THE UNIVERSITY OF NEW SOUTH WALES
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Multi-modal Machine-learned Robot Detection for RoboCup SPL

Jimmy Kurniawan

Software Engineering

<z3249563 jimmyk@cse.unsw.edu.au>

August 24, 2011

Supervisor: Bernhard Hengst
Assessor: Maurice Pagnucco

School of Computer Science & Engineering
University of New South Wales
Sydney 2052, Australia

Abstract

RoboCup continues to inspire and motivate the research interests in cognitive robotics and machine learning. The 2011 rUNSWift team carried out a major overhaul to the vision system and made significant improvement to localisation and locomotion. Many innovations had been introduced to behaviours and the rUNSWift code base to promote modularity.

Major re-implementation was made to the Robot Detection module to ensure compatibility with the new vision architecture as well as introducing two major innovations, machine learning the decision making process and the incorporation of multi-modal system.

This thesis report describes the research and development undertaken by Jimmy Kurniawan in developing the Robot Detection module in the vision system, including the associated modules in Sonar and Robot Filter as well as behaviours for 2011 RoboCup Competition.

Acknowledgements

First and foremost, I would like to give my utmost gratitude to Bernhard Hengst, my supervisor, for his constant support and assistance for the duration of my thesis project. Many of the ideas and milestones would not have been reached if it was not for his lend of hand. Bernhard has been a man of incredible knowledge and marvelous heart. There was not a single discussion that did not completely change my perspective and gave me a deeper insight into my project.

Secondly, to the 2011 rUNSWift development team: Sean Harris, David Claridge, Belinda Teh, Carl Chatfield, Brock White, Yongki Yusmanthia, Benjamin Vance and Hung Duc Nguyen. There was not a single day where I did not enjoy having them around in the robotics lab. We had been through many stressful moments, many laughs and many successes together. Who knew we could go this far since the first friendly game with the NUBots where we lost very badly despite our initial confidence.

Next is to Claude Sammut, Jayen Ashar, Brad Hall and Maurice Pugnucco. The rUNSWift team would not have been possible without them. Jayen has always been of tremendous help in the lab with day-to-day problems. Brad has facilitated and promoted many aspects of the rUNSWift team and made them they way they are today. Maurice and Claude have supported the team since day one and have always believed in the rUNSWift team. Despite the occasional cryptic clues given during the development, it was enough to push us over the wall.

I would also like to thank my family, currently in my home country, Indonesia, for their support from afar. I certainly would never have gotten this far in my career without them and their support since the first day I arrived in Australia.

Last, but not least, I would like to thank my long-term girlfriend, Monita, for her love, support, patience as the victim of my torturous technical discussion and free food delivery during the 10 months or so duration of my thesis project. Without her, the success of this thesis would not have been possible.

Contents

List of Figures

Chapter 1

Introduction

1.1 Opening

RoboCup [8] is an international scientific initiative with the goal to advance the state of the art of intelligent robots, particularly in the area of autonomous soccer robots. The Standard Platform League (SPL) is one of several leagues within RoboCup, where teams use identical robot platform, called 'Nao' by Aldebaran, to play soccer.

Artificial intelligence in the area of robotics faces many challenges, with specialities ranging from world-perception, locomotion and vision. The primary area of research in this thesis project is on the development of a computer vision algorithm. Specifically, on the need to develop a robust Robot Detection algorithm for the 2011 rUNSWift team in the upcoming RoboCup competition.

1.2 Research Problems and Issues

Typically, there are multiple objects on the field that are highly desirable to be detected. These objects provide vital information for the robot, particularly in the higher level decision making behaviour. Such objects may include the ball, goal posts, field lines and other robots.

Since the robots must be autonomous and are not aided by human-operators or any other machines around the field, it has always been difficult to develop a general purpose algorithm for each modules. In this research project, one such module is Robot Detection.

The definition of Robot Detection is a vision algorithm that attempts to detect and identify the shape of a robot on the SPL field. It is also desirable to produce enough information required to calculate the relative distance and heading from the viewing robot, so that it can be used in other modules, such as the robot filter.

Thus, the primary research problem is the need to develop a robust robot detection algorithm that is fast and able to run in a limited resource environment, such as the Nao robots.

1.3 Justification of the Research

Being able to detect other robots on the field, just as being able to detect any other objects, provide great benefit to the general game-playing strategy. Specifically to this research project, Robot Detection would allow the following advantages:

- **High level path planning** - The idea is to plan ahead of time an optimal path that optimally approach the ball while avoid objects (other robots) in the process.
- **Robot avoidance**.
- **Strategy play** - such as passing the ball to an ally robot.
- The idea of utilising the **area of largest empty space** for various behaviours, such as positioning and passing the ball.

On the other hand, the ill consequence of not having a Robot Detection will have shortcomings such as:

- Bumping into other robots, causing locomotion odometry to fail or slip.
- Kicking the ball into the wrong direction or kicking into other robots, which may give the opponents better positioning, thus disadvantaging the team.
- Inability to have optimal global positioning, such as when the ball is being kicked into an ally robot and bounced off the field, where the team will be penalised.

1.4 Aim and Delimitation

The primary aim of the thesis project is to machine learned the presence of a robot and to be able use the result from the machine learner to develop a robust Robot Detection algorithm. The specific feature or implementation of the algorithm often changes from time to time as more requirements are realised.

On the very least, the basic functionality that must be satisfied is to provide the information needed to calculate the relative coordinate from the viewing robot to the detected robot.

The intention in the development of the algorithm shift from a simple leg detection to a proper full body detection to include the waistband, while the mechanism shift from a multi-modal system with predefined parameters to a machine learned decision tree to determine such parameters.

The delimitation of this research project will not include the following:

- Pose estimation which is the ability to tell which way the detected robot is facing relative to the viewing robot.
- Sophisticated robot tracking and path estimation or modelling of motion.

1.5 Methodology

The primary methodology in carrying out this thesis project is by using an agile-style programming practice. The idea is to be able to adapt to changes when new requirement arises and it is common to be consistently changing the requirements as deeper understanding of the software architecture and hardware inner working are acquired. Thus the algorithm is routinely modified to reflect the needed changes.

In addition, Bernhard Hengst philosophy of "Fail Often, Fail Fast" will also be used. It is the idea to have constant, fast practical experimentation as opposed to lengthy theoretical implications [6].

1.6 Outline

This research report will be structured in chapters starting with background and literature reviews, and subsequently the major development of each module in the project, starting from the most significant Robot Detection algorithm and the machine learned decision tree to the sonar and robot filters and the development on behaviours for RoboCup 2011.

Each chapter contains a problem outline of the specific module, how it fits together with the entire software architecture, as well as including the experimentation methods and testing results.

The chapter in behaviours will detail over the motivation on the design of the behaviours, the compromises considered and the effectiveness of the behaviours.

1.7 Definitions

A number of specific and technical terms are used in this thesis report. The following table contains the definition of the terms:

Term	**Definition**
Absolute space	The coordinate space that uses X and Y to refer to a particular position on the actual RoboCup SPL fields.
Colour saliency scan	The saliency scan with all the pixels converted into pre-defined pre-calibrated colours.
Frame	Typically one iteration of a particular process. Most commonly referring to the vision process, where the process performances are measured in frames per second.
Heading	The angle between two points using a specified north direction.
Image space	The coordinate space that uses X and Y to refer a particular pixel or area of pixels on a visual frame.
Relative space	The coordinate space or system that uses relative distance and relative heading to refer to an object on the image from the viewing robot.
Saliency scan	Typically a down-sampled visual frame that is smaller in resolution than the raw frame.
Visual frame	An image

Chapter 2

Background

2.1 Introduction

This chapter will outline the various background and literature materials used in developing the Robot Detection algorithm. The materials consulted prior to the commencement of the project include:

- 2010 rUNSWift Report.
- 2010 B-Human Report.
- Fabisch-Laue-Rofer Robot Recognition and Modelling Paper.

This chapter will provide an analysis of the methods used in the above materials and attempt to find a gap which needs to be filled in terms of the development of Robot Detection as a whole.

2.2 Literature Reviews

2.2.1 2010 rUNSWift report

The 2010 rUNSWift [1] vision code base makes heavy use of region building. The idea of region building is basically to identify potential areas of interest on the field, through using groups of coloured pixels and its shape to determine what the region most likely represents. This process is primarily a scanline procedure over the visual frame.

In specific to Robot Detection, the algorithm commonly finds a group of white region around the foot area and are joined together into a single bounding box. Given the position of the bounding box in image coordinate, the algorithm subsequently attempts to find the waistbands, which are also tagged by the region builder as a region of interest. Finally, candidate regions that successfully pass the sanity checks will be returned as robot regions, while others are discarded. Notice that one of the majority clue verified by the sanity checks is the existence of waistbands. The figure below shows the result of the Robot Detection in the 2010 rUNSWift vision system.

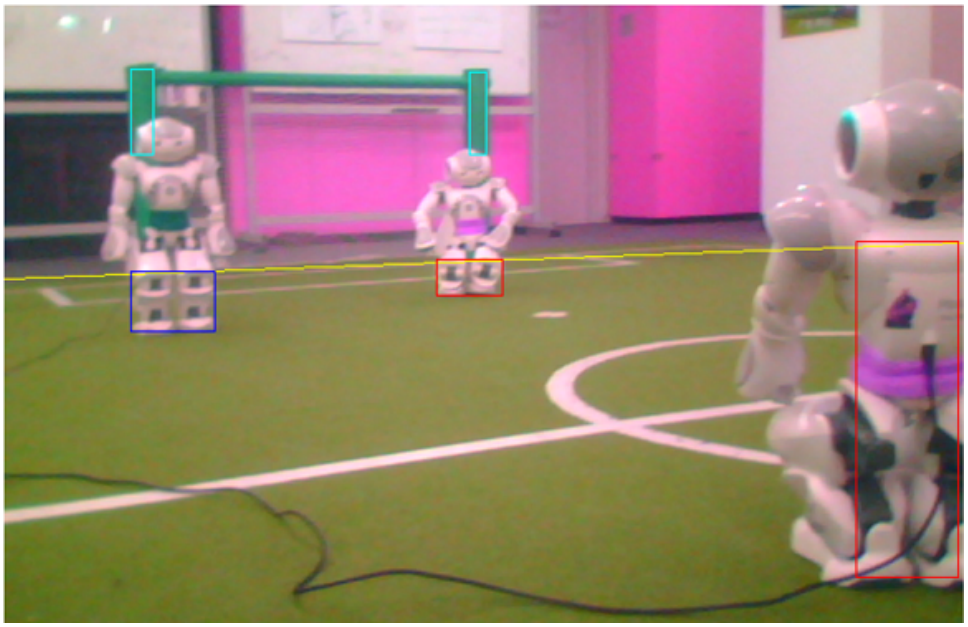

Figure 2.1: 2010 rUNSWift Robot Detection
that shows the result of the algorithm

Please refer to the 2010 rUNSWift report for more details [1].

2.2.2 2010 B-Human report

The 2010 B-Human [12] vision code base deploys similar strategy to the 2010 rUNSWift vision code. The method proposed by B-Human makes use of colour segmentation, which then proceed to build a set of regions through the grouping of colours.

The B-Human Robot Detection placed higher emphasis on the waistband regions as it attempts to find the orientation of the waistband and subsequently use this information to conduct various horizontal and vertical sanity checks to verify the surrounding colours. The candidate region is then returned as a result or discarded when the conditions could not be satisfied. Similarly, notice how this algorithm makes heavy use of the waistband region during the detection. The figure below shows the result of the Robot Detection in the 2010 B-Human vision system.

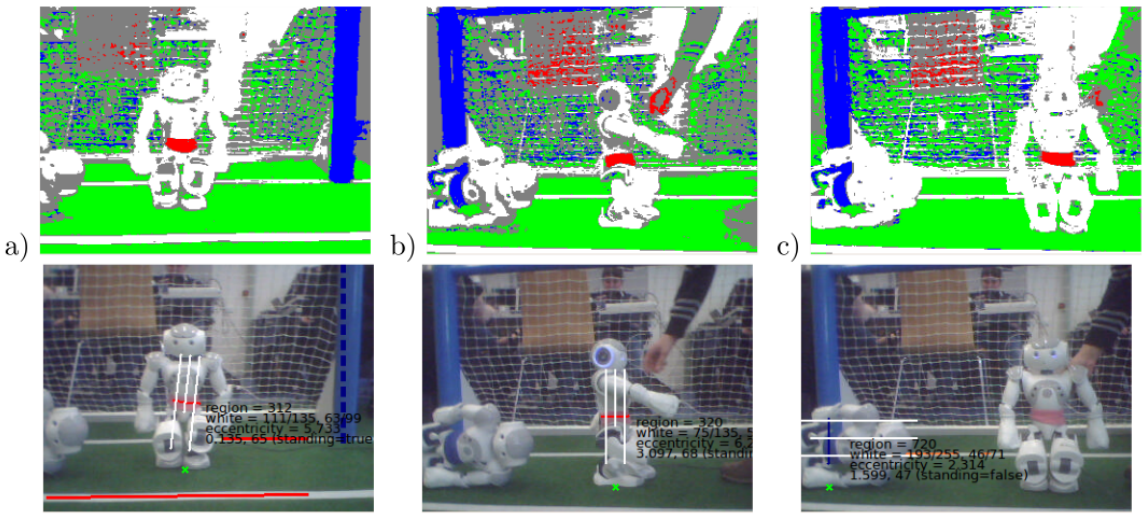

Figure 2.2: 2010 B-Human Robot Detection
that shows the result of the algorithm

Please refer to the 2010 B-Human report for more details [12].

2.2.3 Fabisch-Laue-Rofer Robot Recognition and Modelling Paper

The idea proposed by this paper [2] in Robot Detection uses B-Human [12] vision code base as a starting point and extensions are implemented on top of the code to include the more advanced functionality. The goal of the paper consist of 2 parts, robot recognition and robot tracking. Only robot recognition will be deeply analysed as it is the major segment of this thesis project.

The methodology implemented by this algorithm in the paper utlises the waistband region detected by the B-Human's region builder. Given the middle point of the waistband, the algorithm attempts to search for the convex hull of the waistband through vertical and horizontal scan in a higher resolution. The availability of the convex hull will allow a more accurate estimation of the orientation of the waistband. Subsequently, given the orientation, the algorithm proceeds with a number of scan lines on the top and bottom of the waistband region that attempts to check for surrounding colours and overall shape which essentially verifying the existence of other robots. Once again, take note on the heavy emphasis on the need for visible waistbands. The figure below shows the result of the algorithm.

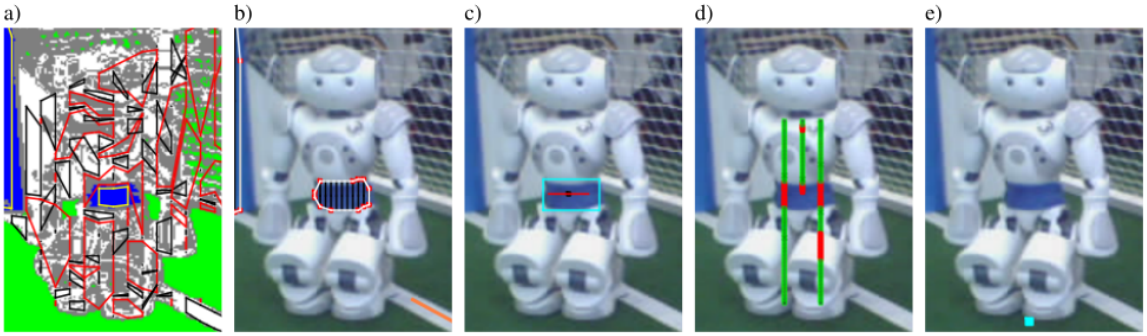

Figure 2.3: Fabisch-Laue-Rofer Robot Recognition
that shows the result of the algorithm

Please refer to the related paper for more details [2].

2.3 Common Ideas

The common idea amongst the different implementations deployed in the materials above primarily revolve around the following features:

- Generating regions from region builder.
- Running sanity checks on regions to verify the existence of robots.
- Heavy emphasis on the waistband where it is treated as the biggest clue when attempting to detect robots in the visual frame.

These common features exist due to the direct tie with the other modules in the vision system. Typically, the region builder attempts to find all the interesting regions and give such regions to each individual vision module, such as ball detection or goal posts detection, to be processed separately. Thus, having a system, such as this one, allow vision to run relatively fast on an embedded device such as the Nao robots. This essentially allows the overall system to run optimally at the targeted 30 frames per second.

2.4 Relevance

The materials reviewed are primarily on the area of Computer Vision with specific focus in Robot Detection, which is in the same area of interest of this thesis project. The methodology proposed and the algorithms implemented make use of similar common features (particularly the sanity checks) to achieve the performance required to run on the Nao robots.

2.5 Emergence of Necessity

The major problem with the methodology proposed in the materials above are their heavy reliance on the waistbands. All techniques treat the waistband as the central starting point of the algorithm or take the waistband as the biggest clue when detecting a robot. In other words, if there are no waistbands, then the candidate will be discarded and no result will be produced.

However, it is not uncommon too encounter scenarios where the waistbands are not visible. For example, when the viewing robot does not have a full body view of the other robots or when the waistbands are obstructed by other objects, such as the arms of the robot or by other robots. In scenarios like these, no results are produced. It may prove to be fatal during the game and would place the team in a disadvantaged position.

Thus the necessity emerged from this problem is to have an algorithm that is able to detect other robots despite not having the waistbands in the visual frame. Implementing an algorithm that treats the waistband as just one of the many clues or evidences when attempting to detect the robot is the gap needed to be filled. The overall necessity is to have a full body robot detection.

