An Approach to a Complete Recognition Module for Legged Robots in the RoboCup Domain

February 12, 2011

Degree Project in Computer Science and Engineering
Hossein Motallebipour, December 21, 2010 (revised February 12, 2011)
hosseinm@ieee.org
Department of Computer Science and Engineering
Lund Institute of Technology
www.lth.se

”The contents of the evaluated report reflects the work done during the project work (VT2004). The material gathered then is definitely sufficient for a good report on machine vision used on AIBO during this project. In other words, I am satisfied with the work Mr. Motallebipour did during this project.”

Dr. Jacek Malec, thesis supervisor and the head of the TCC, Fall 2009

”In my personal opinion only Hossein’s work is complete enough to constitute a successful exjobb ...”

Dr. Slawomir Nowaczyk, team captain for TCC-Lund, Summer 2004

Abstract

A need for autonomous electro-mechanical machines performing tasks that can save human efforts and even be deployed in hazardous environments along with the research in the field of human mind and body has led to new fields often called *AI Robotics*. To provide a mean for stimulating students and researchers around the world for promotion of this new field, an annual world championship, *Robocup*, has been organized with start in 1997. During these competitions, the state-of-the-art technical achievements within *Sensor Systems*, *Mechanics*, *Artificial Intelligence*, *Machine Learning* and many other related fields are presented by the participants. The final goal is to create autonomous humanoids that can win a soccer match against humans by 2050.

Computer Vision and Robotics are tightly bound together as vision is the most important part of the sensorimotor system enabling a robot to act in a dynamic environment. This includes recognizing landmarks for localization, obstacles for collision avoidance, and objects for manipulation. To go one step further in this direction, the challenge within *Sony AIBO quadruped ERS-7* class for year 2004 was to recognize objects in different lighting conditions and to recognize and remember the position of newly detected landmarks. For this class of robots, the playground was a green carpet, surrounded by white walls. Two nets and four beacons in four different two-color combinations were the only permanent landmarks.

The object recognition module was decided only to be designed for rendering the task of recognition of all predetermined as well as unknown landmarks on and around the field given a set of image frames containing shapes presented as color blobs. Most importantly, the module should be designed in a way that it could recognize the regular objects independent of the lighting conditions and calculate the angle and distance to each one as well as the confidence for the calculated results. The outcome was thereafter to be saved in a common space for the other modules to use and act upon.

As the most challenging object to recognize and estimate the distance and view angle to is the ball, the step by step approache to a successful solution is presented in the report. The next subject mostly discussed is how to recognize the nets, where several factors affect the precision and confidence. Other subjects as filtering the information, how to consider the pose of the head, and how to use a voting system for raising the confidence are discussed thoroughly. Also some pros and cons of each method are looked at. For the reasons discussed throughout the text, the recognition was primarily based on color verification and not the shape of the blobs. The module was implemented in *Matlab* for unit test and to verify the theories and later on implemented, assembled, and compiled in *C++* for the final run.

Although individual tests in the lab environment showed on a successful module in recognizing objects on and around the field, lack of a consistent color segmentation

table resulted in a poor quality of recognition in variable lighting conditions during the competitions. Nevertheless, the module is reusable as the methods are as general as possible to fit different platforms with diverse frame rates and dimensions.

Regardless of the outcome, the thesis work presented in this report lays a ground for a vision module that can perform the task of recognition of known and unknown objects. In particular, it reports on the methodology used for recognition of objects with estimation of their distances and angles to the AIBO, for which a software vision module is implemented. The contribution of this thesis is the methodology for detection of the peripheral of the ball blob, using a voting system, an attempt to merge blobs, specially for finding the rectangular shapes and correct estimation of their sizes, and an investigation about using rule based verification of the objects, methods that had not been used before by any teams in previous years.

Acknowledgements

To ”the one” who created us... creators!

On my way to completing this report and my Master’s degree I have received incredible support from my wife whom, after only a few years, I feel I have known my entire life. I would also like to express my gratitude to my parents and brothers (read friends), for their patience, help and support, through all these years. I even appreciate my newborn son giving me the strength to take the last steps. Many thanks to my life-long friend *Engineer Reza Aliabadi* for his many remarks on the report. I hope my dream of living closer to him comes true, someday, somewhere.

The next person in line to thank is my supervisor *Dr. Jacek Malec* for his "being”—and not for his not being where he should be most of the time due to the workload—which has inspired me to choose AIRobotics as my field of interest. *Dr. Slawomir Nowaczyk*, the team captain for TCC in Lund with his fantastic coding style, was not only a supportive gentleman throughout the entire project but also the teaching assistant for my first course in Machine Learning, opening a whole new world to me. The always serious but incredibly friendly *Dr. Henrik Stewénius* taught me first time how to write a technical report. I wish him well wherever he is now. My appreciation goes also to *Engineer Phat Tran* for his being a good classmate and my walking C/C++ encyclopedia during the project and long before that; a friend that never gives up on you and that you can always rely on. *Nora Ekdahl*, the study counsellor guided me in the toughest time of my education. She has always answered all my questions and directed me in the correct way to the graduation. Without feedback from *Dr. Anders Nilsson*, the last man to review my report, I wouldn’t have made it. I appreciate his willingness to take this responsibility. Finally I would like to thank *Dr. Martin Höst*, Director of the M.Sc. Program in Computer Science and Engineering at Lund University.

