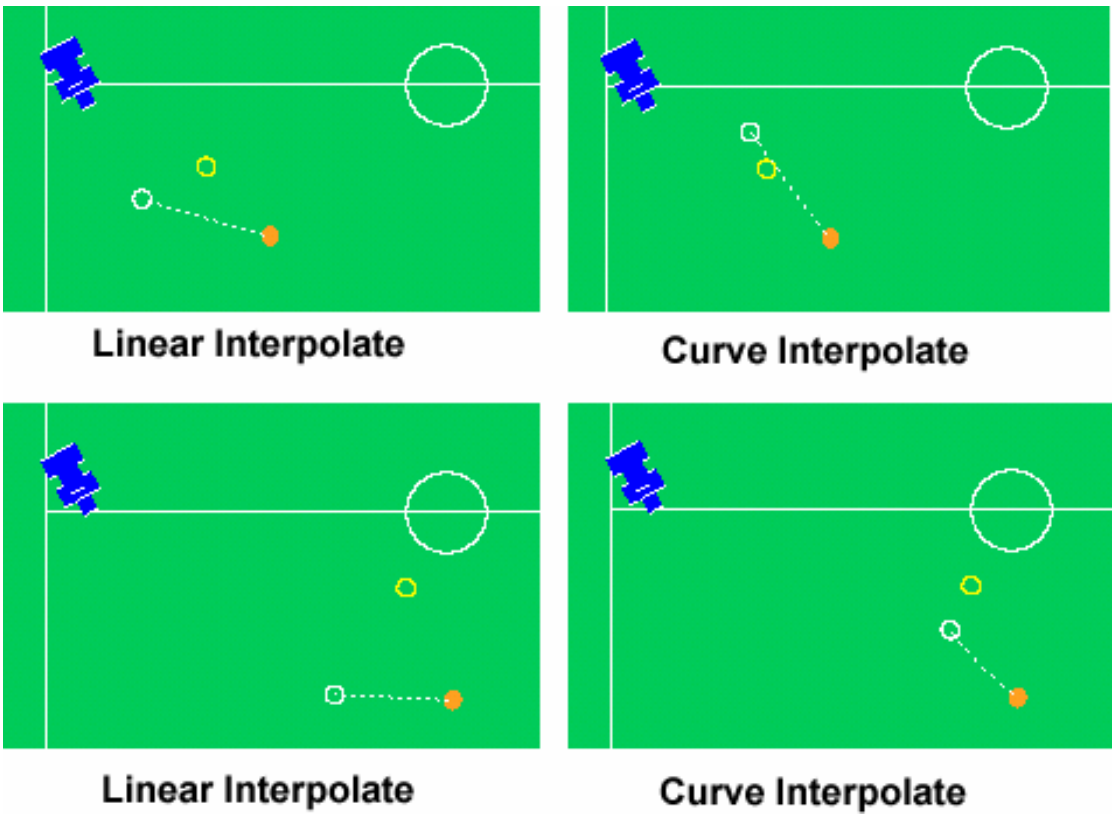


Figure 9-1: Linear Interpolate versus Curve Interpolate. Time Set = 1s in the future, UDP packet received at 1/5 second interval, circular data structure size = 7. Yellow Ball is the actual location. These images were taken from the 'World Model Debugger' on 29-10-05 by reading the log file of the information sent by the robots via UDP packets (using the WMD Packet format). Refer to section 10 for the WMD and section 10.3.2 for the WMD Packet.

9.3. Curve Interpolation

The curve interpolation of the prediction algorithm intends to give a better predicted position than that of linear interpolation using past velocities.

9.3.1. Operation

The algorithm works by finding a curve angle, and linearly interpolates by the number of successful past velocities which satisfy a filtration process.

The past velocities are taken from UDP packets which are stored in a circular data structure for the prediction algorithm. For Curve Interpolation, the prediction algorithm computes the angle differences, and places it in a linear array of size (n-1), where n is the number of UDP packets stored in the circular data structure. Curve Interpolation can only be called when the circular data structure is full.