Figure 2.4: The raw image of obstructed waistband. The waistband is still visible in this image as the image is in much a higher resolution than the saliency image.

Figure 2.5: The saliency image of obstructed waistband. The waistband is barely visible on the middle robot, while the waistbands at the other robots on the back are completely obstructed and unable to be seen.

2.6 Conclusions

In conclusion, in the materials reviewed, all techniques make sure of region builder and have heavy emphasis on the use of waistband information. The problem is in the scenarios where waistbands are not visible and thus results are not able to be produced. The gap is then to employ an algorithm that attempts to have a full body Robot Detection, which takes the waistband information as just one of the many clues available.

Chapter 3

Robot Detection

3.1 Introduction

This chapter will introduce the inner working of the Robot Detection algorithm. The algorithm comprises several processing and logical steps, interleaved with a number of sanity checks. The sections will be broken down into parts that are analogous the way the algorithm is implemented into the rUNSWift 2011 code base. The outline of this chapter goes as follow: satisfying the prerequisites, generating result candidates, bounding of candidates and finally applying the decision tree.

3.2 Concept

The basic premise utilised by the algorithm when attempting to detect a robot is simple. The idea fits in well with the 2011 vision code base [3] where all the information required are already available. The following is the simple step-by-step on how the algorithm works:

- In a visual frame, attempt to detect the edges of the field.
- If the field edges are found, then draw a line along the edge line with the condition that the group of vertical coloured pixels between this line and the edge line must not contain any green pixels. In other words, draw the line along the first green pixel seen from the edge line.
- The shape of this line will either touch on the edge line (meaning that the first green pixel is the first pixel seen) or create an indentation from the edge line (meaning that the first green pixel is some distance away from the edge line). When such indentations occur, it is a sign that a candidate robot region exists.
- Attempt to bound close regions together and run several sanity checks to finally verify the existence of other robots in the visual frame.

Figure 3.1: Concept of robot detection that shows the basic premise on how it works. The blue line is the line generated from running the steps above. Notice the indentation near the robot.

Obviously, the above descriptions on the concept are highly simplified since it is only meant to show the basic premise of how the algorithm works. Notice that no indentations are created around the yellow goal post or the net behind the goal posts. These are one of the many sanity checks that will be implemented into the algorithm that takes care of removing early non-robot regions.

3.3 Prerequisites

Prior to executing the Robot Detection algorithm, which will be outlined in the next section, several preconditions must be met. These conditions ensure the existence of visual data and any erroneous visual data can be discarded as running the algorithm with such data might not produce a sensible result. In addition, the data output from several of these preconditions also provide the necessary data about a particular visual frame, where it can be of assistance to the algorithm, allowing a more accurate result. These preconditions include:

- **Colour Calibrated Visual Frame** - The colour calibrated visual frame [3] [1] (referred to as the colour calibrated saliency scan) is one of the two working base (the other one being the edge saliency scan) for vision processing. The saliency scan is down-sampled from the original resolution of 640x480 to a much smaller resolution of 80x60, the individual pixels are then classified into several pre-defined colour using a pre-calibrated colour table. The figure below is an example of a colour calibrated visual frame.

Figure 3.2: Colour-calibrated saliency scan that shows the type of data the vision system is processing.

- **Kinematics and Camera Transformation Matrix** - The functionality provided by kinematics [1] are crucial as it provides the matrix transformation from image space to relative space. In other words, given an x and y coordinate on the image, the point can be translated into the relative distance and heading from the viewing robot to the specified point on the image. For more detail regarding this functionality, please refer to the 2010 rUNSWift report [1].

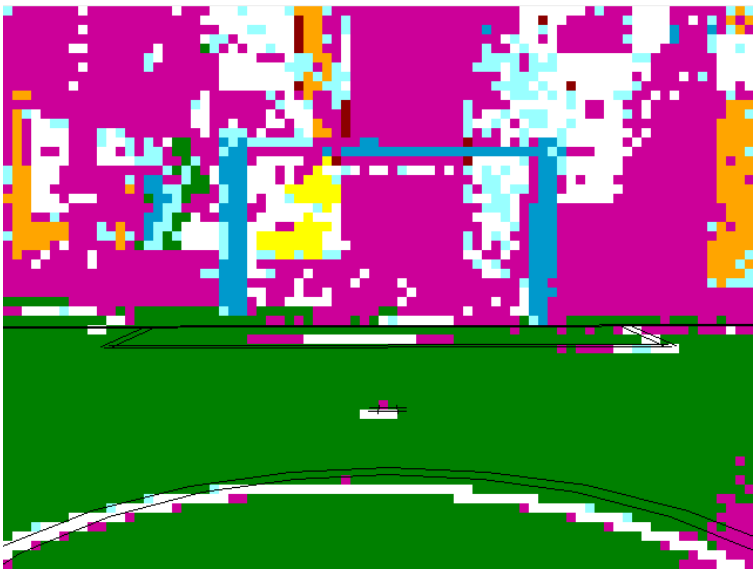

Figure 3.3: Kinematics chain that allows the generation of camera transformation matrix for the use of coordinates translation from image space to relative space.

- **Field Edge Detection** - The subsequent step in the vision pipeline after acquiring a colour classified saliency scan is to carry out the field edge detection [3]. The field edge information is important to the Robot Detection algorithm as outlined in the concept section. It also helps in reducing the search space, thus improving computing performance and removing unnecessary noises. The figure below is an example of the result from the detection.

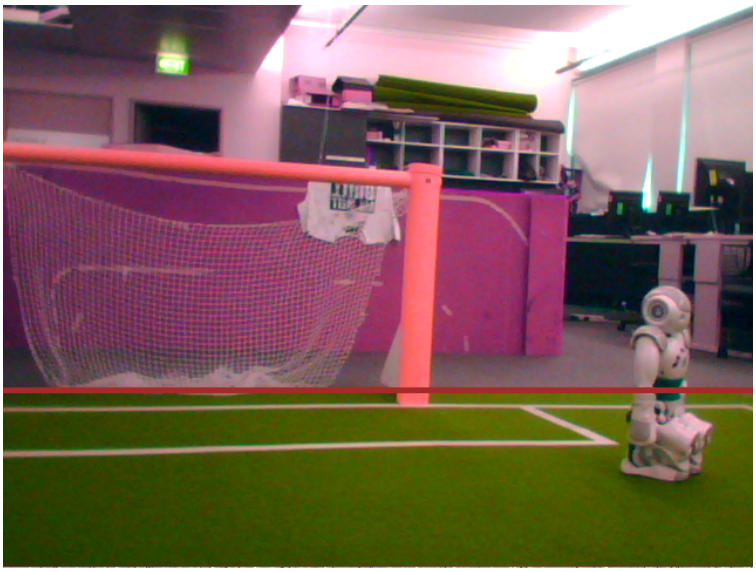

Figure 3.4: Result of Field Edge Detection.
The brown line represents the detected edge of the field.

- **Field Line Detection** - The field line detection [5] is required in providing the algorithm with the necessary input data to ensure that the distance and heading calculation is correct. Robots may be stepping on the white lines and without the field line detector, the bounding box may be extended and thus would provide an inaccurate result. The figure below is an example of the result from the field line detection.

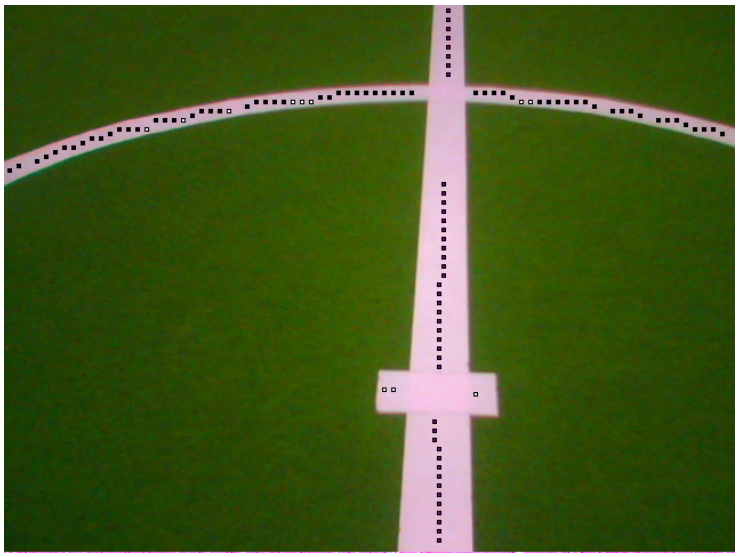

Figure 3.5: Result of Field Line Detection.
The points on the center-circle on the field denote the detected field lines.

- **Sonar Data and Sonar Filter** - Since the Robot Detection algorithm is meant to be a multi-modal algorithm and beside the multiple evidence taken from the visual frame, it is also necessary to incorporate the sonar information as an additional evidence to the final decision making. The left and right sonar sensors on the robot provide a simple way to tell whether obstacles are on the right, middle, or left of the robot. It is also essential to filter the raw sonar information in order to be impervious to noise and spikes in the signal. A more detailed explanation of this particular module will be available in the subsequent chapters.

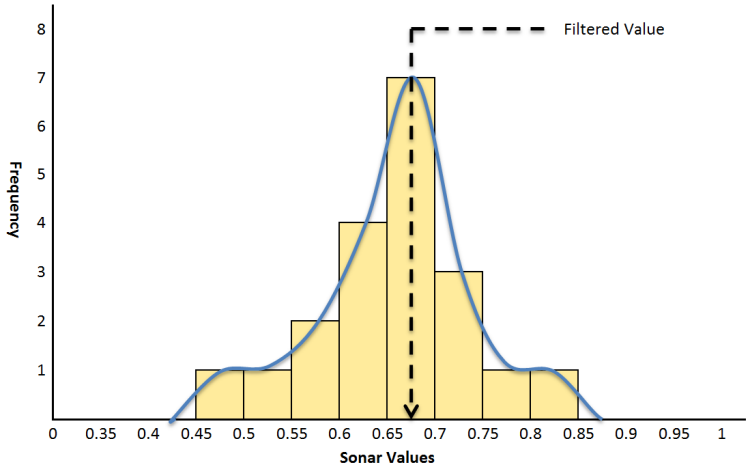

Figure 3.6: A Sonar Histogram
that demonstrates how the filtered value is chosen, that is, the first maximum point.

- **Machine Learned Decision Tree** - The machine-learned decision tree is the final line of sanity checks before candidates are considered to be true results that can be used in localisation and behaviours. The decision tree is learned using several sets of true and false data instances, manually taken from the robot. A more detailed explanation of this particular module will be available in the subsequent chapters.

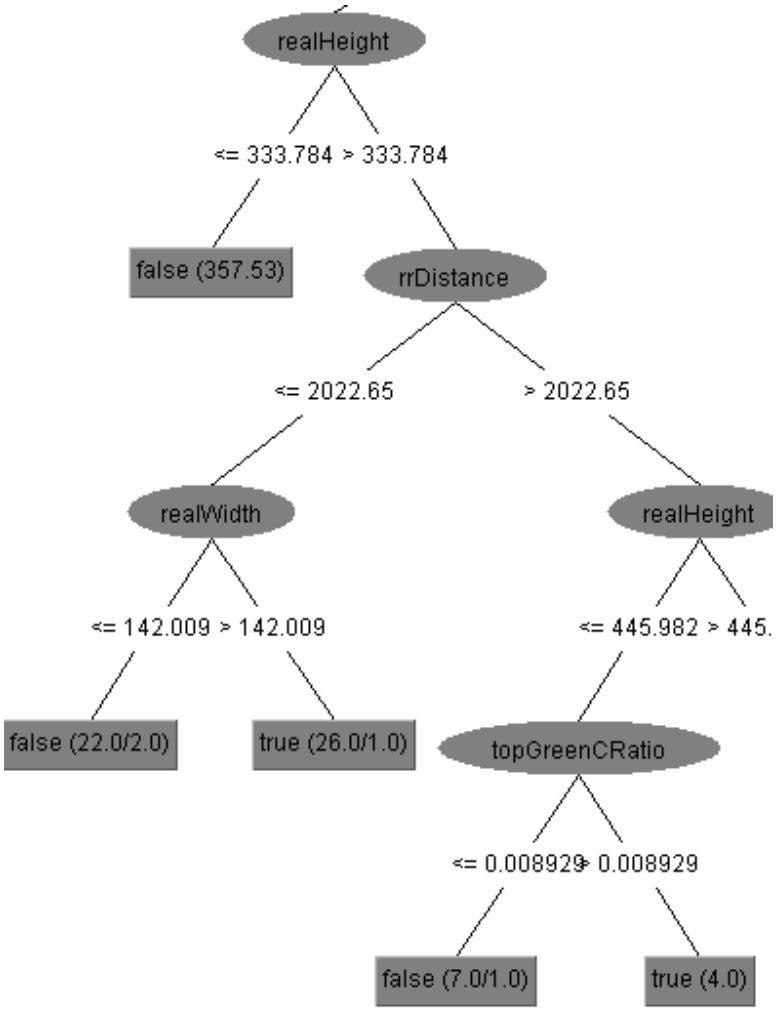

Figure 3.7: A Snippet of the Decision Tree generated by WEKA using the J48 classifier.

3.4 Candidate Robot Points Generation

3.4.1 Robot Points Search

The first step in the execution process of Robot Detection is the generation of candidate robot points. In order to improve performance, the candidate points generated by the field edge detection are recycled and used as the starting points to conduct the scanline search. Similarly, the candidate points generated by field line detection are also recycled and used as the ending points, since it is undesirable to consider the field lines as part of the robot.

Figure 3.8: The generation of candidate robot points that shows the first execution step of the algorithm. The green points are the recycled field edge points, red line is the edge line and the red points are the candidate robot points generated.

Referring to the figure above, the search process is a vertical downward search from the green points (recycled field edge points) until the first green pixel. The various conditions used while carrying out the search include:

- If the field edge points are above the field edge line, then discard the points and start the search from the field edge line.
- If the field edge points are below the edge line, then attempt to search for the first green pixel that has more than 1 neighbouring green pixel. This design decision will be explained later.
- If the green pixel that satisfies the above condition is found or the bottom end of the image is reached, then mark the point as a possible candidate.
- If the field line points are reached before reaching the green pixels, then terminate the search and mark the field line points as the candidate instead. Continue the search with the next column.

Completing these simple searches will generate the result shown in the figure above. Additionally, by linking all the candidate points together we will get an indentation from the field edge as previously mentioned in the concept section. The following figure is the indentation:

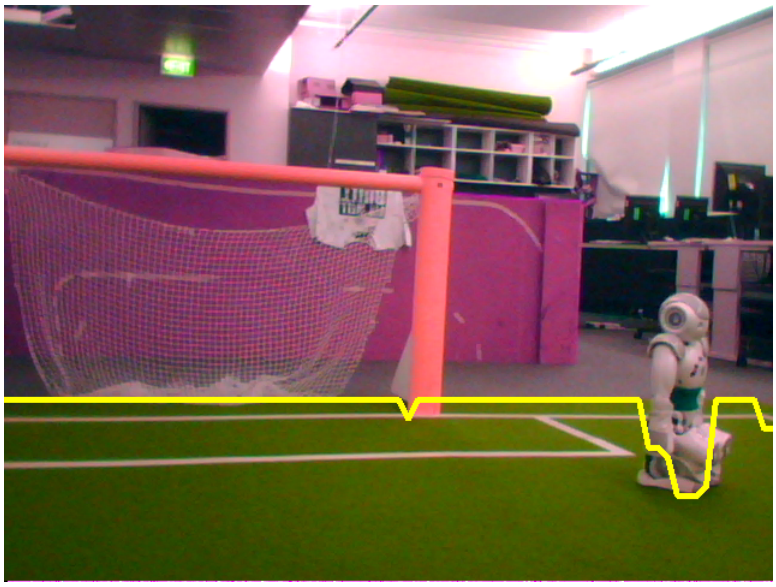

Figure 3.9: The generated indentation after the first search process shows the indentation created from the field edge. The indentation is similar to how it was visualised in the concept section.

3.4.2 Stage 1 Sanity Checks

The second step required to generate a proper set of candidate robot points is the need to run several low-level sanity checks that work on the pixel level in order to remove some unwanted indentations. During the search process in the first step, the following sanity checks or conditions have already been used:

- **Minimum number of neighbouring green pixels requirement**. This is to avoid the random appearance of a single speck of green pixel in the middle of the robot, which is common due to the nature of the vision system that relies on colour calibration. The count function can either use a 4-connected or 8-connected neighbours to find the number of green pixels. The figure below will demonstrate one such scenario:

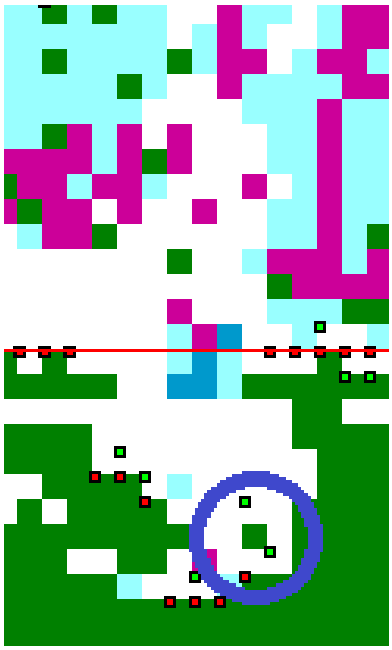

Figure 3.10: Random appearance of green pixels in robot that shows how it can happen. The image is the same from above, but cropped to only show the relevant area. The blue circle denotes the area of interest.

- **Remove points too close to the field edges**. The idea is that, such points either represents a false positive, which is highly likely, or a robot that is standing far away from the viewing robot (Typically over 3 meters away). Regardless, either scenarios are discarded and will not be considered as a possible robot region.