Foreword

> *Computer Vision* refers to processing data from any modality which uses the electromagnetic spectrum which produces an image.
>
> Murphy [29]

It is spring semester 2004. The RoboCup World Championships 2004 in Lisbon (Lisboa) are ahead. Five interested students are gathered to work on Sony's programmable dogs, *AIBO*s, and build a working system from scratch with a deadline about the middle of June the same year. Traditionally, Lund university's RoboCup Team, Team Chaos[1] Challenge (TCC) was put on the Challenges of the RoboCup competitions and I have the pleasure to be one of the team members to shoulder this hard task. I choose from the beginning to work on the motion, but due to some considerations about the man power I reconsider to change to the "vision" group later on. And this is just what I needed. Computer vision has always been one of the most fascinating areas in the field of *autonomous robots* and when I choose the motion system, for a change, I am *forced* again to work on my area of interest. Well, nothing to complain about.

This report is aimed to present how an object recognition module was constructed for AIBOs used for the challenges.

Only a part of the ideas, explained in this paper, were actually used. Some code was implemented but never used - for the reason explained in its place - and some ideas were worked on but never implemented. Still, I find it necessary for the interested reader to get a complete picture of how the whole module was planned to be.

Since we all are humans, no matter how much you try, there is always improvements to do. I hereby ask my dear supervisor and other experts on this field for their indulgence.

[1] "Team Chaos" (formerly Team Sweden) is a multi-university team which has been competing in the 4-legged robot league or RoboCup 2004. ([23], Abstract)

Contents

Chapter 1

Introduction

> Intelligence is in the eye of the observer.
>
> Brooks [7]

Preparing an electromechanical body to make "intelligent decisions"[1] is time consuming, but still fun. However, this needs structure, teamwork and lots of code. One can hardly sit in front of a PC and just make the AIBO intelligent in a couple of days. First step in this long procedure is to see what we have to start with. This is presented in 1.1. 1.2 is a brief text about the environment in which the robot moves. The challenges of year 2004 for this class in the competitions and the assignment this thesis work deals with are outlined in 1.3 and 1.4. In 1.5 the content of the following chapters are explained in short.

1.1 The Platform

The platform was already decided upon: Sony's AIBO ERS-7 (**A**rtificially **I**ntelligent Ro**bo**ts)[2] - the front and back views are presented in Figures 1.1 and 1.2.

AIBO is currently one of the best and cheapest platforms to do research on in the field of Robotics and Artificial Intelligence. "The AIBO programming language, built on top of C++, is provided by Sony as the OPEN-R SDK"([39]), serving as an abstraction layer to the *Aperios*, which is an object-oriented distributed realtime operating system. OPEN-R programs are built as a collection of concurrently running OPEN-R objects. An OPEN-R object is implemented using a C++ object The communication between objects is done using message passing; see [16] and [39]. For more information please refer to [40].

[1] In 1955, Assistant Professor of Mathematics, at Dartmouth College, J. McCarthy, gave for the first time a definition for *artificially intelligent machines*, which, according to him is a machine that

> would tend to build up within itself an abstract model of the environment in which it is placed. If it were given a problem it could first explore solutions within the internal abstract model of the environment and then attempt external experiments.

[2] An intelligent robot is a machine able to extract information from its environment and use knowledge about its world to move safely in a meaningful and purposive manner. ([2], p.2)

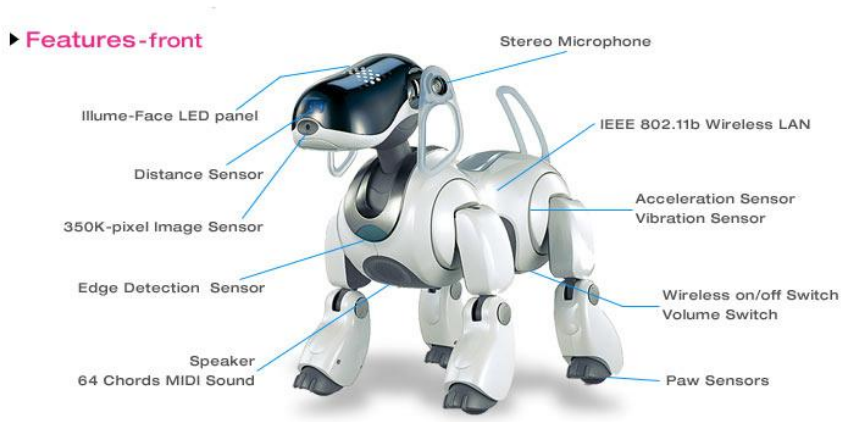


Figure 1.1: The front view of the AIBO ERS-7, White Pearl; picture is copied from the website of Sony Corporation

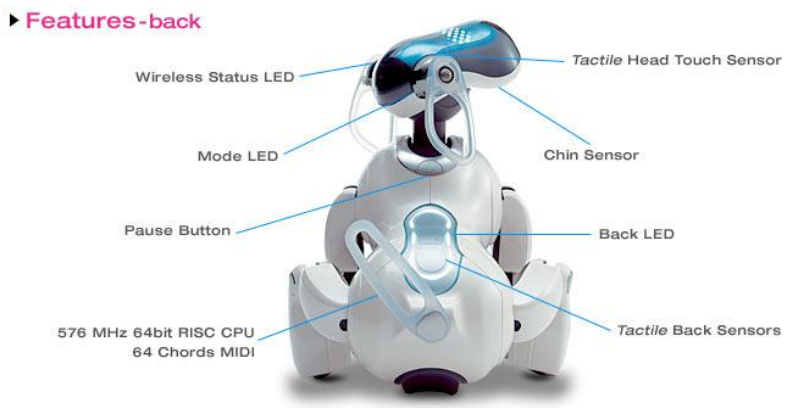


Figure 1.2: The back view of the AIBO ERS-7, White Pearl; picture is copied from the website of Sony Corporation

1.2 The Environment

AIBO's "playground" is a $3100 \times 4600\ [mm]$ green carpet, surrounded by white walls. Two nets and four beacons, in four different two-color combinations, are the only permanent landmarks. The inner side of one of the nets is totally yellow and for the other one totally blue. The four beacons, with the combinations blue over pink, pink over blue, yellow over pink, and pink over yellow are positioned in the four corners of the plane. The ball is an orange plastic sphere with a diameter of $85\ mm$; see Figure 2.4 for a sketch and B.1 for an exact description of the environment.