The number of UDP packets required for curve interpolation varies. The algorithm is implemented to allow tuning of the number of UDP packets required. This is important as the current implementation relies on velocities of field objects taken from UDP packets that are emitted by the robots 5 times a second. If curve interpolation is to be ported into the robots, velocity data is available approximately 30 times per second, and hence the number of the past velocities in the prediction algorithm should be increased to allow a better curve angle to be calculated.

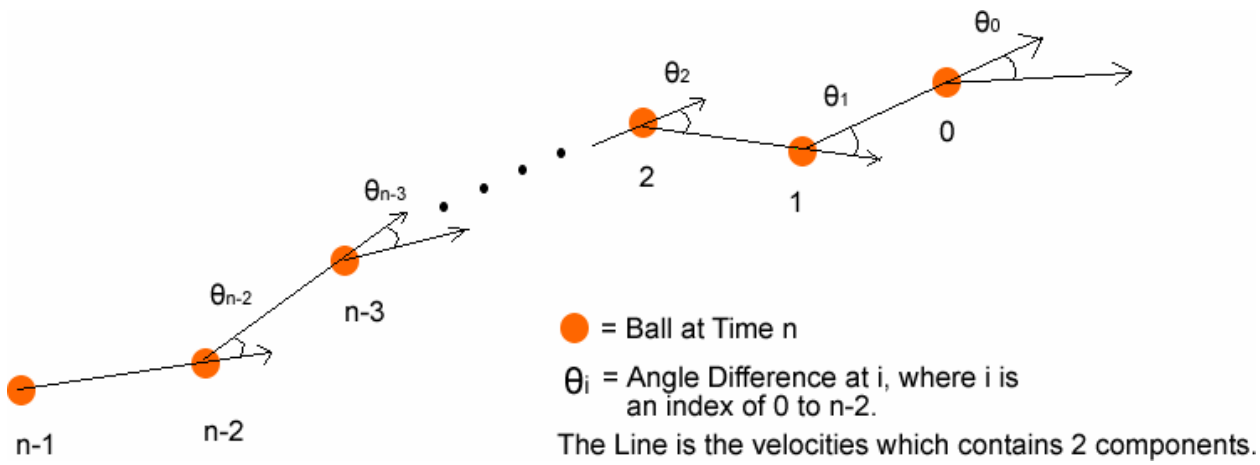


Figure 9-2: Curve Interpolation Process 'Angle difference calculation'. In addition, it computes the vector speed and initial vector angle which are calculated on past velocities that satisfy the filtration process from their angle differences. This is explained in Section 9.3.2 and 9.3.3.

9.3.2. Filtration

The algorithm begins by calculating each angle difference between every past velocity, and then they are filtered. The filtration process currently implemented is simple, but effective. How it decides which past velocities are eliminated can be best explained in the following pseudo algorithm.

BEGIN Filtration Process

1. Find the maximum angle difference and the minimum angle difference.
2. Calculate the midpoint angle difference from the maximum and minimum angle difference.
3. Determine the number of angle differences between the midpoint and the maximum angle difference and determine the number of angle differences between the midpoint and the minimum angle difference.
4. Ignore the angle differences which are fewer in numbers, either those that are above or those that are below the midpoint angle difference.

END Filtration Process

Once this is achieved, the remaining angle differences are used to calculate the curve angle. How this is performed is by averaging the angle differences over the past velocities that have not being eliminated in the filtering process.

9.3.3. Curve Angle, Vector Speed, Vector Angle

Having the curve angle is not enough to give the predicted position. In order for linear interpolation, we require a vector speed and an initial vector angle. The vector speed is calculated by averaging the two past velocities which gave the angle differences. They are then summed over all the averages. This final average is then divided by the number of angle differences that has not been eliminated, ie the sample space. The same method is applied in calculating the initial vector angle.

With the curve angle, vector speed and initial vector angle, we are able to obtain the predicted position, draw the interpolated curve, and see the result in the ‘World Model Debugger’ application.

9.4. Performance

As of current the prediction algorithm has being evaluated on the ball, and has not yet touched on the robots. The reason is because localisation has not included recognised or seen robots from vision to be counted in the Kalman Filter which is responsible in providing the filtered x, y coordinates, and the velocities. So far the Kalman Filter provides filtered information regarding the individual robots on the field, and the ball from vision. The results on the ball are good, and the curve interpolation shows a more favourable result over the linear interpolation in most cases.