- **Remove points near goal posts**. The goal posts are not robots, hence the lack of desire to consider points near goal posts as candidate robot points. This sanity check is crucial, particularly with the blue goal posts which contain the same colour as the blue waistbands. If indentations are created near such goal posts, then false positives might be generated by the algorithm. Thus removing the points in the first place is a good defence to take.

During the search process and in conjunction with the sanity checks, a boolean array is used to keep track of whether a particular point is eligible. This boolean array will be used in the later stages when attempting to bound points together into regions. The values in the boolean array set are solely based on the sanity checks defined above.

These sanity checks are important. The figure below shows the result of not enabling any of the stage 1 sanity checks deployed.

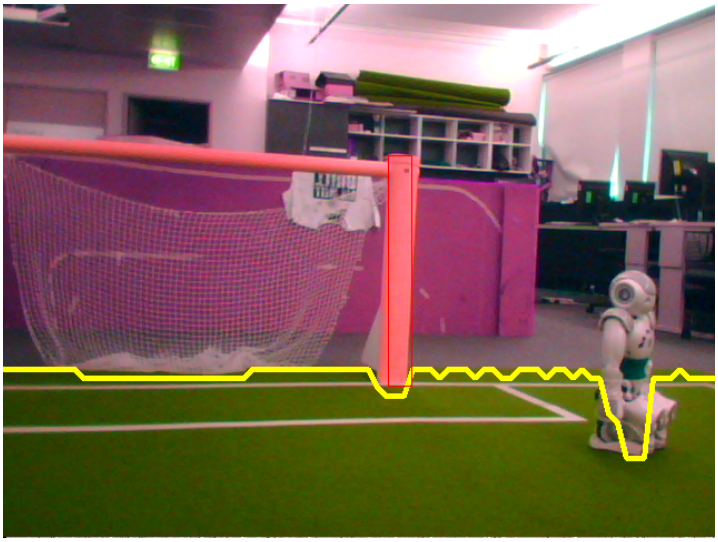

Figure 3.11: A visual frame with all of stage 1 sanity checks disabled that shows what the same yellow indentation line looks like without any of the sanity checks. Notice how the line is noisy and create a number of unnecessary indentations that might generate false positives.

Therefore, the end result of completing the first two steps of the Robot Detection algorithm is shown in the figure below. Unfortunately, despite the best effort to remove false candidates, it still exists. The regions generated after this stage will be subjected to more sanity checks to be completely confident of the result.

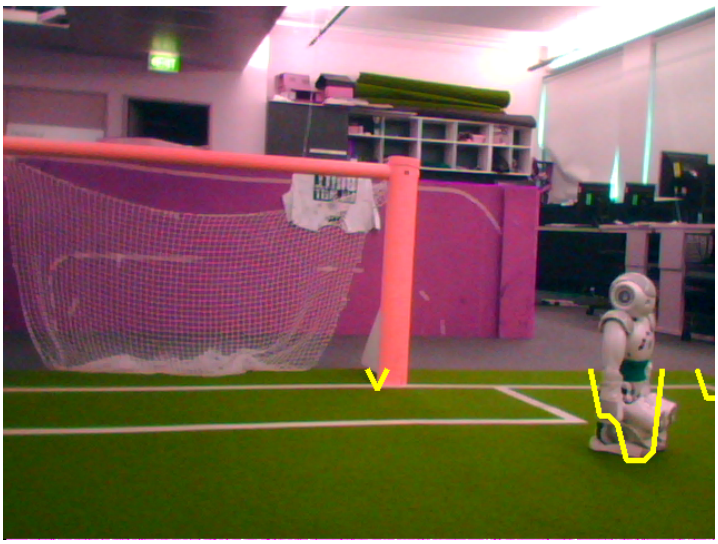

Figure 3.12: The end result after first two steps of the algorithm. Despite its best effort, there are still unwanted indentations in the visual frame.

3.5 Bounding of Points Into Boxes

3.5.1 Boxes from Points

The next step of the execution of the Robot Detection Algorithm is to generate regions based on the points generated in the previous stage.

The grouping of points together is a simple matter of iterating through the boolean array generated from the first stage. The bounding function attempts to find the first occurrence of a true point, mark this index and continue iterating through the array until the next false point is detected. Subsequently, given a starting point and an ending point, a region can be created. This process is repeated until the end of the array.

The figure below demonstrate the result of the bounding of points. The bounding boxes are initially bounded to the field edge lines. This will later be extended to the full body of the robot when attempting to build up the evidence set.

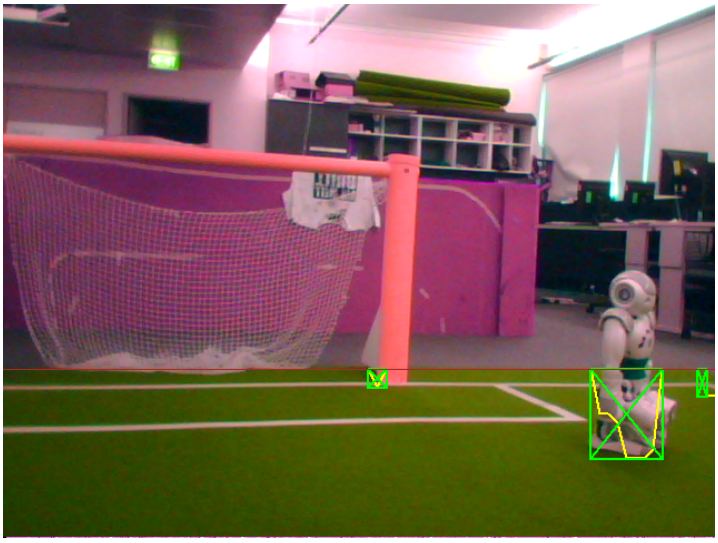

Figure 3.13: Bounding of points into boxes that shows how a region is created based on the candidate points generated from the first stage.

3.5.2 Stage 2 Sanity Checks

The purpose of stage 2 sanity checks is basically two-fold. First is as a fast and simple method to remove unnecessary regions and second is to improve performance by entirely skipping the evidence building stage for false candidates.

There are only 2 sanity checks in this second stage. The first sanity check is to remove bounding boxes that have unsuitable proportion. Basically, this is to remove boxes that have the following characteristics:

- Too small in width, such as 2 pixels wide or less.
- Too big in width, such as spanning the entire horizontal space of the image.

Heights or width-to-height ratio are not considered at this stage as there are not enough information to be able to positively tell whether a region is a true region or a false region. The implementation of this sanity check should be robust as more conditions should be able to be added in later time if found to be needed.

The second sanity check is to merge separated regions together. It is common to see each leg of the robot being detected individually as the waist of the robot is above the field edge line. Thus, it is necessary to merge these regions back together in order to be able to correctly generate an evidence set in the later stages. The following figures are an example of the merging of regions.

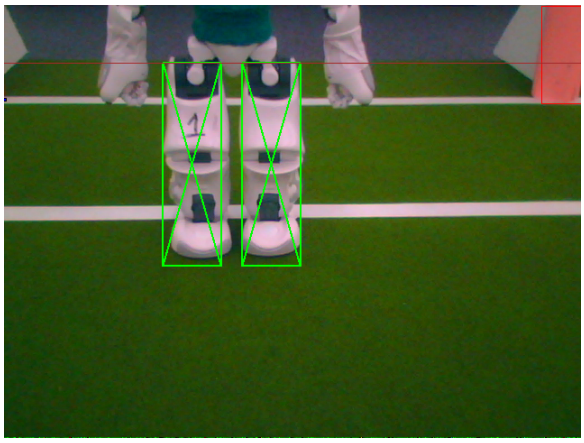

Figure 3.14: Regions prior to merging that shows the individual leg of the robot being bounded separately.

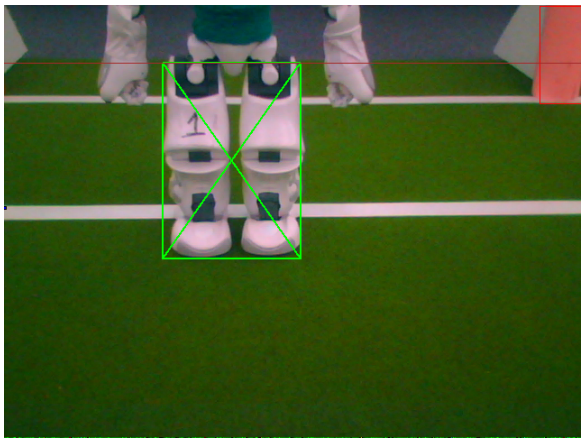

Figure 3.15: Regions after merging that shows both regions being merged together as one as it should be.

The method used to check whether regions should be merged are based on the following:

- For each region, find the middle point on the most bottom row of the bounding box. This is meant to represent the base of the foot.

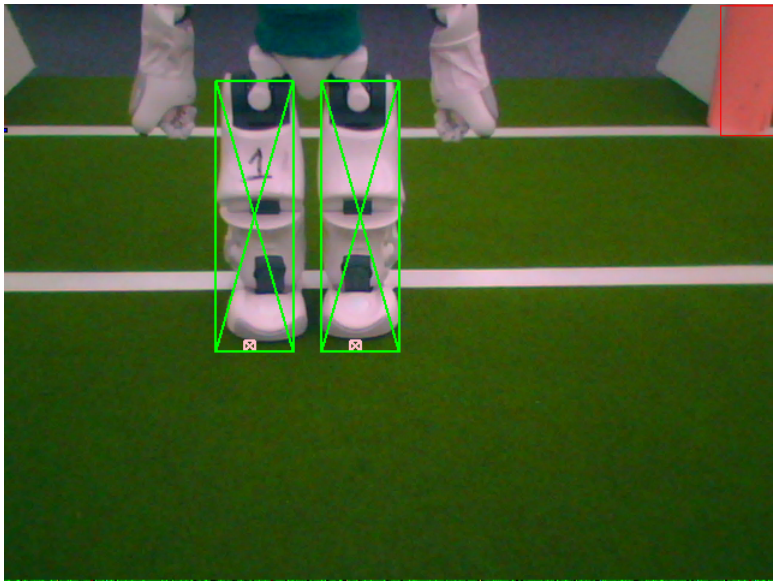

Figure 3.16: Middle point of the most bottom row of bounding box that shows the point used for coordinate translation. The small red box near the bottom of the bounding box represents the base of the foot.

- Use the camera matrix generated from the kinematics chain to find the relative distance and heading to the specified point.
- The relative coordinate values can be converted to absolute coordinate first, but it is entirely possible to do so using simple trigonometry for the next point.
- Attempt to match the regions by calculating the straight line distance from one robot relative point of one region to robot relative point of another region. As mentioned before, this can be done directly using relative coordinates.
- If the straight line distance is found to be less than the threshold then merge the regions together, discard the regions and add the new merged region to the candidate list.

The reasoning behind using robot relative coordinate as opposed to direct pixels distance is due to the fact that, it more precisely and more accurately reflect the distances in real life as opposed to pixel distance. Pixel distance of 1 pixel can mean different things depending on how close or how far away the object is.

3.6 Building Evidence or Clue Set

The second last step in the execution of the algorithm is building up the evidence set before running the decision tree. Similar to the parameters passed to the machine learner in learning the decision tree, the following is the evidences available for use:

- The base of the foot. This is essentially the middle point of the most bottom row of the bounding box.
- Height and width estimation, plus correction.
- Colour histograms and colour ratios.
- Waistband location.
- Sonar information
- Relative distance.

3.6.1 Height and Width Estimation, plus Correction

The height can be estimated solely by using the height of the robot and the relative distance to such robots. While robots may be standing at different height, a rough middle value can be used. This is not a concern as height correction will be carried out in the next stage. Afterward, the angle can be calculated using simple trigonometry and then multiplied by angle-to-pixels ratio to find the top most pixel of the height. The following figure demonstrate how the angle can be calculated.

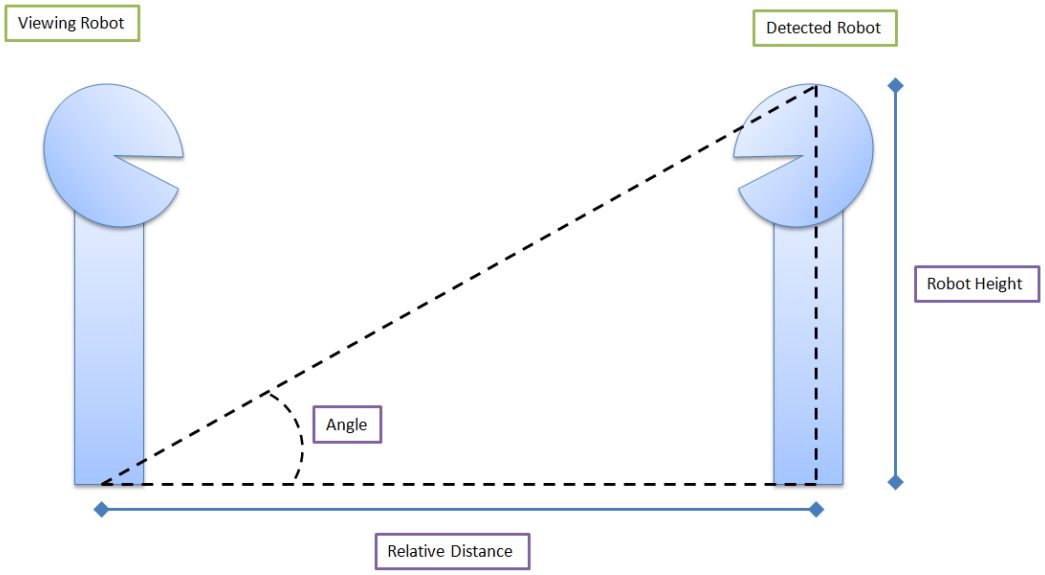

Figure 3.17: How height is estimated in the process of building evidence.

The aforementioned value of angle-to-pixels ratio are chosen through trial and errors from a range of different values, which are then evaluated and values that correspond to the most correct output will be selected. One example of this value would be for every degree in the angle, it represents 15 pixels in height.

The following figure is the result of height estimation.

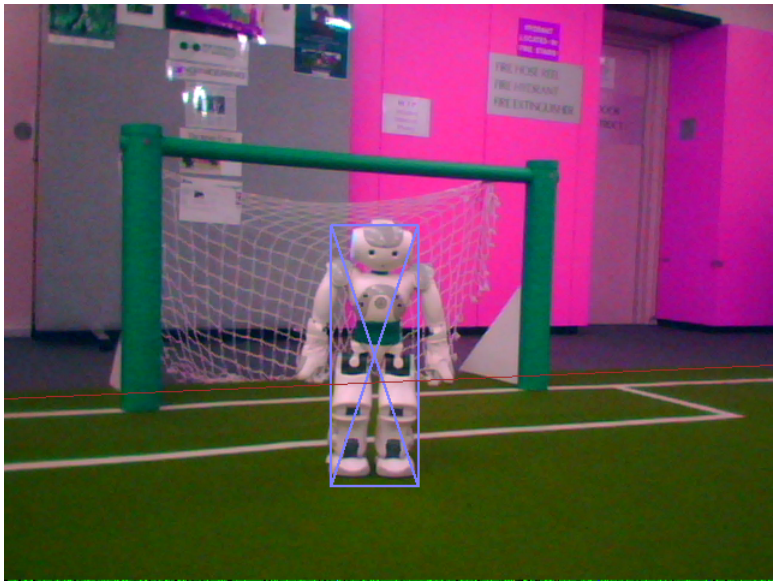

Figure 3.18: Correct height estimation

The following figure is another result of height estimation. Notice that the height is incorrect as it goes over the top of the robot. This will be corrected in the next step where height correction is carried out.

Figure 3.19: Incorrect height estimation where the estimated height is well above the robot.

As seen in the figure above, height estimation can be incorrect, thus it becomes necessary to carry out height correction. Height correction is done by the simple process of scanning downwards from

the estimated point until the first white pixel is seen. The reasoning behind this design is that, typically, background colours are not white and thus the first white pixel seen from the scanning process will be the top of the robot's head. Obviously, in the scenario where the background colours are white height correction will not be useful.

The following figure is the result of carrying out height correction from the previously incorrectly estimated figure.

Figure 3.20: Height estimation correction demonstrates how an incorrectly estimated height can be corrected.

The width of the robot can be found by using a similar mechanism as the region merging formulation. The process is basically as follow:

- Find the starting and ending point of the most bottom row of the bounding box.
- Convert these points into robot relative coordinate using the same camera transformation matrix.
- Calculate the straight line distance between the two robot relative coordinates.

The figure below demonstrates the 2 points used when attempting to calculate the straight line distance, in other words, the width of the robot.

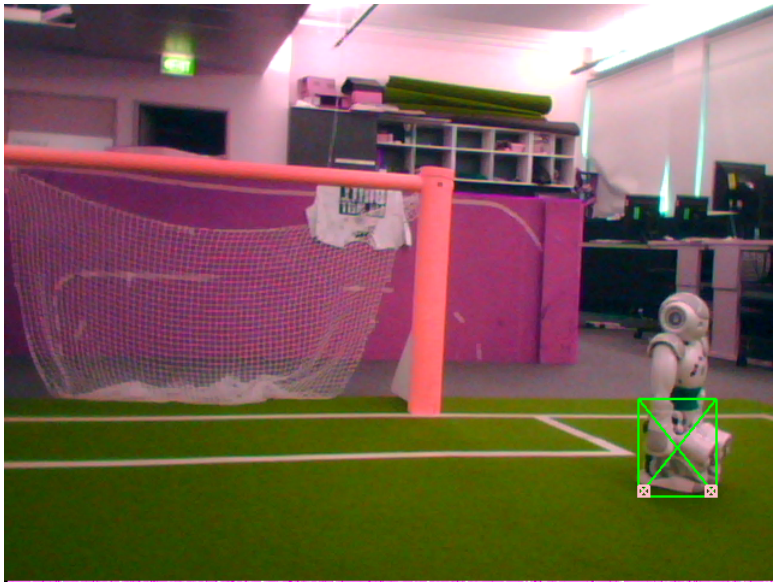

Figure 3.21: Width estimation that demonstrates the two points used when attempting to calculate the straight line distance, in other words, the width of the robot.