1.3 The Challenges

The second step is to see what we are to have at the end. Within this context, the problem at hand was to program the platform in order to carry out the following tasks and cope with the problems that could arise in the meanwhile (refer to Appendix B.4, for the

complete official descriptions)

1. The Open Challenge: as the name suggests, is open for the teams to show their creativity. Vision may or may not be playing an important role in this part, depending on what tasks are planned to be performed by the robot.

2. The Variable Lighting Challenge: the robot is to make as many penalty shootouts as possible within a specified period of time with static opponent robots present on the field, which is exposed to varying lighting conditions. The vision is supposed to be robustly designed in such a way that it can perform reasonably well independent of the lighting conditions.

3. The SLAM Challenge: in a first step, the robot is allowed to walk around on the field and try to learn all present, both previously unknown and known landmarks, on and around it. In a second step the known landmarks are covered up and the robot is to localize itself on the field.

1.4 The Assignment

To design a module, rendering the task of recognition of all predetermined as well as unknown landmarks on and around the field, as described in the previous section, given a set of image frames containing shapes presented as color blobs. Most importantly, the module should be designed in such a way that it could perform the task of recognition for the regular objects on the field successfully, independent of the lighting conditions. The set of objects that needed to be recognized for each challenge was

Challenge 2: The orange ball, the goals, the beacons and the opponent dogs considering the lighting conditions.

Challenge 3: Regular landmarks on the field, consisting of the beacons and the goals and also unknown landmarks.

Challenge 1: As far as it concerned this particular challenge, there were no activities planned and therefore no other objects were to be recognized for this case.

For all cases, estimating the distances of the objects to the robot was also to be provided by the vision module to make the task of localization possible.[3]

1.5 Organization of the Thesis

The report begins with a description of the vision module in chapter 2 as it motivates what the vision module should handle and what should be left to the other parts of the TCC system. That is the analysis of the methods used and how the initial ideas converged to a basis for the implementation of the system. The following chapter 3, will present a theoretical approach to which methods for object recognition have been tried for this particular task.

Several of the practical solutions to the problems presented in chapter 3 are shown with motivations and examples in chapter 4. An evaluation of the system used for the

[3] Refer to B.4.3 for a thorough description of the challenges.

competitions is presented in chapter 5. The report ends with a conclusion about the whole module in chapter 6 and presents some ideal solutions for future work.

Added to the end of the report is the appendix with some basic facts about the AIBO, rules of the competitions, some investigations for future works and the numerical results from the evaluations.

Chapter 2

A Prelude: Vision Module

> I require some originality in thinking. If it is lacking, I call the performance at best reasonable behavior.
>
> Braitenberg [6]

In this chapter follows a discussion that in fact belongs to the introduction since it is not a description of the thesis work but an outline of the whole vision module. That includes even the *segmentation* and *object recognition (sub-)module*. In section 2.1 the emergence of what came to be the architecture of the whole system, implemented by TCC, is presented. Section 2.2 is a short description of segmentation process to produce what was later used in the object recognition module. Section 2.3 motivates then the work done throughout this thesis.

2.1 The Architecture

From the beginning, we decided the architecture of the system to be as in Figure 2.1, which is a schematic, showing how different parts of the system should interact with each other. The system to be programmed by the team is divided into the main blocks

- Vision,
- Localization,
- and Behavioral Control.

Vision and, hence, Localization are depended on a system that can provide them with the necessary data for extracting the information that Behavioral Control needs, to decide upon the reactions to the environment.

On the other hand, Behavioral Control will need to make the physical *interaction* with the environment possible. That is where the Effectors and the Wireless come into the picture. The first one is supposed to enable the interaction with the joints of the dog - controlled by OVirtualRobot - and the second one to be a channel for exchanging information with other dogs, if necessary, which in turn uses the TCP-Gateway.

Here, the first real problem is introduced. That is, whether to let the Behavioral Control to control the head joints, or the Vision-Localization to do it just as Cerberus, [24], presented it in their paper. Behavioral Control is logically the block that should