9.5. Future Work

9.5.1. Filtration Process

The current filtration process is simple, yet effective. It can be more refined, either, in adding more simple rules to cover a wider range of situations, or rely on refining the Kalman Filter parameters, which is related to localisation. In addition, integrating the filtration process with sanity factors as implemented by the robots recognition code (Section 8.2) can improve on which velocities to ignore, as its logic is determined by confidence values assigned from the perspective of the developer.

9.5.2. Localisation

As mentioned in section 9.4, recognised robots from vision are not as of yet been included in the Kalman Filter. This means we do not have access to filtered x, y positions of recognised robots, nor their velocities, which is relied upon by the prediction algorithm.

Once we do have access to recognised robots on the field from vision, the prediction algorithm will work on another level of filtration process. The reason is the possibility that a ball in collision with a robot can either ‘bounce’ away from the robot or be grabbed by that robot should be taken into account.

9.5.3. Behaviour

The current behaviour system in the Four Legged League is relatively simple. For example, in the NUbot’s behaviour the most common action is to scan for the ball and who ever is closest will go for it and then aim the ball towards the goal. Team actions such as passing are lacking and rarely being looked into, due to time and other resources. Behaviour should be emphasised more for the up coming robocup, and by having partial reliance on prediction would provide a much more challenging game play against other teams.

10. World Model Debugger

The ‘World Model Debugger’ (aka WMD) visually shows the current object location from localisation, predicted positions of moving field objects [refer to Section 9 ‘Prediction Algorithm’], and the current behaviour being transmitted among the robots. In the case of robot recognition, the WMD would show their orientation. The previous WMD uses bitmap operations to draw the field objects, taking information from only one robot at a time, using TCP/IP protocol.

The approach for this year’s WMD is to use OpenGL operations for drawing, and allow processing of information from more than one robot. This is achieved by having the WMD listen for UDP packets from multiple ERS-7 robots on the field, which would be logged. In addition, images from an external camera would be taken to report the actual location of the field objects. The logging of the UDP packets and the actual location of the field objects provide valuable data in finding correlations to further improve tuning parameters among object recognition and localisation code; and it would allow better behaviour to be written.

10.1. Graphical User Interface

The graphical user interface for the WMD is created using the Microsoft Foundation Classes [aka MFC in short]. This is illustrated in Figure 10-1.