3.6.2 Colour Histograms and Colour Ratios

After both height and width estimations are completed, the bounding box is modified to incorporate these information and is now properly bounding the full body of the robot. Given this full body bounding box, building the colour histograms and calculating the colour ratios can be carried out.

Building the colour histograms are simple, as it basically involves iterating through every pixels in the bounding box and increment the corresponding colour in the histograms.

Figure 3.22: The different coloured pixels inside the blue bounding box that shows what the data looks like when attempting to build the colour histogram.

For this stage, the bounding box can be split into multiple sections and the colour histograms and ratios can be calculated separately. This is particularly useful when attempting to separate actual robots and false positives that contain groups of white at the top half of the bounding box (such as white backgrounds).

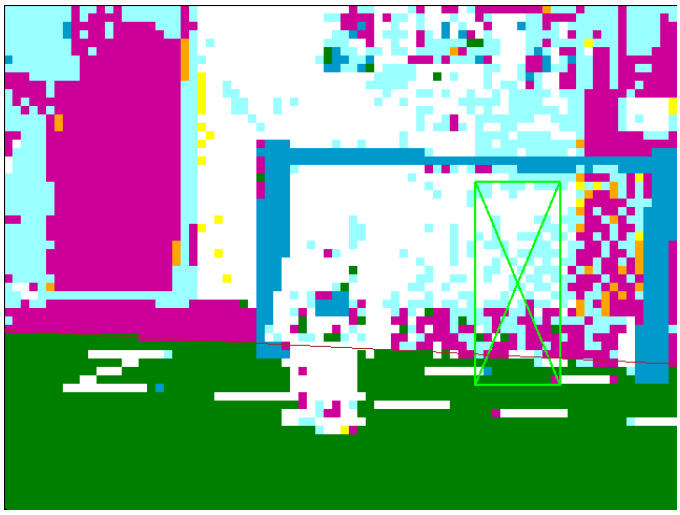

Figure 3.23: Splitting the green bounding box can be useful in ways where false positives may contain the correct amount of white pixels but are distributed unevenly. Notice how most of the white pixels are on the top half of the bounding box.

Finally, when the colour histograms have been completed, the colour ratios can be calculated. A colour ratio is basically how much space a particular colour takes up in a bounding box. Thus, to calculate the colour ratio, it can be done simply by dividing the count of a colour (from the colour histogram) by the total number of pixels in the bounding box.

3.6.3 Finding The Waistband

The waistband of the robot can be found in 2 ways:

- Conduct a scan from the middle bottom point to top of the bounding box. The goal is to find the occurrence of the first waistband colour, that is red or blue, and subsequently find the next non-waistband colour. This establishes the starting and ending point of the waistband in the vertical space.
- A similar technique to height estimation can also be used to find the waistband. Instead of specifying the full body height, the height up to the waistband can be used in this instance. Different robots tend to stand at different knee height, thus the estimation may be incorrect. However the complement of the technique can instead be used. This is where, rather than estimating the position of waistband from the foot, the estimation can be done from the top of the robot's head.

Generally the first method is more robust and would be able to give a more accurate result while still maintaining a fast performance. The second method relies heavily on the height estimation and correction being correct, otherwise the results will be highly inaccurate.

When the position of the waistband is established, a bounding box can be created and expanded to include the whole waistband. This expansion is done using the following steps:

- Receive the middle point or the starting and ending vertical point of the waistband.
- Using the middle point, an outward scan can be conducted until the bounding box covers the entire waistband.
- Using the starting and ending vertical points, a horizontal scan, to the left and right of the point, from the bottom of the waistband to the top can be conducted until the bounding box covers the entire waistband.

The following is the result of waistband detection and waistband bounding box expansion.

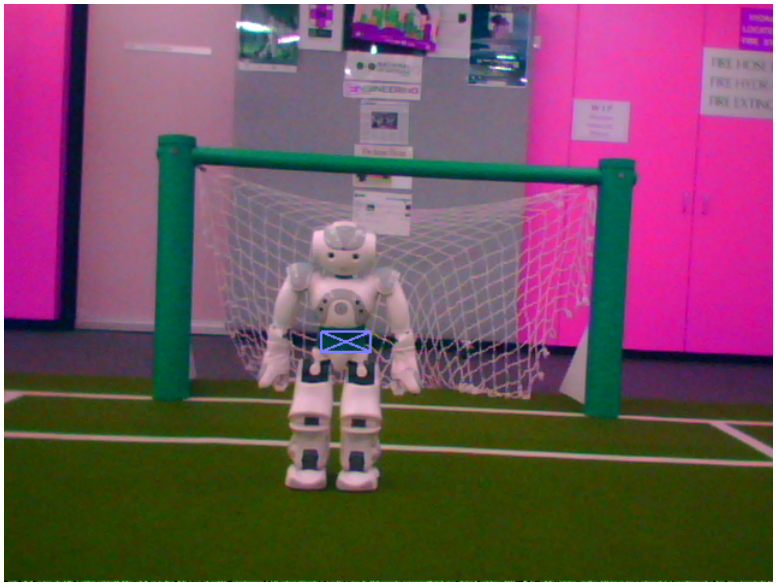

Figure 3.24: Waistband detection and expansion that shows the blue bounding box covering the entire waistband.

3.6.4 Using Sonar Information

The sonar data is crucial in Robot Detection as it is common for robots to get close to each other. When robots are in close proximity of each other, the vision system is unable to detect the robots as the full height of the robot can not be seen, hence none of the above evidence set can be built correctly. Thus the sonar information become an important clue in this scenario.

Using the sonar information requires the implementation of a sonar filter, which will be detailed in the next chapter. 2 instances of the sonar filter is used for Robot Detection, one to represent the left sonar and the other for the right sonar.

Robots detected by sonar take higher precedence than the robots detected by vision as the sonar sensors are relatively accurate at distance of less than 1 meter. Obviously, it is also possible for the sonar sensor to generate false data, hence the need for a filter, as well as the inclusion of many sanity checks. The usage of sonar information is basically as follow:

- Consider both sonar and vision information when the robot's head are facing forward, such that the results from both modules can be used to verify each other. 2 verifications can be done, that is, by comparing the relative distance and heading to the detected objects. If both values greatly differ, then the result can be discarded.
- Directly use sonar information when the robot's head are not facing forward, such as looking over the shoulders or downwards to the feet.

3.6.5 Calculation of Robot Relative Coordinates

The second last step in the Robot Detection algorithm is to calculate the robot relative coordinates by using the point on the base of the foot. As mentioned previously and used in the other steps, the translation can be done using the camera transformation matrix generated by the kinematics chain.

3.7 Running through Machine Learned Decision Tree

The final step of Robot Detection is to gather up all the evidence built up in the previous section and run the parameters through the decision tree. Using the decision tree requires the tree to be trained in the first place. The process of conducting such actions are detailed in the next chapter.

The following is a snippet of the decision tree in valid C code:

A snippet of the decision tree in valid C code

```
//Using colour ratio
if (topRobotCRatio <= 0) {
  if (botRobotCRatio <= 0) {
    //Using Estimated Height
    if (realHeight <= 333.784) isRobot = false;
    if (realHeight > 333.784) {
      if (botOtherCRatio <= 0.194444) {
        //Using estimated width
        if (realWidth <= 144.726) isRobot = false;
        if (realWidth > 144.726) isRobot = true;
      } else if (botOtherCRatio > 0.194444) isRobot = false;
    }
  } else if (botRobotCRatio > 0) {
    if (botWhiteCRatio <= 0.3) isRobot = false;
    if (botWhiteCRatio > 0.3) {
      if (realHeight <= 575.327) isRobot = true;
      if (realHeight > 575.327) isRobot = false;
    }
  }
  .
  .
  .
```

The decision tree allows a binary classification of the region based on the given evidence set and the result is directly used (the isRobot variable) to indicate whether or not a particular region should be included in the result set.

3.8 Results and Discussions

The following is the parameters used in the current implementation:

The different parameters used in Robot Detection

```
ROBOT_FULL_HEIGHT           565   //full robot height
ROBOT_TOP_WAISTBAND_TO_HEAD 205   //waistband to head height
ROBOT_BASE_TO_WAISTBAND     245   //foot to waistband height
RD_DEGREE_TO_PIXEL          14.5  //degree-to-pixel ratio
RD_MAX_SONAR_FILTER_SIZE    20    //sonar window size
RD_SONAR_ACCEPT_COUNT       8     //minimum sonar histogram
   value for usage

//Verifications use between sonar and vision robots
SC_MAX_SONAR_DISTANCE       800
SC_MAX_SONAR_HEADING        DEG2RAD(15)

//Maximum detect-able robots
SC_MAX_DETECT_DISTANCE      3000
//Smallest possible robot height
SC_MIN_ROBOT_HEIGHT         140
//Distance from the edge line to be ignored when generating
   points
SC_SCAN_IGNORE_DISTANCE     2
//Minimum number of neighbouring green pixels when generating
   points
SC_MIN_GREEN_ACCEPT         3
//Minimum number of waistband colour when finding the
   waistband
SC_WAIST_BAND_ACCEPTABLE    2
//Minimum size threshold in stage 1 sanity checks
SC_MIN_PIXELS_WIDTH         3
SC_MAX_PIXELS_WIDTH         SALIENCY_COLS - 2
//The straight line distance threshold used for merging
   regions
SC_SAME_ROBOT_WIDTH         (230 * 1.1)

//Sonar: The head cameras pitch and yaw when determining
   whether
//the robot's head is facing forward
TOP_CAMERA_MIN_PITCH          -0.35
TOP_CAMERA_MAX_PITCH           1
BOT_CAMERA_MIN_PITCH          -1
BOT_CAMERA_MAX_PITCH          -0.15
MIN_YAW                       -0.75
MAX_YAW                        0.75
```

The robot detection algorithm generally has the greatest success when a full body bounding box can be established as it allows the building of the entire evidence set. Partially bounded robot may also be classified as a robot, however, it is often discarded as there are not enough information to make an informed decision.

The following figures demonstrate where the robot detection algorithm is a success.

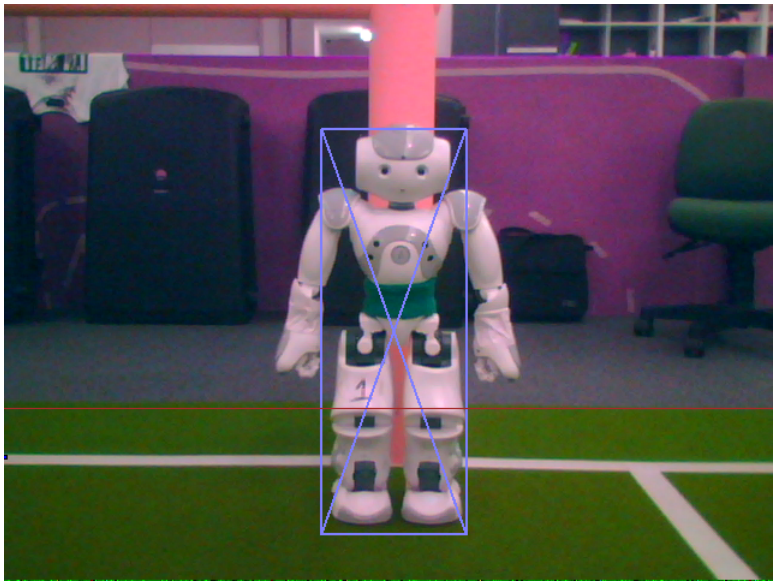

Figure 3.25: Successful robot detection No.1 that shows a robot being detected while standing in front of the goal post.

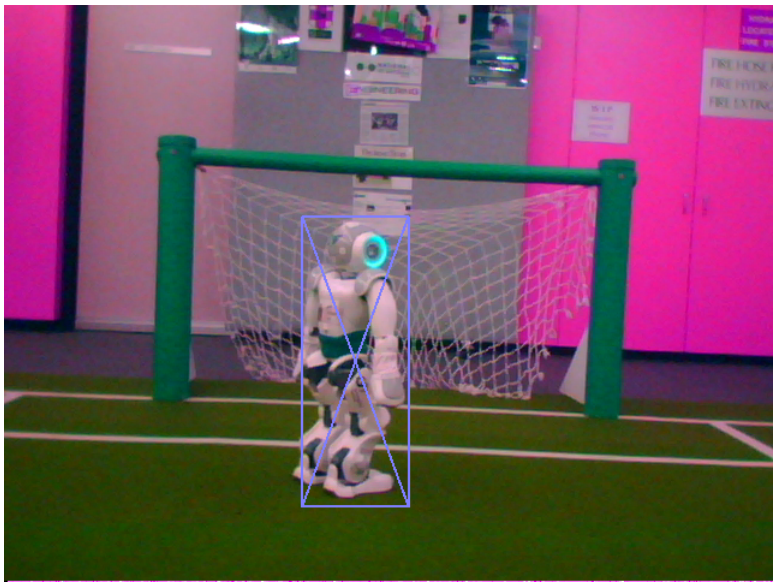

Figure 3.26: Successful robot detection No.2 that shows a robot being detected from its back.

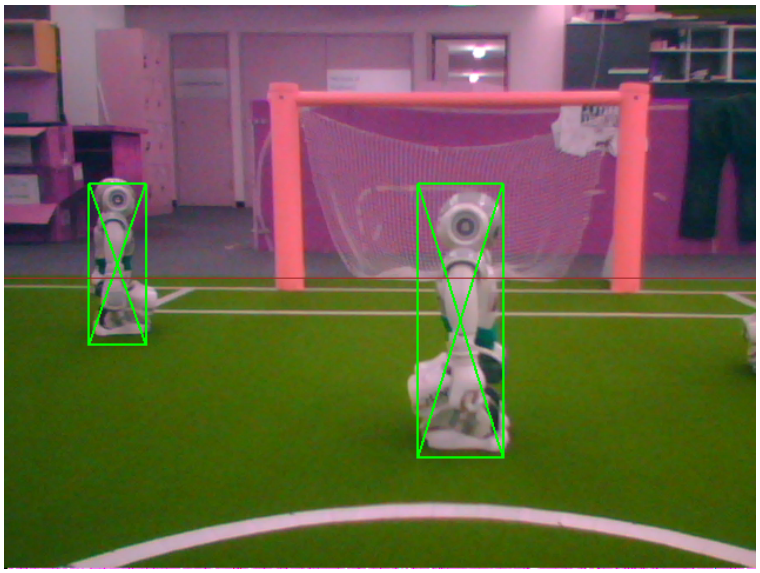

Figure 3.27: Successful robot detection No.3 that shows two robots being detected despite the waistbands are obstructed.

Figure 3.28: Successful robot detection No.4 that shows all three robots correctly identified with the correct team colour. The green box denotes robots that do not have the waistband.

The following figure is where robots are detected, but not perfect.

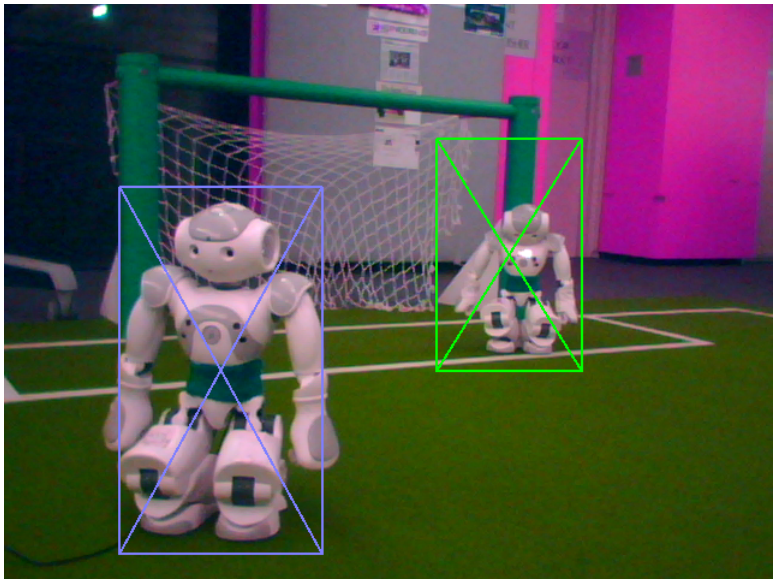

Figure 3.29: Partially successful robot detection that shows the robot on the left being correctly detected, but the robot on the right, while detected, is not detected perfectly.

The robot detection also has a number of problems, these include:

- Incorrectly identifying the white triangle behind the goal posts as robot.
- Unnecessarily lengthen the bounding box when robots are standing on the field lines when the field lines are not detected.
- Incorrect height estimation with white background.
- The arms of the robot being detected as separate robots.
- Unable to merge the regions around the arms of the robots into the main body region.

The following figures will demonstrate the problems outlined above.

Figure 3.30: The white triangle behind goal posts being incorrectly detected.