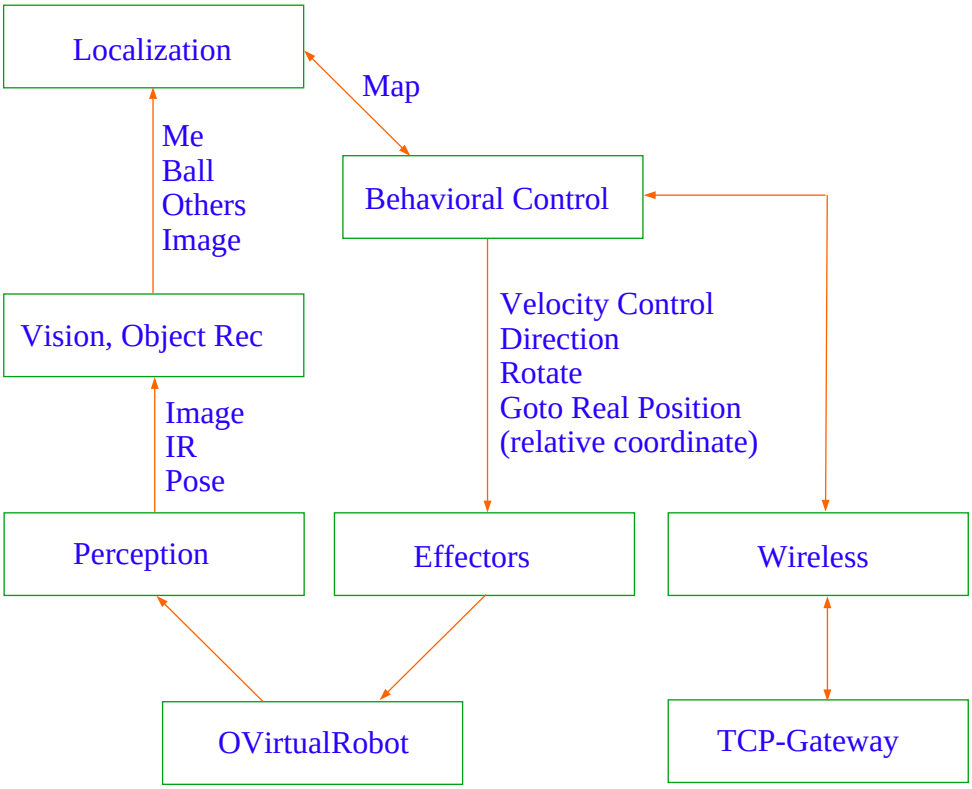


Figure 2.1: The older system architecture was discarded already after a month

have this control, because once an agent can localize itself in a room, it decides where to look next and that is a part of its behavior. But this is far from the optimal solution. Here, when the dog is looking at an object that it is not able to percept perfectly, it is to turn the head toward and focus on the object to ascertain itself of the object being what it "thinks" that object is. This makes an interference with the task of the Behavioral Control. This was a thoroughly discussed problem during the whole project. And not only this project:

> ..... The paradigm in vision shifted to a philosophy where perception exist to support the behaviors of a robot. Furthermore, the vision processing for each behavior might necessitate radically different representations and algorithms. The division and encapsulation of vision processing into compact, well-defined routines was also extended to pan-tilt control..... Behavioral-based vision has arrived!
>
> Kortenkamp [22]

Meanwhile we came across the idea of using *Tekkotsu* from The Carnegie Mellon University [44]. The advantage is that it already has a solution for the bottom blocks of our system architecture, namely, the Perception, the Effectors, and the TCP-Gateway.

What we need now, is just to divide "our part of the system" into three blocks,

- Vision, containing Object Recognition and Localization,

- Behavior, deciding upon the actions and reactions of the AIBO,
- and Wireless,

all interacting with a common block called the *World State*, WS, in a *Round Robin* fashion, as in Figure 2.2.

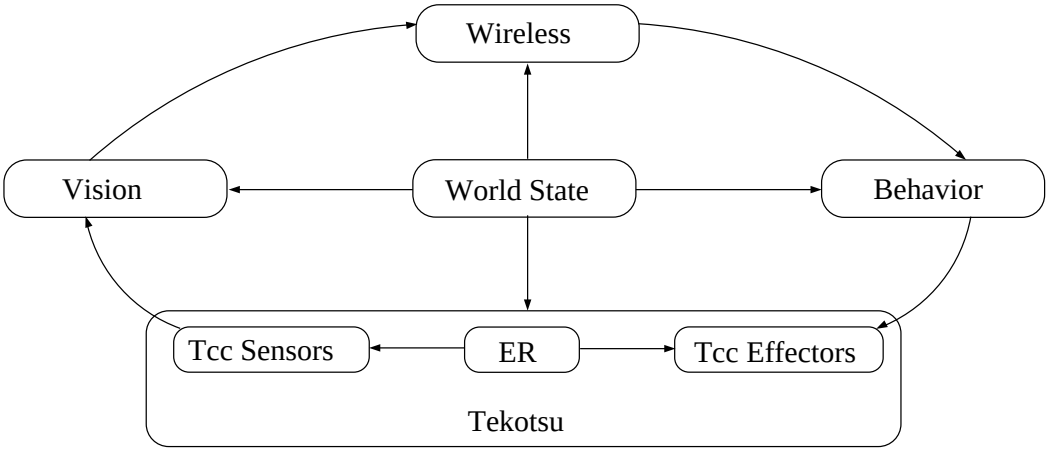


Figure 2.2: The system architecture used as the final solution. The four upper blocks were totally implemented by the TCC

The following pages are a thorough discussion about how and on what bases the Object Recognition module of the Vision block was constructed.

The objects on the field can be divided into 3 different groups:

- Immobile objects, or Landmarks: For the robot to be able to localize itself on the field to perform actions that helps it reach its goals. These are the beacons and the nets.
- Mobile objects: Those that the AIBO usually performs some actions on. The usual object of this type is the ball.
- In the case of the challenges there is a third group with immobile objects that cannot be recognized as landmarks. These are opponent players, in the second challenge, and unknown landmarks, in the third challenge, the SLAM.

In order to succeed with the task of localization *dead reckoning* can be a solution. Dead Reckoning (derived from deduced reckoning) provides information regarding how far a vehicle is thought to have traveled based on the rotation of its motors, wheels, or treads (odometry). ([2], p.248)

But anybody that ever has employed robots for performing a task in a real or experimental world, has faced the problem that "although shaft encoders can provide highly reliable positional information..... through direct kinematics, in mobile systems they can be very misleading" ([2], p.248)

There have thus to be other alternatives to replace the dead reckoning with, or at least make a composition of this method and others in order to have a reliable Localization-

module. In this respect, the vision has been an inseparable element of the *sensor suites*[1] in robots - of the same obvious reason that humans are depended on it.

Our vision system is based on two modules:

Recognition: The object recognition[2] problem can be defined as follows. Given

1. a scene consisting of one or more objects chosen from a collection of objects $O_1, O_2, ..., O_n$ known a priori, and
2. an image of the scene taken from an unknown viewer position and orientation,

determine the following:

1. Which of the objects $O_1, O_2, ..., O_n$ are present in the scene
2. For each such object, determine its position and orientation relative to the viewer. ([34], p.751)

The sensor suite used in this module actually contains only the camera, but as is explained in section 3.9 we will also need values from the neck and head joints, in order to make additional computations.