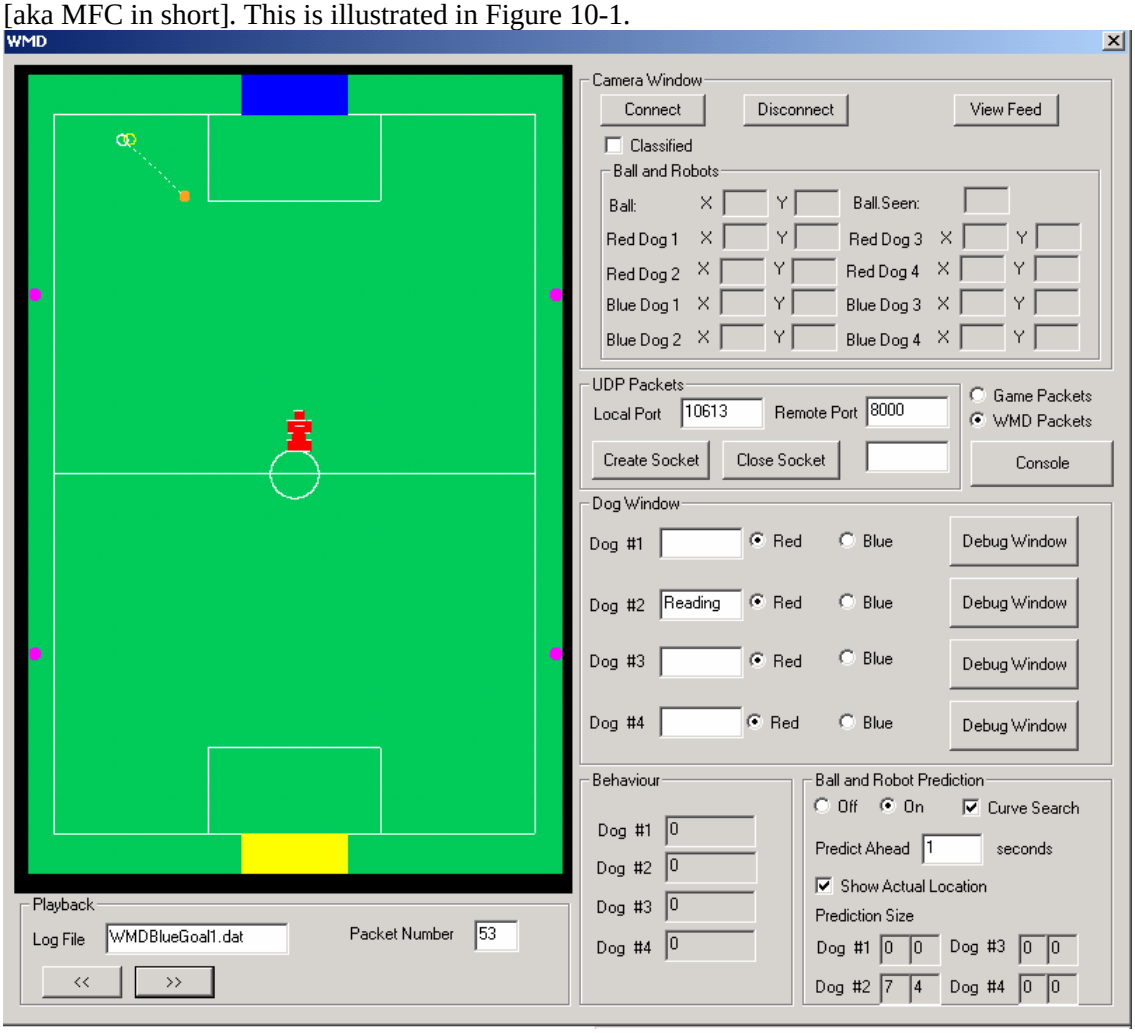
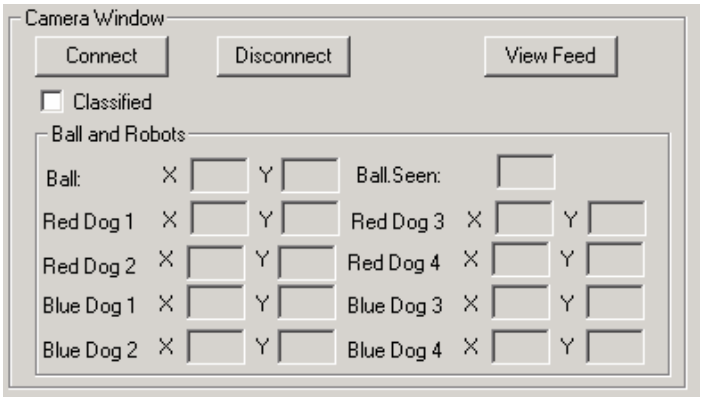


Figure 10-1: WMD Graphical User Interface. Current size of the circular data structure for the prediction algorithm is 7. Refer to Section 8.3 for more information.

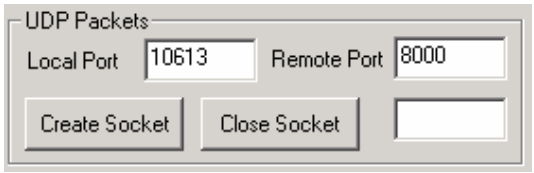
The features which the WMD has to offer are grouped in grey rectangular boxes. Their operations are explained in the following sections.

10.1.1. Camera Window



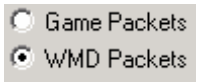
The 'Camera Window' is used for displaying seen objects from an external camera mounted above the field. To interface with the external camera, the user is required to click on 'View Feed' [5]. Once field objects are recognised their locations will be displayed in the 'Ball and Robots' group.