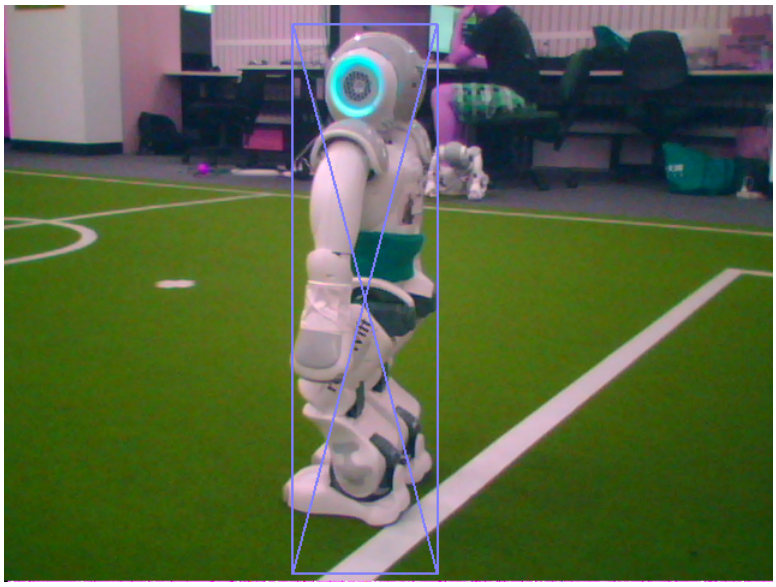

Figure 3.31: Lengthening of bounding box when field lines are not detected.

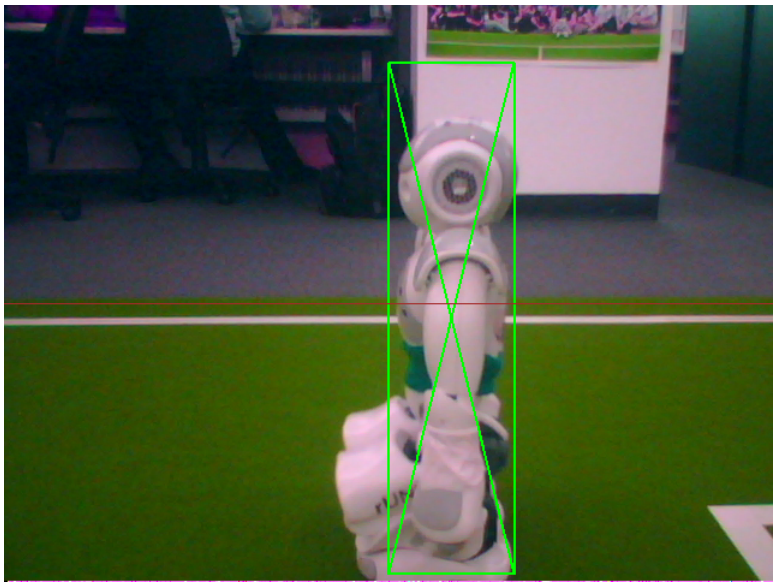

Figure 3.32: Incorrect height estimation when the background is white.

Figure 3.33: Mis-detection when the background colour is white in raw image.

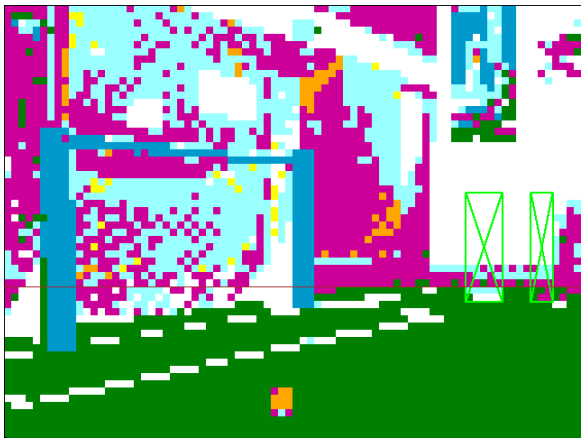

Figure 3.34: Mis-detection when the background colour is white in saliency scan.

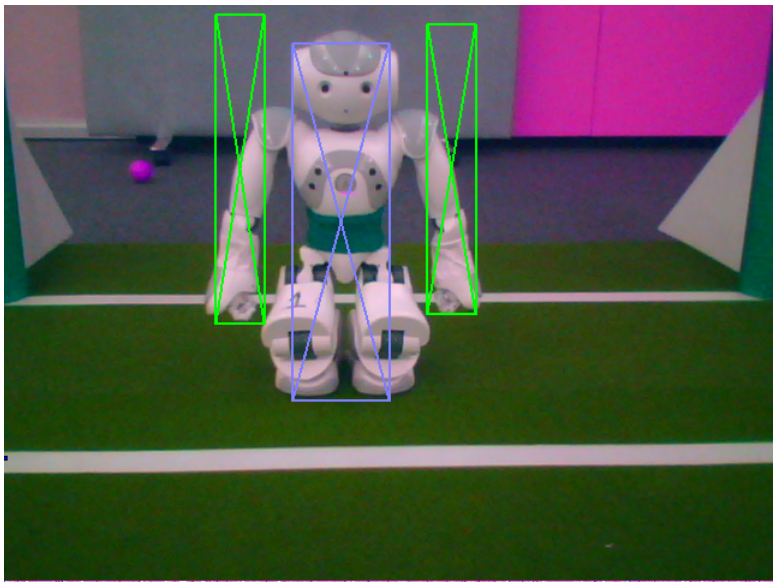

Figure 3.35: Mis-detection on robot arms and merging failure that shows the arms of the robot being detected individually and unable to be merged into the main body region.

In the current implementation, three out of four of these problems are actually solved. However, the fix added to solve the problems are inelegant. All of these fixes are hard-coded restriction on the results. Thus, there is a possibility that these fixes may discard actual true result.

The following is the list on how the hard-coded fixes are implemented:

- Solving the white triangle problem is done in 2 steps. The first step is by removing any candidates that are found to be behind the goal posts. While the second step is to directly remove any candidates that are in close proximity of the goal posts. The threshold used in the second step is chosen to a value that is smaller than the circular cross-section of the robot. Thus it is impossible for the robots to be in a position near the goal posts where it would not be detected.
- Solving the lengthening of bounding box is done by scanning the surrounding neighbours of the robot points inside the bounding box. Typically, the lengthening is caused by a small number of spikes, thus, by scanning for neighbours, the spikes can be detected and removed.
- Solving the mis-detection on the robot arms are done by checking for neighbouring regions. If the neighbouring regions are close, such as within several pixels, and contain waistband information (that is, the waistband is detected), then the arm regions are directly discarded.

In the future, as more sophisticated tools and functionality are implemented, hopefully these hard-coded sanity checks can be removed. Solving the problems outlined above will just become one of the many steps in providing accurate evidence set.

3.9 Conclusions

In conclusion, the algorithm used by robot detection is multi-modal and make use of the machine learned decision tree as the last line of defence in sanity checks. The decision tree requires the build up of evidence set, thus many of the functionality implemented are specific to providing accurate evidence set while discarding any bad regions.

The detection has the highest success when a full body bounding box is available. Partially bounded body can still be detected, but is typically discarded due to the lack of enough evidences. Several edge cases that cause incorrect objects to be detected also exist in the algorithm, but are fixed using several hard-coded sanity checks. These hard-coded sanity checks may be too strict with the possibility of removing true candidates as well.

Chapter 4

Machine Learning the Decision Tree

4.1 Introduction

This chapter will outline the design decision in implementing the decision tree, the usage, usefulness and the result of using the decision tree in the robot detection algorithm. This module fits in together in the software architecture as the final sanity checked carried out by the Robot Detection algorithm before exposing the results to other modules.

4.2 Motivation

The decision tree is machine-learned due to the existence of many parameters when building up the evidence set in the Robot Detection algorithm. It becomes highly undesirable and difficult to maintain and determine the effect and usefulness of each parameters. Each parameter also tend to have several threshold values. Finding and evaluating the threshold values manually are undesirable. Thus it was decided that using a decision tree classifier would be much simpler and easier to maintain the process of acquiring a working decision tree.

4.3 Software Used

The software used for machine learning the decision tree is called WEKA [10]. The software is a Java [7] program that offers a Java implementation of the C4.5 [11] decision tree classifier called J48 [10].

4.4 Parameters

The following table lists all the parameters that are generated by the robot detection and are available to be used for machine learning.

Parameters	**Variable Name**	**Type**
Height in millimeters	realHeight	REAL
Width in millimetres	realWidth	REAL
Robot Relative Distance	rrDistance	REAL
Is the Full Height of Robot Visible?	isHeightClipped	BOOLEAN
Filtered Value of Left Sonar Sensor	leftSonar	REAL
Filtered Value of Right Sonar Sensor	rightSonar	REAL
Detected Waistband Location Relative to the Bounding Box Value 0 for bottom, 1 for middle and 2 for top	waistBandLoc	{0,1,2}
Bottom Section of Bounding Box Colour Ratio		
Orange	botOrangeCRatio	REAL
Yellow	botYellowCRatio	REAL
Waistband Colour (Red and Blue)	botRobotCRatio	REAL
Green	botGreenCRatio	REAL
White	botWhiteCRatio	REAL
Black	botBlackCRatio	REAL
Other	botOtherCRatio	REAL
Top Section of Bounding Box Colour Ratio		
Orange	topOrangeCRatio	REAL
Yellow	topYellowCRatio	REAL
Waistband Colour (Red and Blue)	topRobotCRatio	REAL
Green	topGreenCRatio	REAL
White	topWhiteCRatio	REAL
Black	topBlackCRatio	REAL
Other	topOtherCRatio	REAL
Is the Region a Robot? (Hand classified)	isRobot	BOOLEAN
Basic Robot Pose Estimation (Hand classified)	robotPos	{front,back side}

It is common to dynamically enable and disable different parameters to test for different combinations of parameters and the different results attained from it.

The usefulness of any particular parameters are determined through analysing the data set for the distribution of the results. In other word, it is an attempt to look for correlation in the data set that could emerge as a pattern. This pattern can then be utilised to distinguish between true and false data. For example, having most of the true result reside in the lower ranges of values, while most of the false result would be on the higher range of values. Values in between are mixed and would usually require the presence of other parameters to be completely positive. The figure below will demonstrate an example of the pattern.

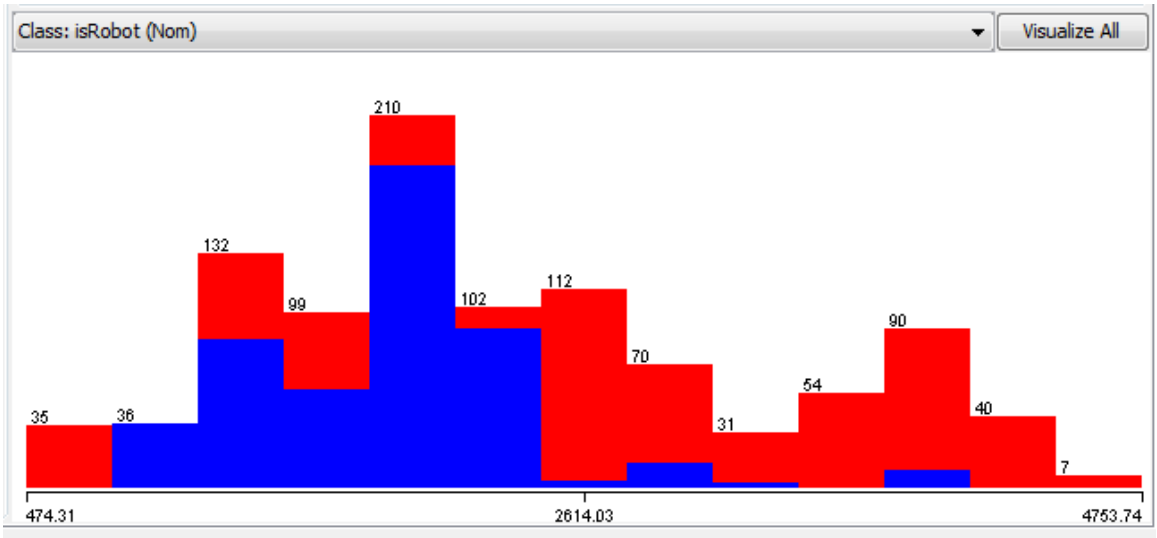

Figure 4.1: WEKA Visualiser: pattern in data et
that shows an example of how patterns emerge from the data set.
The blue bar represents the true data, which are mostly on the right, while the red bar represents the false data, which are mostly on the left.

4.5 Data Gathering and Classification

Generally when using a machine learning classifier, a large data set is required, in order to have enough sample data to train the classifier and test data to verify the policy learned. In the current implementation, the method of acquiring the data set can unfortunately be time consuming. The method used basically consist of processes such as:

- Taking a visual frame dump of the robot running around an empty field. If any candidates are generated, then all of them will be classified as false.
- Taking another visual dump with multiple robots on the field. The mixture of static and moving robots are highly desirable to partially simulate a real game. Afterward, for every candidates generated from the frame dumps, manually check, verify and classify them.

These particular processes are very time consuming and in the future, an unified tool to allow simpler gathering and classifying of the data and the running the classifier can be implemented.

4.6 Running Through Classifier

In order to be able to dynamically turn any particular parameter on or off before running the classifier, various information about a visual frame is first stored in a text file in its raw pre-calculated form. For example, storing the actual number of white pixels in the bounding box as opposed to directly storing the ratio of white pixel. However, storing raw information are not useful, as it lacks any association amongst different parameters. This is especially true to the classifier as it could not draw a valid conclusion with the given parameters. Thus, an intermediate program is developed to specifically take in these raw data and convert them into usable parameters outlined earlier.

The following texts below is an example of the raw storage format used for the intermediate program.

An Example of the Raw Storage Format Used

```
#True Robot No.1:
578.878 #realHeight
240.815 #realWidth
38,38,38,38,38,...,43,43,43,43 #fieldEdgeYCoordinates
0,0,0,0,0,0,71,0,0,49 #topColourRatio
120 #totalTopPixels
0,1,0,0,0,27,25,0,33,34 #bottomColourRatio
120 #totalBottomPixels
448,168 528,360 #boundingBoxPosition
488,360 488,360 #detectedWaistbandPosition
2460.07,-0.0584932 #robotRelativeDistanceAndHeading
no #isHeightClipped
true #isRobot
front #basicRobotPose
```

The WEKA software also required the input file to conform to its standard formatting. Therefore, the same intermediate program can be extended to include such functionality as well. The text below is an example of the WEKA input format. Please take note that the example is not meant to show the input file in its entirety as the actual input contains well over 1000 elements in the data set, thus roughly equating to about 2000 lines of data. The example is only meant to show the format used by WEKA.

An Example of the WEKA input format

```
% Robot detection data set
@RELATION RobotDetectionMachineLearning

@ATTRIBUTE realHeight            REAL
@ATTRIBUTE realWidth             REAL
.
.
.
@ATTRIBUTE isHeightClipped       {true, false}
@ATTRIBUTE isRobot               {true, false}

@DATA
41.8109,183.496,...,0.0,0.0,0,1446.93,true,false
42.8584,57.7365,...,0.0,0,1483.18,true,false
```

4.7 Using The Result

The output of using the J48 classifier in the WEKA software is in the form of a decision tree. The formatting of the tree is in a simple ASCII format that is easy to read and user-friendly. Unfortunately, this format is not useful for programming purpose. The decision tree needs to be converted into valid code to be useful. Thus, the same intermediate program mentioned previously is once again extended to include this functionality as well. This can be done through analysing the pattern of the output format and with several simple rules, the decision tree can be converted into valid C code.

Example of WEKA Decision Tree Output

```
J48 pruned tree
------------------

topRobotCRatio <= 0
|   botRobotCRatio <= 0
|   |   realHeight <= 333.784: false
|   |   realHeight > 333.784
|   |   |   botOtherCRatio <= 0.194444
|   |   |   |   realWidth <= 144.726: false
|   |   |   |   realWidth > 144.726: true
|   |   |   botOtherCRatio > 0.194444: false
|   botRobotCRatio > 0
|   |   botWhiteCRatio <= 0.3: false
|   |   botWhiteCRatio > 0.3
|   |   |   realHeight <= 575.327: true
|   |   |   realHeight > 575.327: false
topRobotCRatio > 0
|   botWhiteCRatio <= 0.5
|   |   topRobotCRatio <= 0.032051: false
|   |   topRobotCRatio > 0.032051
|   |   |   topOtherCRatio <= 0.364286: true
|   |   |   topOtherCRatio > 0.364286: false
|   botWhiteCRatio > 0.5
|   |   topWhiteCRatio <= 0.433333
|   |   |   botRobotCRatio <= 0.009615
|   |   |   |   realHeight <= 490.97: false
|   |   |   |   realHeight > 490.97: true
|   |   |   botRobotCRatio > 0.009615: false
|   |   topWhiteCRatio > 0.433333: true
```

Having this functionality can allow for quick testing and evaluation cycles as there is no need to manually convert the tree by hand, which can be highly time consuming, especially with 8 parameters or more, where there would be over 20 individual if/else if statements, each with different floating point values representing the thresholds.