Localization: is highly depended on the recognition module. It needs a list of all detected objects, their relative distances and angles to the robot in order to compute its global position and orientation on the field. A separate issue that is not going to be discussed in this report.

2.2 Segmentation

The information necessary for the recognition module is provided by the segmentation module, which divides every image[3] taken by the camera into regions of different colors (segmentation). "The region is often referred to as a "blob," and the extraction process is known as blob analysis."([29], p.228) Blob analysis is a relatively straightforward algorithm. It involves first thresholding the image. The resulting binary image is then evaluated for connectivity. The so created groups are considered blobs. The final step of blob analysis is gathering data about the blobs, for instance, their shape or image as well as their relationship to one another.

According to B.Schiller et al. in their article *Vehicle Guidance Architecture for Combined Lane Tracking and Obstacle Avoidance* ([22], p.167)

> Although the blob analysis in straightforward, it is usually computationally expensive. Connectivity analysis alone has several algorithms designed

[1] ([29], p.203)

[2] Detection is not the same thing as recognition. Detection means that the robot is able to identify a face, which is reactive. Recognition would label the face and be able to recognize it at a later time, which is a deliberative function, not reactive. ([29], p.246)

[3] An *image* is essentially a way of representing data in a picture-like format where there is a direct physical correspondence to the scene being imaged.

> to enhance computation time. Determining the relationship between blobs can also be costly.

And as a matter of fact, this is the main reason why in our case the frequency of the frames used by the whole system drops from 30 possible to only 14 $frames/s$, leaving, consequently, not much time for the recognition and localization modules to perform exhaustive methods. This is the first reason for implementing an intuitive module based not on the best present algorithms for object recognition.

TCC applies the hybrid method for color segmentation based on *seeded region growing* (SRG) ([45]) used by the Team Chaos since the competitions in 2001.

The camera in AIBO uses the *YCbCr* system to represent the colors. The image is first segmented into colors by the hardware in AIBO. In Team Chaos's solution the resulting image is used to build seeds, using a simple thresholding algorithm. Then the SRG is applied to consider the difference between each seed and its neighboring pixels. By thresholding the differences the seeds will hopefully grow to produce what we call the color blobs, or blobs. Blobs of the same color are then merged together if they are close enough.

2.3 Object Recognition

By calibrating the colors, the thresholds are changed to include or exclude different sensed YCbCr combinations in an area that is specific for a particular color. An observation made when the calibration of the system was still not prepared was

> It may be better if we look for a landmark even *within* the pink blob, because the walls are sensed by the AIBO as pink (see Figure 2.3).

Since the whole recognition is based on colors, it is a terrible situation that may occur if the colors are not properly calibrated, an example of which can be found in Figure 2.3.

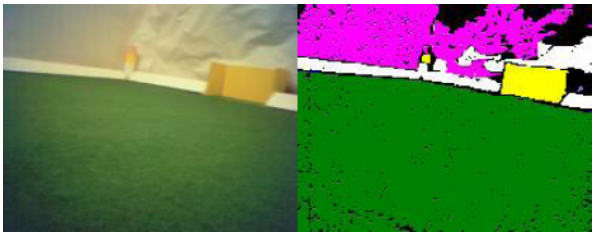


Figure 2.3: Evidence of the white walls being segmented as pink in some frames. See the image database in the appendix.

Sometimes this *was* the case. Many functions had to be removed, because of the difficulties with the color segmentation. Adding extra functions to a sensitive system, with malfunctioning color segmentation makes the procedures of recognition more difficult to cope with. An example is that if the AIBO cannot see every object in its distinct color, then putting intuitive constraints on the localization of the ball on the field (see the table on page 91) will result in poor or illogical estimations of its position. Suppose that, because of wrong segmentation, a yellow goal behind an orange ball is seen as

orange. Then we have an orange blob with white on left, white on right, white on the upper side and green under. This would mean that the dog is closer to the ball, than it actually is, standing with its face pointing to the ball, and a wall behind it, which could be anywhere on the field, except in front of either of the nets. On the other hand, the poor segmentation had its own advantages. That is, many methods, implemented, planned or suggested in this work are results of attempts made to encounter the problem of poor segmentation and to carry out the task of recognition without relying solely on colors.