10.1.2. UDP Packets



The 'UDP Packets' group allows the user to begin listening for the robot packets by entering the local port which the packets are sent to, and then clicking on 'Create Socket'. Examining log packets can not be performed unless the socket is closed by clicking on 'Close Socket'. As of current the operation is examining the packets. If listening, then on the blank field will display the IP of the robot that sent the packet.

10.1.3. Packet Selection

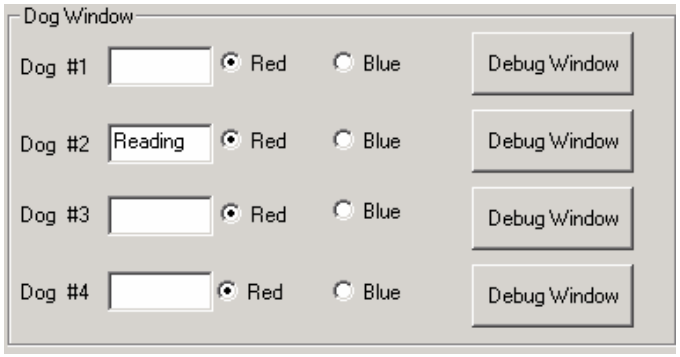


The packets that the robots use during a match is referred to as game packets, and they only send information regarding the sending robot and the ball seen by the sending robot. For the prediction algorithm to work, it was necessary to introduce the velocity information in the WMD. While making this change, we allowed all available field objects to be sent, rather than just two.

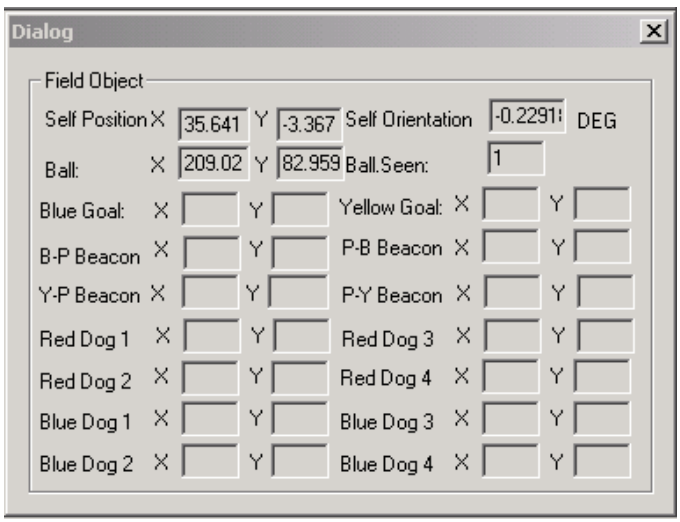
10.1.4. Console

Used for displaying information to debug the application by writing the operations that has been performed.

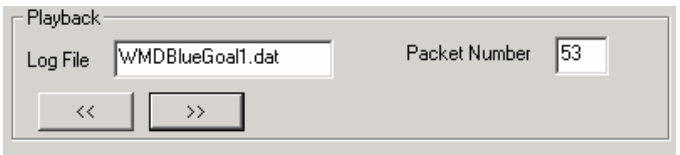
10.1.5. Dog Window



The 'Dog Window' indicates which UDP packets are currently being captured or read from a log file. When listening, it is necessary to choose the team colour as team information is not being sent. This is not the case when reading from a log file. The 'Debug Window' button pops up additional information regarding the packet, as shown below.

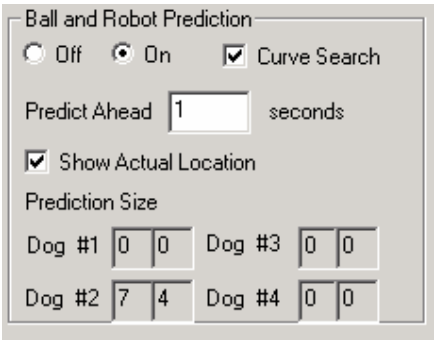


10.1.6. Playback



The 'Playback' window allows reading of a log file by entering the filename, and selecting if the format is a 'Game Packet' or 'WMD Packet'. The user may go directly to a packet without the sequential approach using the arrow buttons.