Example of the Valid C Code Generated by The Intermediate Program From WEKA Decision Tree Output

```
if (topRobotCRatio <= 0) {
  if (botRobotCRatio <= 0) {
    if (realHeight <= 333.784) isRobot = false;
    if (realHeight > 333.784) {
      if (botOtherCRatio <= 0.194444) {
        if (realWidth <= 144.726) isRobot = false;
        if (realWidth > 144.726) isRobot = true;
      } else if (botOtherCRatio > 0.194444) isRobot = false;
    }
  } else if (botRobotCRatio > 0) {
    if (botWhiteCRatio <= 0.3) isRobot = false;
    if (botWhiteCRatio > 0.3) {
      if (realHeight <= 575.327) isRobot = true;
      if (realHeight > 575.327) isRobot = false;
    }
  }
} else if (topRobotCRatio > 0) {
  if (botWhiteCRatio <= 0.5) {
    if (topRobotCRatio <= 0.032051) isRobot = false;
    if (topRobotCRatio > 0.032051) {
      if (topOtherCRatio <= 0.364286) isRobot = true;
      if (topOtherCRatio > 0.364286) isRobot = false;
    }
  } else if (botWhiteCRatio > 0.5) {
    if (topWhiteCRatio <= 0.433333) {
      if (botRobotCRatio <= 0.009615) {
        if (realHeight <= 490.97) isRobot = false;
        if (realHeight > 490.97) isRobot = true;
      } else if (botRobotCRatio > 0.009615) isRobot = false;
    } else if (topWhiteCRatio > 0.433333) isRobot = true;
  }
}
```

Given the C code, the final step is to simply copy and paste the code into the Robot Detection algorithm. Since the parameter names and variable names are kept the same, there is no modification required to run the code.

4.8 Result and Evaluations

The decision tree generated above in the previous section is run with the following configuration into the WEKA learning suite:

WEKA J48 Classifer Configuration Parameters

```
#Default parameters value are used
Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     RobotDetectionMachineLearning
Instances:    1209
Attributes:   18
              realHeight
              realWidth
              botYellowCRatio
              botRobotCRatio
              botGreenCRatio
              botWhiteCRatio
              botBlackCRatio
              botOtherCRatio
              topYellowCRatio
              topRobotCRatio
              topGreenCRatio
              topWhiteCRatio
              topBlackCRatio
              topOtherCRatio
              waistbandLoc
              rrDistance
              isHeightClipped
              isRobot
Test mode:    10-fold cross-validation
```

The following is the result of running the classifier:

WEKA J48 Classifier Result

```
Time taken to build model: 0.17 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        1182       97.7667 %
Incorrectly Classified Instances        27        2.2333 %
Kappa statistic                          0.9535
Mean absolute error                      0.0295
Root mean squared error                  0.1476
Relative absolute error                  6.1754 %
Root relative squared error             30.1819 %
Total Number of Instances             1209

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
  0.985     0.027      0.959     0.985     0.972      0.982    true
  0.973     0.015      0.99      0.973     0.981      0.982    false
Weighted Avg.
  0.978     0.02       0.978     0.978     0.978      0.982

=== Confusion Matrix ===

   a   b   <-- classified as
 472   7 |   a = true
  20 710 |   b = false
```

As seen above, using the J48 classifier provides great benefits overall. The J48 classifier is able to achieve 97.7% accuracy with the 1209 instances in the given data set and able to learn a decision in a short amount of time. It is entirely possible to be constantly modifying the parameters passed into the J48 classifier to get 100% accuracy, however, it is often undesirable to do so since the decision tree will not attempt to be a general purpose decision tree, but instead it will intentionally fit itself into the data set, neglecting the fact that there may be unknown cases and new data sets.

Notice that sonar information is not incorporated into the learning process of the decision tree as it proves to be highly difficult to attain the correct sonar value without a proper unified tool. The sonar filter tends to exert lag and may gives an entirely incorrect value, hence why it was not used in the first place as one of the parameters to train the decision tree.

The use of machine learning in this scenario has greatly improved the overall performance, particularly, in both the accuracy of Robot Detection and the development time.

The development time has received the greatest impact overall with the usage of this module. There are no more guessing game when determining the threshold values for each parameters. It also removes the ambiguity and uncertainty of the effect of changing a single parameter threshold, since the classifier take into account of every parameters when training the decision tree.

This method, unfortunately, also has its flaws. There are a few edge cases where false results are generated. This is primarily caused by a pair of true and false candidates that exert similar parameters value. An example of this scenario is when an indention is created and the background colour above such indention happens to be white. As the major colour in the robot is white, it is understandable how such scenario could be mistaken. The 2.2333% of inaccurately classified instances can be attributed to this particular flaw.

The classifier is also vulnerable against data that are new or have not been included into the sample data. There could essentially be more edge cases that could cause other major problems.

The result of the classifier may also be highly sensitive to the colour calibration. A major change in the method of carrying out colour calibration will greatly affect the result. For example, a different calibration that produces much lesser white would throw off the result.

In future implementation, with the unified tool, one could instead store the raw pixels directly and apply the colour calibration on the fly in the intermediate program while generating a WEKA compliant input file. In fact, WEKA J48 classifier could be integrated into such tool. More detail about this future work is detailed in the future work chapter.

4.9 Conclusions

In conclusion, the machine learning is done through WEKA. All of the parameters are stored in raw form and later converted into the proper input format. The decision tree is subsequently converted to valid C code that can be used directly in the Robot Detection algorithm.

The use of machine learning to train the decision tree has greatly improved the performance and accuracy of Robot Detection and its greatest impact is improving development time. Overall, it greatly ease the process of constantly changing the parameters or adding new parameters to test for its effect and usefulness as a whole.

The shortcoming of the current approach can be solved in the future with better tools and more data set to train the decision tree.

Chapter 5

Sonar Filter

5.1 Introduction

This chapter will outline the motivation behind implementing a sonar filter, its usage, usefulness and the inner working of the algorithm itself as well as the evaluation on the result of using the filter. The sonar filter fits in together with the whole architecture as one of the many evidences built up by the robot detection algorithm to determine the presence of other robots in the viewing robot forward vicinity.

5.2 Motivation

The motivation behind using a sonar filter (or any filter in general) is to remove or minimise the impact of noise in the data set. The figure below demonstrate the existence of noise from the sonar sensor.

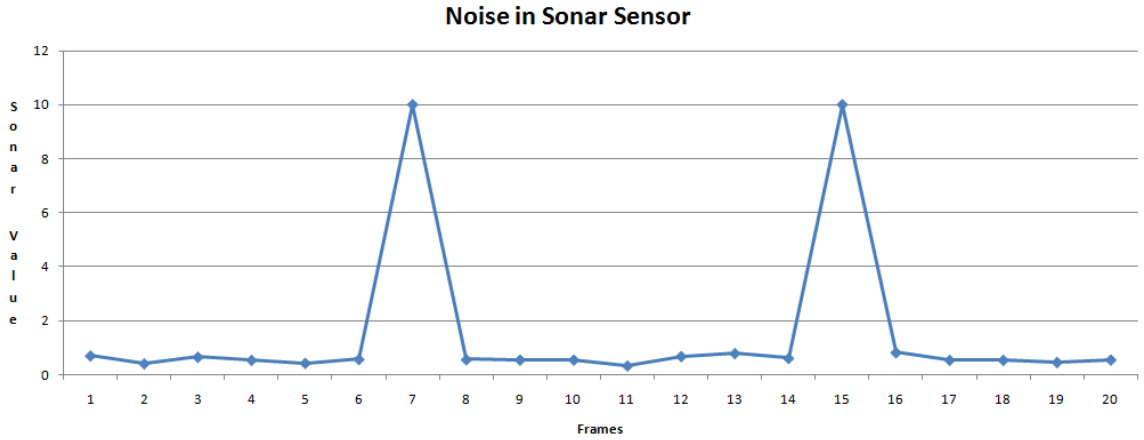

Figure 5.1: Noise in sonar sensor that demonstrates the presence of noise even if the objects being detected is static.

Using the sonar data in its raw form will yield inaccurate, error-prone and inconsistent result as a sudden jump in the input value will cause unexpected behaviour that may not be wanted, such as triggering a function that deals with system shut down.

Similarly, in RoboCup SPL, the sonar sensor is usually used for detecting close objects, that are not able to be detected by the vision system, such as other robots. When other robots are close, it is often desirable to halt the current running behaviour, avoid the robot and resume the previous behaviour. However, without filtering the sensor data, the robot may attempt to avoid nothing, which is high undesirable, particularly during critical moments during the game.

5.3 Algorithm

The basic concept, requirements and step by step process of the sonar filter algorithm is basically the following:

- The algorithm uses a window size of N previous sonar values being given by the sonar sensors. The window is a list storage that constantly add more data while removing the oldest data if the size N has been filled up. The value of N is arbitrarily chosen or determined depending on its usage, the data input rate and the processing power of the device.
- A higher value of N will give a more accurate result as there are more sample data that can be drawn upon. On the other hand, a lower value of N will allow the system to be more reactive as less sample data can lead to less process and perception lag. Generally there is a need to find the balance between the two creteria. The figure below will demonstrate the working of the window.

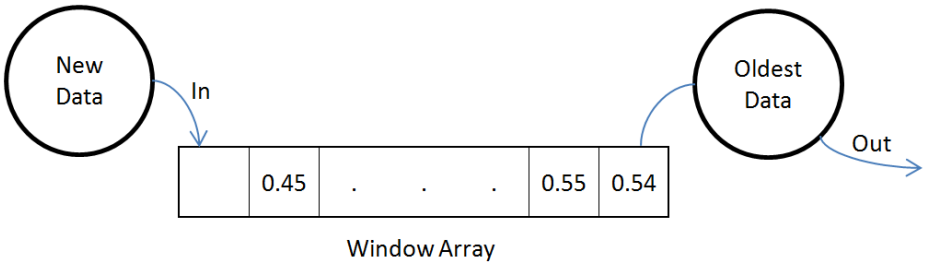

Figure 5.2: Sonar window example shows how the window mechanism works

- The sonar filter primarily utilise a histogram of input values from the window to determine the filtered value. The histogram is essentially a count array that increments or decrements a particular value range as more new data comes in and old data goes out. The bounds of the histogram are arbitrarily chosen depending on the usage of the sonar filter, value of input data and the maximum accurate range of the sonar.

- Since the input sonar data are in the form of real value, the histogram needs to be divided into several K parts of equal range size. The choice of the value of size K is again arbitrary. A smaller value of K will gives a broader filtered value (less accurate), but more reactive, while a higher value of K will give a narrower or tighter filtered value (more accurate), but may cause more perception lag. The figure below will demonstrate the working of the histogram of values.

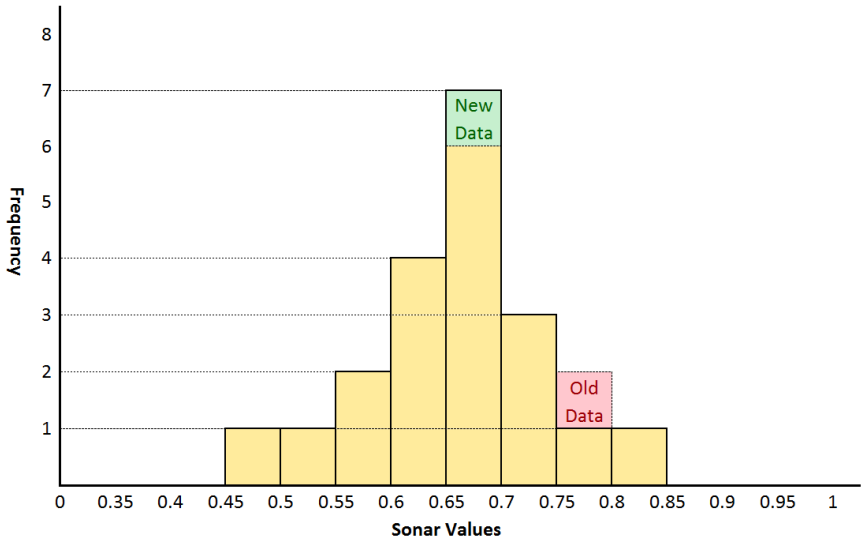

Figure 5.3: A sonar histogram that demonstrates how new value comes in and old value goes out.

- Choosing the filtered value from the histogram is simply done through executing a search for the first maximum count value in the histogram. The index of which the maximum value reside is then converted into the range of value. For example, index 5 represents the range of [0.45-0.50). Afterward, the median value of the calculated ranges is then used as the filtered value , but it is still in the raw form, so further conversion is needed. The figure below will demonstrate how the filtered value is found.

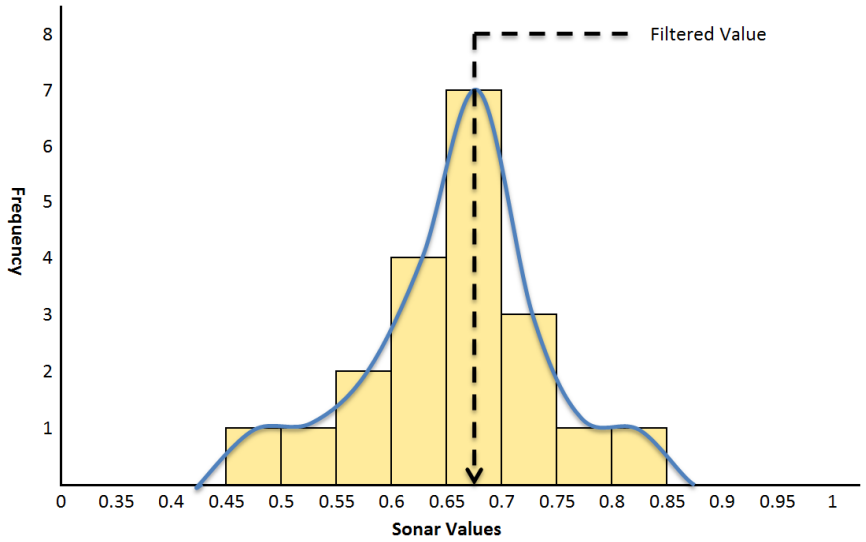

Figure 5.4: A sonar histogram that demonstrates how the filtered value is chosen, that is, the first maximum point.

- The filtered sonar value is subsequently converted into the same metric as the other module in the software architecture so its usage can be consistent and simplified. The conversion is based on a number of pre-recorded value between the sonar value and the actual distance between the robot and the object. Thus using simple arithmetic and by interpolating across the all recorded data point, the actual distance value can be obtained from raw sonar value. This actual distance is in the same space as the relative distance from the relative space.

- Since the Nao has 2 sensors, that is, one on the left and the other on the right, all that needs to be done is to keep 2 histograms fo values and thus producing 2 filtered values to represent each side.

5.4 Result and Evaluation

In the current implementation of the sonar filter a window size of 20 is used, while the bounds of the histogram are ranged from 0.0 to 1.0 with the split size K of 20 slices or parts, which gives a range size of 0.05. In other words, the range goes from [0-0.05), [0.05-0.10) and so on. Since the sonar sensor in the Nao could not detect anything closer than of value 0.3 (which is meant to represent 30cm from the sonar sensor to the object), the first 6 histogram bars or values are truncated into the 7th value range (that is, [0-0.35)). Furthermore, an additional bar is added to the end of the histogram to represent anything of value of greater than 1.0 to signify no objects being presence in the front vicinity of the robot.

The use of sonar filter has allowed the Robot Detection algorithm to be more accurate, as there are less noise in the evidence set. There are still occasions where 2 or more histograms values are close together, but is unfortunately not handled correctly in this implementation. The figure below is an example of the flaws in the current implementation.

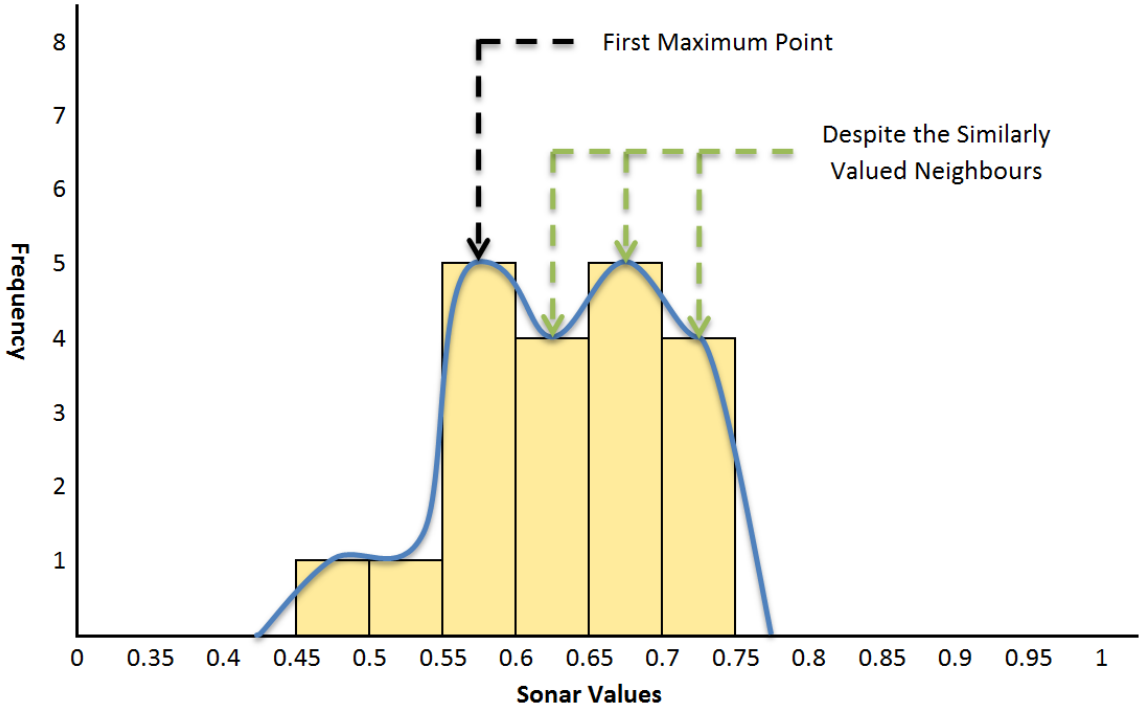

Figure 5.5: A sonar histogram that demonstrates the problem associated with the current implementation

Ideally, similar values across some radial distance should be averaged together in order to produce a more accurate and consistent representation of the true value.