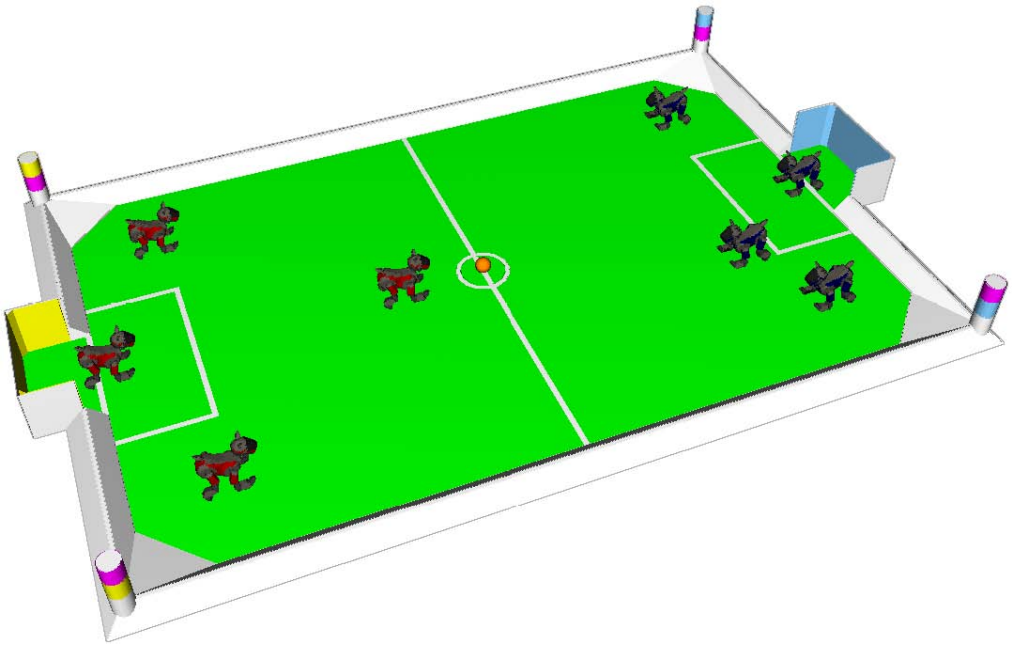


Figure 2.4: Field Colors

Also the light changes may, as expected, affect the blob building. For example, when the first table-created by the calibration tool made by C. Axelsson and J. Törner ([3] and [4])- was loaded onto the memory-stick, the AIBO could see colors Orange, Carpet and lots of yellow, instead of white. One possible explanation could be that, since the table was produced using pictures taken in low, indirect daylight, the test pictures taken, in stronger light conditions, will be experienced as yellow everywhere. And that was probably the case. Pictures for the color calibration were taken on a cloudy day, but tests were done on a sunny day with the sun shining through the window, lighting up some parts of the field.

The procedure of implementation was as follows: Most of the methods, particularly those having a mathematical basis, were first implemented in Matlab in order to test the idea itself. If it showed that the method worked well, then it was also implemented in C++, where each single method was again unit-tested to avoid misprints and malfunctions. After that the whole vision system, where possible, was tested with hand-made test functions, and then merged into the whole system and tested on the AIBO.

G. Dudek and M. Jenkin have made the following observation regarding sensors in the context of mobile robots:

- Real sensors are noisy

- Real sensors return an incomplete description of the environment
- Real sensors cannot usually be modeled completely

They also add that ([11], p.52)

> Understanding the extent to which these three generalizations hold can lead to considerable difficulties when "ideal" algorithms are applied in the real world.

This is the second reason why the presented object recognition module is implemented on an ad hoc basis: to avoid unnecessary complications and consequently waisting time and processing power.

Chapter 3

The Object Recognition Module

> The fundamental issue is that Artificial Intelligence and Computer Vision have made an assumption that the purpose of vision is to reconstruct the static external, world (for dynamic worlds it is just supposed to do it often and quickly) as a three dimensional world model. I do not define that this is possible with the generality that is usually assumed. Furthermore I do not think it is necessary, nor do I think that it is what human vision does.
>
> Brooks [7]

Before proceeding with the methods for recognizing the objects, we should note that the farther the object is, the fewer the number of pixels and the more strict the shape conditions should be.

Hence, in order to separate noise from real objects, we will have to decide a least number of pixels for each object, depending on how large the object is. For example, we have to measure how large the image of the beacon is to be at least, if it is sensed by the dog from the other side of the field; refer to Figure 3.1.

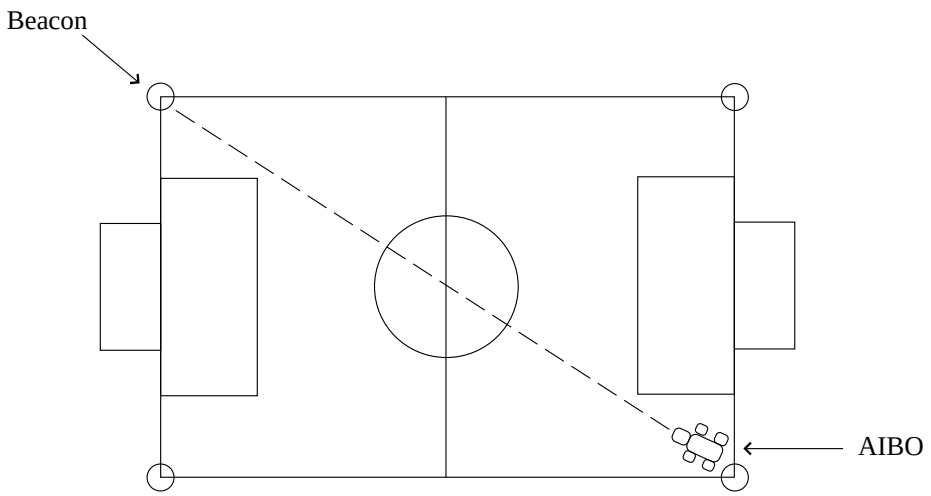


Figure 3.1: The dashed line shows the largest distance between the robot and an object on the field, which indeed is a beacon

In the following sections, methodology used for recognition of all the objects are discussed and motivated; the beacons in 3.1. This section contains also a motivation

about the reason that the beacons are the first objects to be searched for in an image frame. Recognition procedure of the ball is presented in 3.2, the nets in 3.3, the opponent robot in 3.4, and non-predetermined objects in 3.5. Thereafter follows the section 3.6 where determination of angle and distance to objects is explained. The usage of a median filter for establishing the final results from any calculation is then motivated and explained in 3.7. As the robot's head has 3 degrees of freedom, 3.28, the image can in some cases be turned in any possible way, changing the situation for the calculation of distances and recognition of shapes. Section 3.8 is about how to make the necessary changes in order to deal with this problem. Pose of the head can also help to verify the calculated distances, as discussed in section 3.9. At last, section 3.10 shows how the value of confidence for recognition of each object is decided and passed to the Localization.

3.1 Finding the Beacons

In all but two of the revised reports from last year (2003), ([9], [10], [19], [24], [25], [26], [30]), the teams begin their object recognition with beacons, then the ball, the nets and then the robots. The German Team 2003 starts with the ball and then continues with the "field lines, goals, flags, and other platers..."([35]), whereas the University of Texas Austin Villa ([42])"start with the goals because they are generally the largest blobs of the corresponding colors and once found they can be used to eliminate spurious blobs during beacon and ball detection." Then they search after the ball, then the beacons and the rest of the objects.