10.1.7. Ball and Robot Prediction



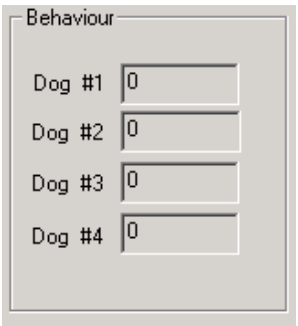
The 'Ball and Robot Prediction' allows the user to turn on or off the prediction algorithm. The prediction algorithm is only available when reading from a log file. The reason is because it allows for careful analysis of data, and provides time for developers to localise problems. Following are the features available:

- For linear interpolation the user is required to enter a positive number in the 'Predict Ahead' entry.
- For comparing with the actual location, the user is required to check the 'Show Actual Location' box.
- For curve interpolation, the user is required to check the 'Curve Search' box.

The 'Prediction Size' data is for use with curve interpolation. As mentioned in Section 9.3.1, the prediction algorithm can only function when the circular data structure is full. Hence the first field indicates the size of the circular data structure, and the second field indicates the current pointer to the circular data structure.

NOTE: the prediction algorithm works only with the WMD packets, not the Game packets.

10.1.8. Behaviour



The 'Behaviour' window shows the current behaviour messages that are sent among the robots. The current behaviour system is simple, and this is an indication of why it should be necessary to improve on. The current data that is sent in the behaviour part of the UDP packet is the distance of the ball in centimetres when it is seen by a robot. If not seen, then the distance is 10000.

10.2. OpenGL

The graphics for the WMD are created using OpenGL libraries. The reason behind this design decision is the less overhead in a PC machine equipped with an OpenGL graphics card over one that does not have one.

10.2.1. The Ball and Robots

The coordinates for the ball and robots are inverted depending on whether the teams are changed, and the orientation is 180 degrees off. As the OpenGL coordinates are different from the World Model coordinates, functions have being written to cater for these conversions.

10.3. UDP Packet

The early versions of WMD used the 'WinPcap: The Windows Packet Capture Library' to listen for UDP packets. The reason was due to experimenting with the 'EtherDetect Packet Sniffer Version 1.2' which was built using this library. In order to begin capturing UDP packets, the user would require selecting a network adapter and then click on connect. This is considered tedious, and the code to implement the UDP packet sniffing using the WinPcap library was very low level, however the UDP packets were captured in the early version of WMD.

An improvement was to use winsock libraries rather than WinPcap to perform the UDP packet sniffing. To begin listening for UDP packets, the user would only require entering the local port to which the robots are sending the UDP packets, and the code had the ease of modification if required.

10.3.1. Packet Interpretation

The interpretation of the UDP packets is a modification of the existing code for interpreting the UDP packets within the ERS-7 robot. The reason was because of the ease of changes if required, and it has already been tested before hand.

10.3.2. Game and WMD UDP Packets Log Format

The following is the format of the Game Packets for **each** line of the log file.

Attributes	COMMENT
msg->sourceBotID: {int}	Determine by the IP address: 10.0.1.31 - SourceID = 1 10.0.1.32 - SourceID = 2 10.0.1.33 - SourceID = 3 10.0.1.34 - SourceID = 4
self.colour: {int}	0 = RED , 1 = BLUE
self.x: {float} self.y: {float}	
self.orientation: {float}	
self.sdx: {float} self.sdy: {float} self.sdtheta: {float}	Standard Deviations
ball.seen: {int}	0 = Not Seen, 1 = Seen
ball.x: {float} ball.y: {float}	
ball.sdx: {float} ball.sdy: {float}	

The format of the WMD Packet is similar to the Game Packets; however it includes the velocities vX, vY of the ball, and the following field objects:

- Self and Ball
- All Four Beacons
- Both Goals and Goal Posts
- Team Robots
- Opponent Robots

Refer to Section 12.1 for the format of the WMD Packets.

Modification of NUbot's UDP Packet code was necessary to send out the WMD Packets.

10.4. External Camera

The WMD is a combined effort of another group member, Steven Nicklin. Steven is responsible for interfacing with the external camera, porting colour classification from NUbot's vision to be made compatible with the resolution of the external camera, then calling the OpenGL functions for drawing the actual locations of the field objects [5].

To activate the External Camera, the user would require clicking on 'View Feed' which opens a dialog that allows the video streaming, colour classification, blob formation and recognition to be observed [5].