The particular choice of the window size, histogram bounds and the split size in the sonar filter has been working well, that it provides an accurate result with minimal lag. The lag of the filter is roughly 4 to 7 visual frame. Since the Nao is configured to run at the average rate of 25 to 30 frame per second, the perception lag produced by using the filter is roughly from 160 to 280 millisecond.

The table below shows the result of taking measurements and comparing it with the result of the filter.

Actual/True Distance	Detected/Reported Distance	Error
300	345	15%
350	395	12.86%
400	445	11.25%
450	445	1.11%
490	495	1.02%
500	495	1.00%
550	544	1.09%
600	594	1.00%
630	644	2.22%
730	744	1.92%
750	744	0.80%
820	795	3.05%
830	844	1.69%
870	895	2.87%
950	945	0.53%

As seen in the result above, the sonar filter is able to report the correct distance of the object. The results of detecting close object (less than 450 millimeters away) may have a high error, however, during a game, it does not pose any problems as a margin of error of around 50 millimeters are acceptable.

The following is another set of tests that attempt to test the lag of the sonar filter. The reported distances are taken while the objects are being moved away or into the robot. Since the sonar filter can be configured to show the reported distance on every frame, it can be used to measures the how long it takes for the sonar to settle into the correct value.

Actual Distance	Reported Distance After 5 Frames	Reported Distance After 10 Frames
300-500	395	495
400-600	445	545
500-800	645	795

As predicted, the sonar filter has some lag in the system. Most results are correct by the 6th to 8th frames, which equates to roughly 260 millisecond. 260 millisecond is unfortunately longer than expected and may cause an adverse effect on the game.

However, by reducing the window size to 10, the results below can be obtained:

Actual Distance	Reported Distance After 5 Frames	Reported Distance After 10 Frames
300-500	495	495
400-600	545	545
500-800	795	795

Reducing the window size to 10 allows the sonar filter to be much more reactive, however, it starts to become noticeable that the reported distances are starting to jump frequently.

5.5 Conclusions

In conclusion, the implementation of the sonar filter is simple that is unable to correctly handle the more sophisticated scenario. However, in the context of its usage in the RoboCup SPL matches, it is adequate, that could provide a relatively accurate result, while maintaining minimal lag. The filter is also fast in performance and thus does not add on unnecessary delay to the execution of higher level behaviour.

Chapter 6

Robot Filter

6.1 Introduction

This chapter will outline the motivation behind the need for a robot filter, the usage, usefulness and the inner working of the robot filter algorithm. The result and evaluation of the algorithm are also included towards the end of the chapter.

6.2 Motivation

The motivation behind a filter is to remove noise. This is the same case for robot filter. The Robot Detection algorithm, despite its best effort to remove any false or inaccurate results, unfortunately still produces them from time to time. Thus it becomes highly desirable to filter the result of the robot detection. Filter the result allow the higher level behaviour to be more precise in such a way that it does not produce unwanted behaviours or interfere with the vital behaviours.

6.3 Algorithm

The robot filter is a simplified Kalman filter [9] that excludes many of the more complicated aspect of Kalman filter. The robot filter is a simple model that only deals with static observation and does not incorporate noise or in any way attempt to model movement of the other robots. One of the primary requirement of the robot filter is to keep every hypotheses node in relative space. Meaning that the position of the hypotheses nodes are not directly mapped to the absolute X and Y position on the field, rather, it is mapped to the relative position from the viewing robot.

The reasoning behind such design decision is to completely exclude any filtered positioning information from the current viewing robot, as an error in the localisation of the robot could completely throw off the result of the robot filter, thus it is highly undesirable. If any higher level behaviours require the absolute coordinate of these hypotheses, then through the use of simple trigonometry (with the localisation information of the viewing robot), the absolute X and Y position can be determined.

The execution of the robot filter requires the following modules or component in the programming code:

- Odometry information from locomotion.
- Observations generated from the result of the Robot Detection algorithm.
- Variable to store the previous odometry information.
- A main list containing processed hypotheses.
- A secondary list containing queued hypotheses.

The step by step process of the robot filter progress as follow:

- Given the odometry information from locomotion, calculate the difference with the previous odometry information in terms of forward, left and turn parameter values. Then update previous odometry to the current odometry information.
- Iterate through the processed hypotheses list and remove any robot hypotheses that has not appeared for `MAX_ROBOT_LOST_COUNT` frames. Conversely, this is similar to removing or merging hypotheses that have high variances. The value of `MAX_ROBOT_LOST_COUNT` is chosen arbitrarily and depends on the processing power of the device. In the current implementation, a value of 125 is chosen and with the Nao being able to run at 25 to 30 frames per second, this roughly equates to 5 seconds of timeout period.
- Similarly, repeat the same process with queued hypotheses. The idea of using a queued hypotheses list is to avoid creating a hypothesis for an observation that only appears once. The current implementation requires the observation to appear twice in a window of 5 frames. If any candidates satisfy the condition, then it is added to the main list.
- For the remaining hypotheses in the main list, iterate through each node and proceed with a process update. Each node is converted into relative coordinate to absolute coordinate with a temporary assumption that the robot is positioned at the value of X and Y of (0,0). Afterward, the odometry difference is added to the absolute coordinate and converted back to relative coordinate. As mentioned previously, the robot filter is a simple model and does

not attempt to model moving objects, thus only a constant process variance update is added. This can be done by taking the square root of the current variance squared plus the constant process variance.

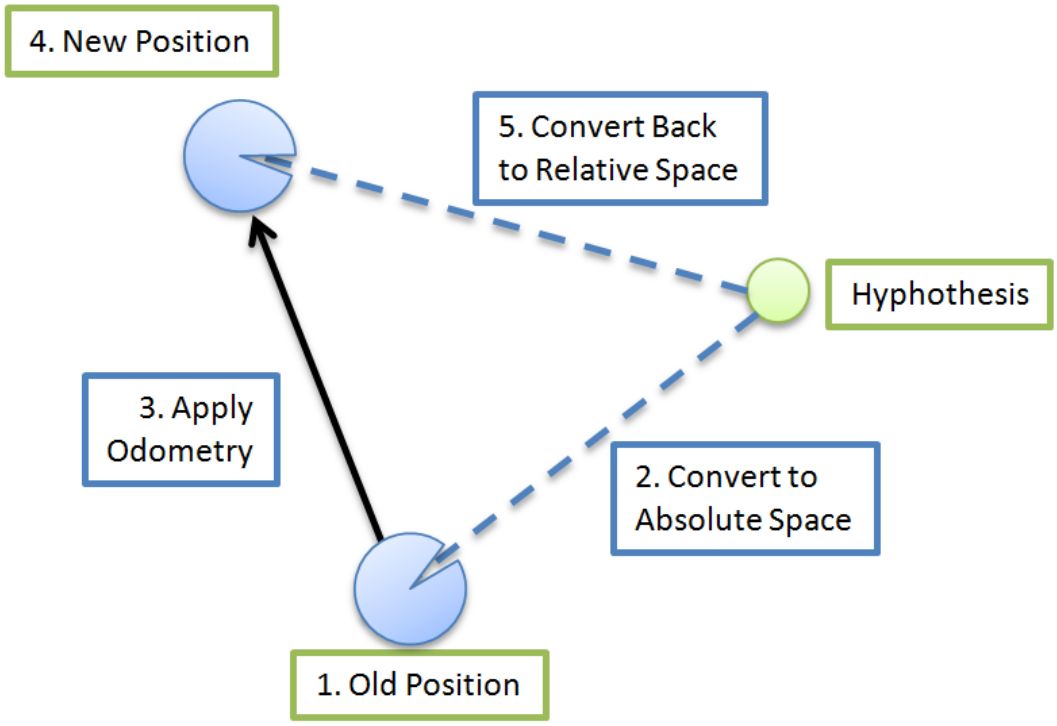

Figure 6.1: The process update phase that demonstrates how the update is carried out.

- For each observation, estimate the variance by employing a simple model where the further away the observation, the higher the variance becomes. The exact formulae used for this step is contributed by David Gregory Claridge. The same formulae is also used when estimating the variance of a field edge observation prior to a local localisation update. Please refer to his honours thesis report for more details [4].
- For each observations, attempt to find the corresponding close hypothesis in the main list. This can be done by simply iterating through the main list and calculate the straight line distance between the observation and the hypothesis. Calculating the straight line distance can be done in both relative coordinate and absolute coordinate. The current implementation employs calculation in relative coordinate since doing conversations are expensive and the resources saved could be better spent on other modules.
- If a matched hypothesis is found then proceed with a Kalman update. This is done by taking into consideration of the variances of both nodes. Then pick a point in the straight line distance closer to the node with lower variances. This new point will be the new hypothesis.
- If no match is found, then add the observation to the secondary list to be processed further in the next iteration

6.4 Results and Discussions

Using the robot filter allows the system to be more accurate in terms of knowing the position of other robots. However, the nature of using a filter means that it contains lag and due to the lack of a more sophisticated model, the filter is only suitable for long-term planning or plans involving far robots.

Figure 6.2: One of the test scenario in testing the robot filter. This image contains three different robots each with a different configuration of the waistbands.

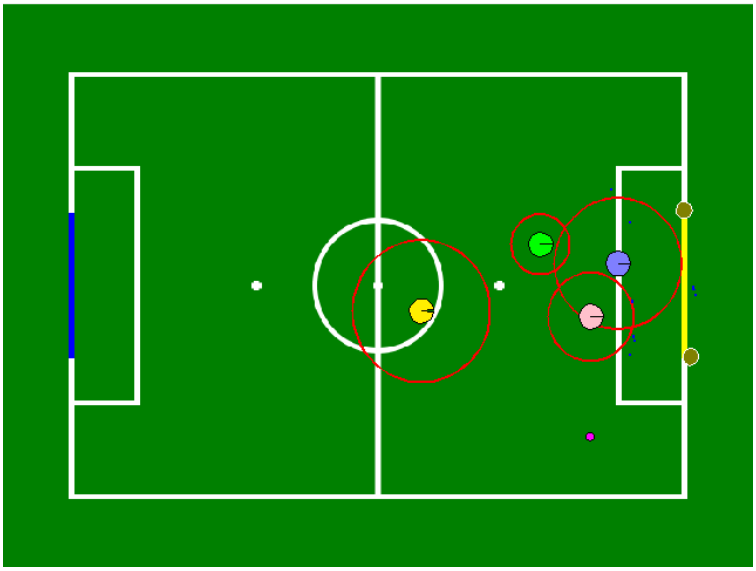

Figure 6.3: The generated hHypotheses from visual observations shows the hypotheses being plotted on the field from the visual observations above.

The filter is not suitable for use in scenario where it involves close robots. This is primarily caused by the method at which Kalman update is carried out. Since a constant threshold is used to find a match between the hypothesis and the observation, hypotheses that are in close proximity to the viewing robot may be progressively updated and would end up in similar position to the previous position, despite being updated with odometry.

An example of this is when an observation is taken from sonar information. The sonar sensors often still trigger despite the robot trying to move away from the object (such as turning on the spot to the left or right away from the object), since the object is still in proximity of being detected by the sensor. This scenario will cause the hypotheses, updated with odometry, to jump back to its previous relative coordinate. Such scenarios are particularly undesirable when attempting to do robot avoidance, since the robot may turn excessively until the sonar sensors no longer detect the object.

Similarly, robots that are in motion, particularly fast robots, may end up being modelled as multiple hypotheses in the algorithm. Unfortunately, none of these problems can be address due to the simplistic nature of the filter.

6.5 Conclusions

In conclusion, the robot filter model is a simple model that only attempts to model static object. The filter is fast and provide accurate data regarding far robots, however, due to the lack of more sophisticated model, robots that are close or in motion may not be handled correctly.

The filter can certainly be improved to include the more sophisticated techniques and thusly able to correctly tracking moving robots and close robots.

Chapter 7

Robot Behaviours

7.1 Introduction

This chapter will outline the design decision in implementing the behaviours for RoboCup 2011, the usage, usefulness and the performance of the behaviours. The behaviours developed for this chapter are specific roles that have to be included and it is vital to have specific roles during the game to accommodate to various strategy. Running all robots on the field with a single identical behaviour is undesirable as there are various penalties and strategy disadvantages that may fall upon the team.

7.2 Team Role Switching Skill

Given the many specialised roles developed for the robot for use during the 2011 RoboCup competition, it becomes necessary and desirable to have a common skill that dynamically assign roles to the robots. The idea is to use the distributed information such as the global positions of the robots and the believe-state of the perception of the ball to determine which robots should be assigned to what roles.

In the current implementation, the role switching skill is structured as a tree. The structure of the tree can be seen in the figure below:

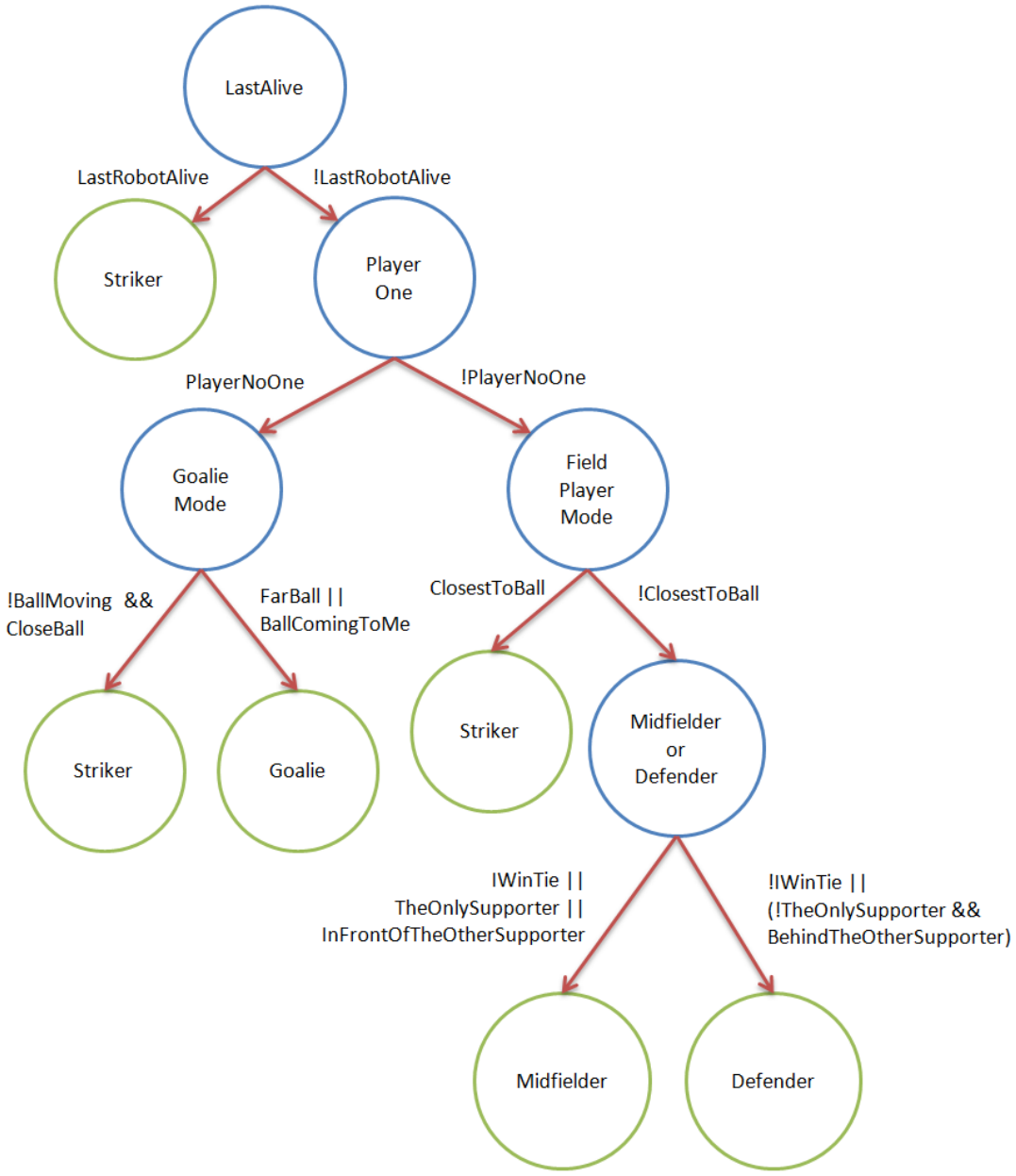

Figure 7.1: Role switching decision tree that shows how dynamic role switching is done. Green circles denote terminal nodes, blue circles are decision making node, while text in black are conditions to reach a node.

Parameters are evaluated at each of the related node and when a terminal node is reached the corresponding behaviour or role is returned to the main function. If a role that is different to the previous role is received, the robot will be assigned to the new role. Since information are distributed, all the remaining robots will be aware of this role switch and will also dynamically switch their role if necessary. For example, a midfielder is transitioned into a striker, hence causing the previous striker to transition out of the role and become one of the supporters.

The development of role switching skill has proved to be difficult prior to the competition as testing the behaviour requires coordination of a full team of robots and it is often difficult to correctly tell the current role of the robot before it is well developed into the role.

Another negative side effect of this problem is when two robots are constantly switching in between two roles and eventually would settle into a dead zone where both robots are assigned to the same role since the robots are within the threshold of each other. This is the most common when determining which robot should become midfielder and defender since the parameters used require the information of the other supporters. A sudden false jump in the localisation of one of the supporters will also throw off dynamic role allocation since the implementations are not able to correctly handle mis-localised robots.

Overall, the development of role switching skill is important. The most important aspect of role switching skill is the transition into striker. This includes the transition from supporters and goalie into striker when certain condition holds. Such transition are usually strict and intentionally done so to avoid having multiple strikers. Scenarios where the role switching excels include:

- The goalie being transitioned into striker when the ball gets too close to team own goal while the previous striker transition out of its role as to avoid kicking the ball into own goal.
- Transition into supporters for positioning and afterward transition into striker when condition holds, such as, after blocking the opponents kick or retrieving the ball after a fail kick.