As is mentioned in UNSW's report ([9], p.29), "Beacon Recognition 1: If two boxes' min/max, height/width, are reasonably close, then they can be beacons." We use this obvious fact to find the beacons.

We should note that putting too many constraints on the shape of the two beacon parts would filter out the possible candidates for being a beacon. Take also into account that beacons do not always seem square-shaped, since, for instance, the cylindrical beacon, looked at from a low stance, would be like the one in Figure 3.2.

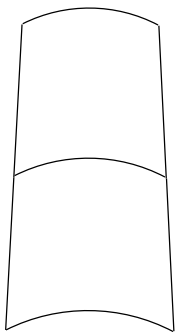


Figure 3.2: How a beacon will look like from an AIBO's point of view, standing close to it. As expected, the two parts are not perfect squares

If we really were interested in making such constraints we would have to investigate if the roundness of the top and bottom of the beacon can be checked to be within certain bounds.

The same goes for the nets. A net may look like the one in Figure 3.20a, if it is looked at from a place close to one of its corresponding beacons, or like Figure 3.20b, if it is looked at from a position in front of and close to it.

The only exception is of course, as can be seen in Figure 3.3, that if the ball moves along the straight line l, the face of the ball toward the eye is the same circular shape regardless of the position of the ball along l. This is not the case when the object is not circular, as with the beacons and the nets.

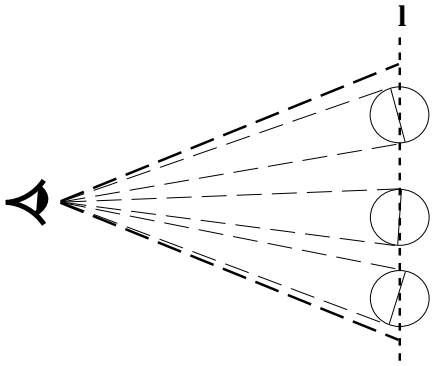


Figure 3.3: The ball is always round, no matter from what angle it is looked at

The color pink is one of the colors that usually is simple to find on the field without any mistakes. This is what the experiences have shown, assuming that the colors are adequately segmented. Besides the fact that this is the primary reason why the whole recognition starts with the beacons, they are the highest objects on the field. Hence, any blob higher than the beacons can be ruthlessly discarded.

A question will be: what if we just see a little part of a pink blob? Then it will not give us its full length and we will not find the true center of the other part of the beacon either.

The answer is that we know that a beacon is made up of two parts with two distinct colors. The solution is hence to continue the search through the second part and hope to get its full height. The part with the largest height is to be chosen for estimating the distance to the beacon.

Also, experiments have shown the fact that AIBOs cannot move as close to a beacon as to all the other objects on the field and we can, most of the time, estimate the distance with a quite good accuracy. We can always use the larger part of the beacon for deciding the distance by comparing the sizes of the two, in number of pixels.

3.2 Finding the Ball

Three approaches were chosen one after another and thoroughly worked on. The first approach shortly referred to in 3.2.1 showed to be a computationally costly method for recognition of adjacent points and hence discarded very soon. The method explained in 3.2.2 looked straight forward and cheap and showed promising results, in the beginning, but was discarded when the problem of short range ball recognition was encountered.

The last approach discussed in 3.2.3 is what was used later in the implementation of the module.

A logical hint that makes the recognition even more reliable is the distortion at the edge of the image frame presented in 3.2.4.

3.2.1 Using Hough Transform[1]

If the shape had been a straight line, this method would be in general easy and reliable to use. But when it comes to circles, although the transform is also a circle, it is still under investigation and not quite as easy to perform as for a straight line. Also many calculations are needed, what puts our platform under pressure and can cause a decrease in the number of images analyzed per second.

3.2.2 The Idea of Using Rays

Consider now the following problem

> If a shape, as the one in Figure 3.4, is to be a section of a circle then we will have to see if the region around it fulfills the condition for being one-that is to be of the same color as the blob and outside which, is to be a part of the background

To solve this problem, begin from the center point of the blob and let rays of pixels point in different angles and for each, calculate what pixel the end of the ray should be, if the length of the ray is equal to the radius of the circle. If enough rays coincide with the "correct point", and further, if there are not any points belonging to the blob, then there is a good chance that these points are on the circle; see Figure 3.4.