10.5. Performance

The current performance of WMD is good, and its functionality is as required.

10.6. Encountered Problems

10.6.1. MFC threading

One of the main concerns during the WMD development was with MFC threading. The WMD application gave run time errors whenever an MFC thread attempted to access or mutate a dialog variable. The run time error was because MFC is not thread safe in object level but is on class level. This problem was resolved by creating an object handle, HWND, for the MFC thread. Having the ability to access dialog variables by MFC threads makes the WMD application more responsive. For example, logging data is a thread. Whenever a packet is read, it would write 'Reading' within the blank field of the 'Dog Window', as shown in the figure of section 10.1.5.

10.6.2. fprintf and fscanf

The current logging of the UDP packets (using fprint) is integers and floats. This means that reading the packets (using fscanf) require integers and floats; however the actual Game Packets and WMD Packets use doubles. The reason was to remain consistent with the Game Packets among the robots. Therefore, in order to begin using the WMD Packets from the log file, they would have to be taken in as floats, and then convert to doubles (so as to reduce precision warnings). Initial attempts of writing and reading using doubles caused incorrect numbers (and substantially long numbers) to be written to the log file and incorrect numbers when read from the log file.

10.7. Future Work

10.7.1. Robot Coach

The Robot Coach is intended to offer corrections to the developer whenever an actual location does not match closely to a location reported by vision. It can be further developed to provide a crude form of online supervised learning, such that the robots should be informed that their current parameters should be corrected, either manually or automatically. Due to time constraints (taking into account of debugging and testing) this was not tackled.

10.7.2. Mapping Real Field Over OpenGL Field

This feature intends on overlaying the real field from that taken by the external camera over the OpenGL field. It would allow the user to switch from observing the data within the OpenGL field to the real field provided by the external camera within the WMD application. This was not tackled because the external camera is scheduled for installation beyond the time of this project.

10.7.3. Display Lists

An improvement in performance of any OpenGL application is using display lists. The reason there is an improvement is that the OpenGL commands within the display lists are 'precompiled', and if a user has an OpenGL graphics card, these commands resides in the graphics memory, hence execution would be faster, (up to 3 times quicker in frames per second). If for any reason, the WMD application appears to be drawing slow, then converting a partial or full amount of OpenGL commands to display lists is advisable. This is another reason why OpenGL was chosen rather than using standard bitmap operations.

10.7.4. OpenGL Fonts

This feature is useful whenever there is more than one robot sending UDP packets. NEHE's lesson 13 (http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=13) allows text to be mapped within OpenGL. By implementing OpenGL fonts, the WMD application can display the numbers of the robots within the WMD.

10.7.5. Log File Functions

Make query functions on the log file available. For example, the user can query on a particular robot if more than one has been logged, and display the results in the WMD.

10.7.6. WMD Packet Functions

Since the WMD Packets contains other field objects, the data gather should be of use. Future developers should make extensions to the WMD, such that these data are not left redundant.

11. REFERENCES

1. *Robocup Official Site*. 1998-2005, The Robocup Federation. http://www.robocup.org/
2. *ERS-7 Specifications*. 2005, Sony Entertainment Robot Europe. http://www.aibo-europe.com/1_1_3_ers7_specifications.asp
3. *ERS-7 Entertainment Robot AIBO Product Tour*. 2005, Sony Electronics e-Solutions Company LLC. http://www.sonystyle.com/intershoproot/eCS/Store/en/imagesProducts/ProductTour/computing/ers7/tour01.html
4. Chalup, S., *Newcastle Robotics Laboratory*. 2002-2005. http://robots.newcastle.edu.au/
5. Nicklin, S., *NUbots: Object Recognition in Robotic Soccer*. 2005, University of Newcastle.
6. Henderson, N., *Colour Classification in Robotic Vision*. 2005, University of Newcastle.
7. Röfer, T., et al., *German Team Robocup 2004*. 2004, German Team.
8. Niku, S.B., *Introduction to Robotics : Analysis, Systems, Applications*. 2001: Prentice Hall.
9. Stewart, J., *Calculus*. 4th ed. 1999: Brooks/Cole.
10. *Wikipedia - The Free Encyclopedia*. http://en.wikipedia.org/

12. APPENDIX A

12.1. WMD Packet Format

The following is the format of the WMD Packets for **each** line of the log file.