7.3 Supporters - Midfielder and Defender

The supporter behaviours primarily consist of the role of midfielder and defender. The idea of having a midfielder is to place the robots on the opponents side of the field to handle a number of scenarios such as ball passing or retrieving the ball after a fail kick by the striker (for example being bounced off other robot or the goal posts), while the idea of having a defender is to place the robots on the team's own side of the field for various defending scenarios, such as blocking the straight line ball path from the opponents' strikers or the ball to our own goal.

In this thesis project, the development on supporters are divided among two rUNSWift members, thus only the development that has direct responsibility will be detailed. The supporters behaviour are a collection smaller sub-behaviours that are put together in a format similar to a decision tree. Traversing the decision tree require an analysis of the current situation and evaluating a number of parameters, such as the relative distance to the ball, the global absolute position and the relative positioning information of the striker.

Specific to this thesis projects, there are three sub-behaviours developed. These are:

- A routine that programmed the robot to return to its predefined global position after a time frame at which the ball could not be seen.
- A routine that intentionally blocks (move into) the ball-goal line path between the ball and our own goal.
- A routine that avoid the ball-goal (move away from) the ball-goal line path between the ball and the opponents goal.

The remaining sub-behaviours are developed by Yongki Yusmantia and are related to the positioning of both the supporters and defenders during the game. Please refer to the related report for more information [13].

The definition of the ball-goal line path is basically the straight line between the ball and the middle vertical point of the goal posts. Avoiding the line path require the robot to avoid the triangle region created from joining the ball to the individual goal post. Where as blocking the line path involves intentionally stepping into the triangle region. The figure below will demonstrate the triangle region used in the behaviour.

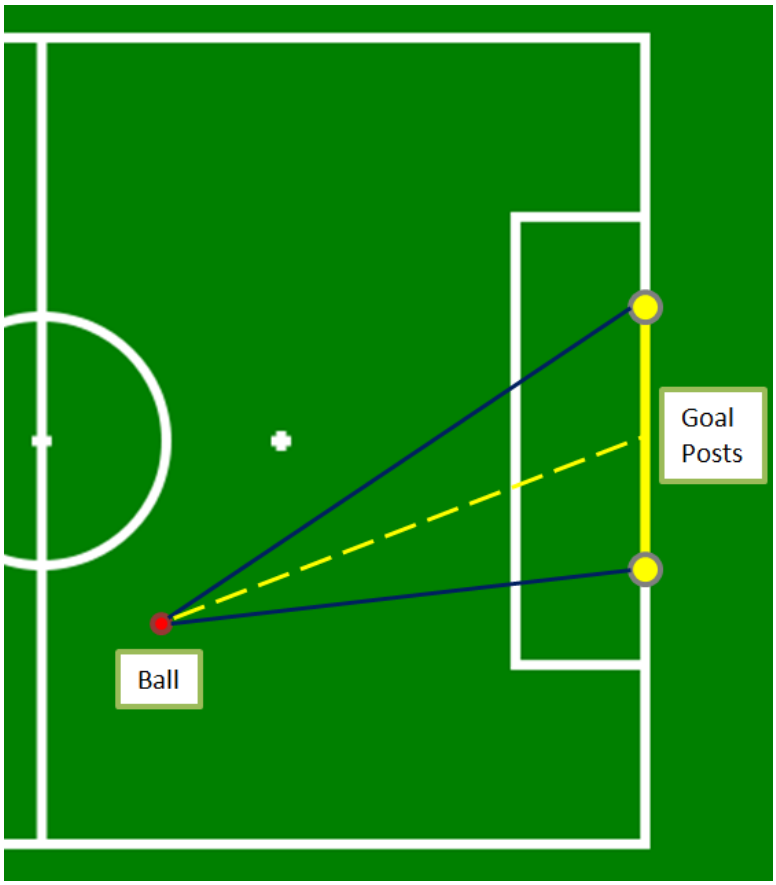

Figure 7.2: Ball-Goal Triangle Line
that shows how the ball-goal triangle line is established.
The dark blue lines are the triangle region, where as the yellow dashed line is
the straight line path between the ball and the middle of goal posts

The supporter behaviours have proved to be a success during the 2011 RoboCup competition where the following scenarios have occurred that change the tide of the game:

- The defender successfully blocked the ball and charge forward to become the striker.
- The defender, being positioned well, was transitioned into a striker by role switching skill when all the other robots are unable to see or approach the ball optimally.
- The midfielder successfully avoided the striker when the striker attempts to kick the ball towards the opponents goal.
- The midfielder successfully retrieved the ball after a fail kick and transitioned into a striker by the team switching skill to continue the dominance.

7.4 Penalty Striker

The penalty striker is a specialised behaviour developed for the penalty phase of the game. The major part of penalty striker is a derivation from the striker skill, which in itself is highly similar to the 2010 rUNSWift code. Thus, please refer to the related report [1] for more details.

The extension developed for the penalty striker behaviour is the inclusion of the Robot Detection algorithm. The behaviour attempts to find the optimal target for the ball to be kicked towards. The step by step process of the progression of the behaviour goes as follow:

- Approach the ball that is placed on the penalty spot. The striker skill routine is called in this phase of the behaviour with the special flag enabled indicating a penalty shootout so that a special behaviour branch can be executed, which leads to the next point.
- The robot stops and attempt to pan the head left and right for a specified amount of time. The panning scan is intended to look for goal posts and the opponents goalie. In the current implementation, if only one post is seen, then the other post is estimated using the current localisation information.
- When information regarding the two goal posts and the opponents goalie are available, the robot will attempt to find the straight vertical distance from the opponents goalie to both of the goal posts and essentially favoring the one with the larger distance. In other words, to find a target location that has the biggest opening.
- During the panning scan, having multiple targets are common and could be caused by various reasons, such as slipped localisation or false robot detection. In order to counter this problem, a histogram of targets are built and at the end of the panning scan, the target location with the highest frequency is chosen as the primary target.
- When the primary target has been established, the special penalty shootout flag is disabled and the penalty striker behaviour will return to striker routine. The striker routine will then proceed to line up routine and kick the ball towards the target position.

Unfortunately, during the 2011 RoboCup competition, this behaviour is not used, since none of the games played proceed to the penalty phase. During the development of this behaviour, various tests are carried out and the robot is able to find the correct target position and kick the ball into the goal. Several occasions occurred where despite producing the correct target, the ball was not kicked towards the target. Upon closer observations, this is primarily caused by the margin of error in the line up routine where speed is prioritised over accuracy when a position is considered to be good enough.

Overall, this behaviour is solid, but is unable to be used during the competition. Hopefully, this routine can prove to be useful in future RoboCup competition.

Chapter 8

Future Work

8.1 Introduction

This chapter will outline various extensions to the current implementation of the Robot Detection algorithm that are unfortunately unable to be included within the given time frame. This chapter are divided into 2 major sections. One is the addition of tools and modifications to the current implementation such that existing functionality can be better maintained and improved over time, while the other section is on extensions of the algorithm to include new features and making the detection more powerful.

8.2 Tools and Modifications

8.2.1 Unified Data Gathering and Decision Tree Learner Suite

This tool is specifically related to the machine learned decision tree developed for the Robot Detection algorithm. In the Machine Learning The Decision Tree chapter, it is outlined that the major flaws in the techniques used is the difficulty of maintaining the process of acquiring and classifying the data sets and modifying or removing current parameters and adding new parameters.

The ambition of implementing this tool is the unification of all the individual steps into one single tool, as opposed to the current use of 4 different tools, which are, manual data gathering and classifying, raw data conversion to WEKA input format, the WEKA software and the conversion of the decision tree to valid C code. Another major flaw with the process is the many manual intermediate process between the use of each tools where a large number of files are constantly being moved and scripts being run. This is error-prone and difficult to notice.

The tool should have a method to smoothly transition from one tool to another and preferably transparent from the users. The tool could also incorporate a way to speed up the classification process and automate certain aspects of it (such as using the test suite database). The tool should also be incorporated into the rUNSWift code base, sharing the same architecture such that many of the common modules do not need to be duplicated in a different set of programs.

8.2.2 Lesser Dependency to Colour Calibration

This modification is specifically related to the machine learned decision tree and the Robot Detection algorithm. The current implementation is highly sensitive to colour calibrations, where a change of significantly different colour calibrations may cause the result to be inaccurate and error-prone.

One way to achieve this modification, prior to converting the raw information to WEKA input format to learn a decision tree, is one could store such information as raw pixels as opposed to the current colour calibrated pixels. Afterward, the specified colour calibration can be applied on the fly when generating the input format.

This modification highly complements the unified tool mentioned in the previous section, where the process of on the fly calibration application can be done transparently and require no user input.

Typically, robots have different camera setting and during the competition, teams may be required to play on more than one SPL field. It is common to have multiple colour calibrations for each robot and each field. Thus, by default, it is undesirable to have one single decision tree for all robots and all the fields, and hoping that it would work. This modification and the addition of the tools may produce different decision tree for each robots and could be deployed easily depending on the scenario.

8.3 Extensions

8.3.1 Robot Pose Estimation

This extension directly relates to the Robot Detection algorithm. The current algorithm only attempts to find the relative distance and heading to the detected robot from the viewing robot, which, essentially allowing behaviours to find the global x and y positions of the other robots. While this can be sufficient in terms of basic strategy play, it is unfortunately lack the depth in the deeper strategy game play.

Robot pose estimation basically refers to the estimation of where the other robots are facing. The granularity of the pose estimation may vary depending on the usage. In most scenario, being able to tell whether the other robots are facing towards or away from the viewing robots can provide a massive advantage to the team, in a sense that, forward facing robots should be avoided and backward facing robots can be ignored. The more advanced implementation of being able to tell exactly the degree of where the other robots are facing can provide an even greater advantage, essentially closing the gap in terms of high level decision making with real soccer game.

8.3.2 Robot Filter Using Distributed Information

This extension directly relates to the robot filter algorithm. The current robot filter is a simple model that lacks many of the more sophisticated feature that could prove to be useful. The information generated by the robot filter is unfortunately tied to a single robot and is not transmitted to another robot in anyway. While a single robot can correctly map the other robots on the field, the robots have blind spot in the camera and most of the times, the behaviours deployed will not intentionally look for other robots on the field. In other words, Robot Detections are done on the fly while looking for the more important objects, which are the ball and the goal posts. Thus, it becomes highly desirable to have a single robot filter that incorporates all the information from all the robots to create a global map.

The addition of a global mapping of other robots on the field can provide a massive advantage to other modules and the high level strategy game play. One advantage would be to re-localise lost robots. Being able to detect own team mates will allow lost robots to quickly localise itself again solely from using the global map. Verifications in the lost robots can be carried out to confirm the position by simply looking at key objects from the proposed position. Another advantage is the ability to incorporate strategy playing, such as passing the ball, path planning, robot avoidance and opponents ball kicking prediction (such as predicting when the opponent is going to kick the ball into the goal).

8.3.3 Distributed Robot Finding Routine

This is an extension to the behaviours where instead of relying on other behaviours to accidentally detect the presence of other robots on the field, the behaviour will allow a coordinated method of looking for other robots. The idea of coordinated behaviour is such that, robots with the more important behaviour, such as the striker, should be allowed to keep going, while those with less important roles can transition to other behaviours that could benefit the team as a whole, such as running this particular behaviour.

The addition of this behaviour can keep the global map updated and thus able to constantly use the map in its most accurate form. Having an accurate map means a more precise behaviour and would be a great advantage to the team.

8.3.4 Robot Avoidance Planner

This is an extension to the behaviours of the robot. The current implementation of robot avoidance is primarily based on using the sonar sensor when the other robots are in close proximity. While sufficient in basic terms, it is common to observe the lag in the sonar sensor that cause a delay in the robot avoidance routine. Such delay proves to be fatal during the game as the robots tend to bump into other robots, getting stuck and having the odometry slipped or in the worse case reset. Either way, an off odometry is erroneous for both the localisation module and the robot filter as the results are no longer accurate and it may take a while for the hypotheses to be correct again.

Hence, the ambition of having a robot avoidance planner is so that, robot avoidance can be planned in advance and be executed as part of the walk planner in such a way that the robot may approach the ball optimally while avoiding other robots. In order to be able to achieve this extension, the global map mentioned in the previous point is highly essential.

8.4 Conclusions

In conclusion, the addition of the unified tools and the modifications to the current implementation will allow the Robot Detection to be more accurate while the extensions proposed will massively improve the functionality of the detection to include the more advanced techniques that will provide an even better result.

The extension to the behaviours to include various Robot Detection routine can be useful in terms of building up the global map and maintain the map during the game. The map can then be used by other modules to provide an even greater advantage that benefits the team as a whole.

Chapter 9

Conclusions

9.1 Overview

This thesis project has addressed the main research in three-fold. First is successfully incorporate a machine learned module in attempting to detect the presence of other robots on the field. Second is the transition into a multi-modal system that gather many evidences and build up the confidence in the result. Finally, achieving robustness and compatibility with the new vision software architecture where new features can be added with minimal changes and impact to the current implementation.

9.2 Summary

All the modules implemented for this thesis project has achieved significant result. The following is the concluding summary of each modules and algorithms.

- **Robot Detection** - has produced reliable results when a full body bounding box is available as the evidence set can be built up accurately for use by the machine learned decision tree. The problems on the mis-detection of objects have been addressed with several hard-coded sanity checks, but could be further improved and integrated into the core steps, particularly into the machine learned decision tree.
- **Machine Learning the Decision Tree** - the J48 algorithm in the WEKA learning suite [10] is able to achieve a correct classification rate of 97.7% with over 1200 data instances. Data are mixed with both true and false data in order to build up a respectable data set that is sufficient to train the tree and to conduct cross validations. The remaining 2.3% of incorrect classification are essentially the same problem outline in robot detection and caused by false data that have similar characteristics to true data, such as the white triangles behind the goal posts.
- **Sonar Filter** - the filter is able to produce accurate result with an acceptable amount of lag. The result data has shown an average error of roughly 4% between the measured distance and the reported distance. 4% roughly equates to less than 15 millimeters of difference, which is completely acceptable for use during the game. The implementation of sonar filter is robust where many parameters can be modified to suite the new of new requirements.
- **Robot Filter** - the filter employs a simplified Kalman filter [9] that attempts to model the placement of other robots on the field. Unfortunately it does not attempt to model moving robots. The implementation of robot filter can be extended and improved with the more sophisticated techniques of Kalman filter and other localisation techniques.

9.3 Closing

The implementation done on robot detection in this thesis report has its significance in the employment of a multi modal system and the usage of machine learning to train the decision tree. The multi modal system allows the algorithm to still detect the presence of robot despite the lack of visible waistbands in the visual frame, which all of the algorithm proposed in the literature reviews are unable to do. In fact, all of the algorithms have heavy emphasis on the presence of waistband.

The use of machine learning also means the algorithm would only improve over time as more data instances are gathered and with the inclusion of the many future works tools and modifications that can be incorporated into the algorithm, particularly the unified tools, the process of conducting machine learning will only become simpler, easier, faster and more accurate, in terms of gathering and classifying new data set.

Bibliography

[1] Brad Hall Brock White Benjamin Vance Claude Sammut David Claridge Hung Nguyen Jayen Ashar Maurice Pagnucco Stuart Robinson Yanjin Zhu Adrian Ratter, Bernhard Hengst. rUNSWift Team Report 2010 Robocup Standard Platform League. Technical report, School of Computer Science and Engineering, University of New South Wales, October 2010.

[2] Thomas Rfer Alexander Fabisch, Tim Laue. Robot Recognition and Modeling in the RoboCup Standard Platform League. *Proceedings of the 5th Workshop on Humanoid Soccer Robots @ Humanoids 2010*, 2010.

[3] Carl Raymond Chatfield. rUNSWift 2011 Vision System: A Foveated Vision System for Robotic Soccer. Honours thesis, The University of New South Wales, 2011.

[4] David Gregory Claridge. Multi-Hypothesis Localisation for the Nao Humanoid Robot in RoboCup SPL. Honours thesis, The University of New South Wales, 2011.

[5] Sean Harris. Efficient Feature Detection Using RANSAC. Honours thesis, The University of New South Wales, 2011.

[6] Bernhard Hengst http://www.bernhardhengst.com/. Bernhard Hengst, 2011.

[7] Oracle http://www.java.com. Java, 2011.

[8] Board of Trustees http://www.robocup.org. Robocup, 2011.

[9] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering 82 (1): 3545. Retrieved 2008-05-03*, 1960.

[10] Geoffrey Holmes Bernhard Pfahringer Peter Reutemann Ian H. Witten Mark Hall, Eibe Frank. The WEKA Data Mining Software: An Update. *SIGKDD Explorations, Volume 11, Issue 1*, 2009.

[11] J. R. Quinlan. C4.5: Programs for Machine Learning. *Morgan Kaufmann Publishers*, 1993.

[12] Judith Mller Armin Burchardt Erik Damrose Alexander Fabisch Fynn Feldpausch Katharina Gillmann Colin Graf Thijs Jeffry de Haas Alexander Hrtl Daniel Honsel Philipp Kastner Tobias Kastner Benjamin Markowsky Michael Mester Thomas Rfer, Tim Laue. B-Human Team Report and Code Release 2010. Technical report, e Universitt Bremen and the German Research Center for Artificial Intelligence (DFKI), October 2010.

[13] Yongki Yusmanthia. Simulation and Behaviour Learning for Robocup SPL. Honours thesis, The University of New South Wales, 2011.