Attributes	**COMMENT**
msg->sourceBotID: {int}	Determine by the IP address: 10.0.1.35 - SourceID = 1 10.0.1.36 - SourceID = 2 10.0.1.37 - SourceID = 3 10.0.1.38 - SourceID = 4
self.colour: {int}	0 = RED , 1 = BLUE
self.x: {float}	
self.y: {float}	
self.orientation: {float}	
self.sdx: {float}	
self.sdy: {float}	
self.sdtheta: {float}	
ball.seen: {int}	0 = Not Seen, 1 = Seen
ball.x: {float}	
ball.y: {float}	
ball.sdx: {float}	
ball.sdy: {float}	
ball.vX: {float}	X velocity of Ball
ball.vY: {float}	Y velocity of Ball
pyBeacon.seen: {int} pyBeacon.x: {float} pyBeacon.y: {float}	Pink Yellow Beacon
pbBeacon.seen: {int} pbBeacon.x: {float} pbBeacon.y: {float}	Pink Blue Beacon
ypBeacon.seen: {int} ypBeacon.x: {float} ypBeacon.y: {float}	Yellow Pink Beacon
bpBeacon.seen: {int} bpBeacon.x: {float} bpBeacon.y: {float}	Blue Pink Beacon
blueGoal.seen: {int} blueGoal.x: {float} blueGoal.y: {float}	Blue Goal
LblueGoalPost.seen: {int}	Left Blue Goal Post
LblueGoalPost.x: {float} LblueGoalPost.y: {float}	
RblueGoalPost.seen: {int}	Right Blue Goal Post
RblueGoalPost.x: {float} RblueGoalPost.y: {float}	
yellowGoal.seen: {int} yellowGoal.x: {float} yellowGoal.y: {float}	Yellow Goal
LyellowGoalPost.seen: {int}	Left Yellow Goal Post
LyellowGoalPost.x: {float} LyellowGoalPost.y: {float}	

RyellowGoalPost.seen: {int} RyellowGoalPost.x: {float} RyellowGoalPost.y: {float}	Right Yellow Goal Post
team1.seen: {int} team1.x: {float} team1.y: {float}	Team Robot 1
team2.seen: {int} team2.x: {float} team2.y: {float}	Team Robot 2
team3.seen: {int} team3.x: {float} team3.y: {float}	Team Robot 3
team4.seen: {int} team4.x: {float} team4.y: {float}	Team Robot 4
oppon1.seen: {int} oppon1.x: {float} oppon1.y: {float}	Opponent Robot 1
oppon2.seen: {int} oppon2.x: {float} oppon2.y: {float}	Opponent Robot 2
oppon3.seen: {int} oppon3.x: {float} oppon3.y: {float}	Opponent Robot 3
oppon4.seen: {int} oppon4.x: {float} oppon4.y: {float}	Opponent Robot 4

13. APPENDIX B

13.1. TEAM MEMBERS

Name: Kenny Hong
Student Number: 2103674
Degree: Bachelor of Combined Computer Engineering / Computer Science

Name: Steven Nicklin
Student Number: 3008009
Degree: Bachelor of Computer Engineering

13.2. SUPERVISION

Professor Rick Middleton would be our supervisor for the project as he is one of the academic chief investigators of the Newcastle Robotics Laboratory, and that we have spoken with him about our project.

In Semester 2 of 2005, Professor Rick Middleton was on sabbatical. Co-supervisor for that period is Dr Lu Gan.

13.3. COMMERCIAL CONFIDENTIALITY and INTELLECTUAL PROPERTY

This project has no commercial confidentiality as the NUbot is a University Based Team. Following the Robocup Competition, the source code is released into the public domain.

Intellectual property for this project is so far unknown.

13.4. EQUIPMENT AND CONSUMABLES

The robots are currently available. A high quality camera has been bought to overlook the entire field. This camera is used for the ‘World Model Debugger’ application.

Any additional hardware would likely be supplied by the Newcastle Robotics Laboratory.