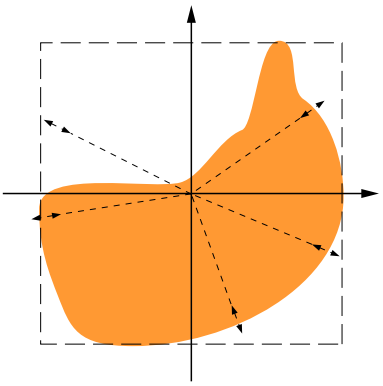


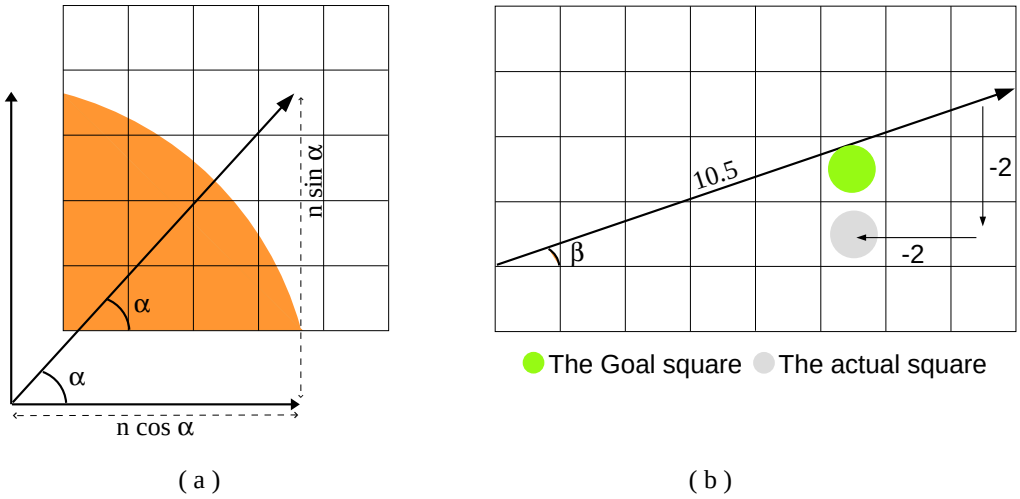
Figure 3.4: The bounding box is marked with a dashed square. The rays are marked with dashed lines, which are extended by double headed arrows, showing the range within which the edge of the blob, in color, will be looked for. In this particular case 4 out of 5 will succeed. The blob is obviously not a circle, which using more rays on the upper half of the bounding box will hopefully reveal

Let us divide the length of the bounding box by 5, 10 or 20-in proportion to the length of its side. This gives the number of rays, in whose directions we are going to

[1] See [11], [13], and [46] for definitions and explanations

look at the borders of the circle. If the width of each quadrant is $\frac{width}{2}$, there is a need for $n = 2(\frac{width}{5n})$, rays to be looked at, where n is 1, 2, or 4. Dividing 360 by the number of rays gives the angle between each couple of consecutive rays.

In each direction, depending on how long this ray is, divide the line by the number of pixels. Suppose we call the number of pixels on a ray n. We say that if the border of the blob goes through the pixels $n + l\gamma$ to $n - l\gamma$, where $0 < \gamma < 1$ and l is the length of the bounding box, for enough rays, then the blob *is* a circle. And now the question is: Should we not adjust the straight line to the grid? The answer is *yes*. It depends on what angle the line (or ray) has and also on the size of the bounding box.



(a) (b)

Figure 3.5: What square the ray will end in depends on the length of the ray itself, but also on the angle of it. Different angles can change the length of the ray, as much as ± 1 square

Consider Figure 3.5a. A ray in the direction α with the length n should end up in the square ($n\ cos\alpha$, $n\ sin\ \alpha$), so it suffices to look at the squares $B = (n\ cos\ \alpha \pm 2)$, if $l = 20$ and $\gamma = 0.1$. But the problem here is that B coincides with a square that is passed through by the ray, but in Figure 3.5b, we notice that decreasing x and y by 2 does not give us the correct square. It is therefore important to do the decreasing of the length of the ray, before we calculate x and y on the grid map

The advantage of this procedure is that it is easily implemented and even for a ball, covering the whole height of the image, it would take only between $\frac{360\times60}{20} = 1440[pixel]$ and $\frac{360\times60}{5} = 5760[pixel]$ to check.

Hence, between 1440 and 5760 pixels will be investigated. This means that for a circle covering almost the whole image, we will only need an overhead of some trigonometrical calculations plus color IDs of 1872 pixels to be compared with the correct color. This is a huge reduction in terms of number of pixels visited, compared to edge-detection algorithms, such as Hough Transform, named in the previous section.

The reason for using 5, 10 or 20 is that division by these numbers does not give that many decimals as it would with other numbers. In this regard, 2 and 4 could also be used, but then the number of pixels to check will increase without having any positive effects. Also, dividing by 5, 10 or 20 seemed reasonably enough when several different grid maps were drawn and tested.

Note that, if we are going to use the same method for balls that are hiding or occluded, then we will need to modify it as to take into account how large the section we are going to look at is; see Figure 3.6.

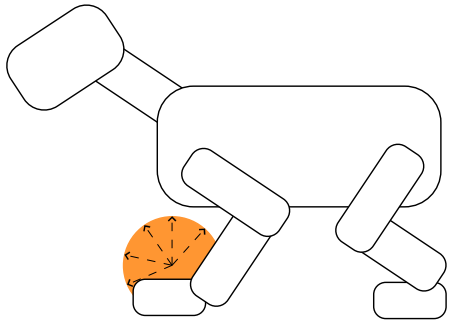


Figure 3.6: A ball occluded by a dog. The same method as for a totally visible ball is not applicable here

Displacement of the Center[2]

A problem is that the blob could be noisy, on any side of the image, as it is in Figure 3.7a. Then the center of the image is not what we calculate from $\frac{majoraxis+minoraxis}{2}$. But since the sizes of the bounding box are already checked, we can probably take the risk to miscalculate the position of the center of the ball by a few pixels.

Figure 3.7 shows that in 3 out of 4 possible cases, the calculated origin of the circle (or blob) is not situated in the same square as the real center. What makes it unnecessary to count on, is that the difference of the length of the ray, with or without these cases taken into account, is just the length of 1 pixel in any direction. By taking this into consideration, when choosing the length tolerance of the ray, the effect of the error will be eliminated.

Hence, the suggested solution would be to choose the function floor ($\lfloor . \rfloor$) of the division for the length of the ray. It is also important to do the rounding after the calculations are finished in order to eliminate the risk of increasing the errors related to it.

Compensation Attribute

An attribute, $compensation = 1$, was added, as the name suggests, to compensate for moving the center of the region to "one step left and one step up" in the grid map, according to Figure 3.7.

Length Tolerance

As we have seen earlier, for a 25×25-grid, a length tolerance of 2 pixels would suffice. The maximum length of the circle's radius for this type of AIBO is 80 pixels, where the tolerance can be about 5 pixels. An approximation of the correct tolerance could be

[2]This was unnecessary to proceed with, since the whole method of using rays was totally discarded

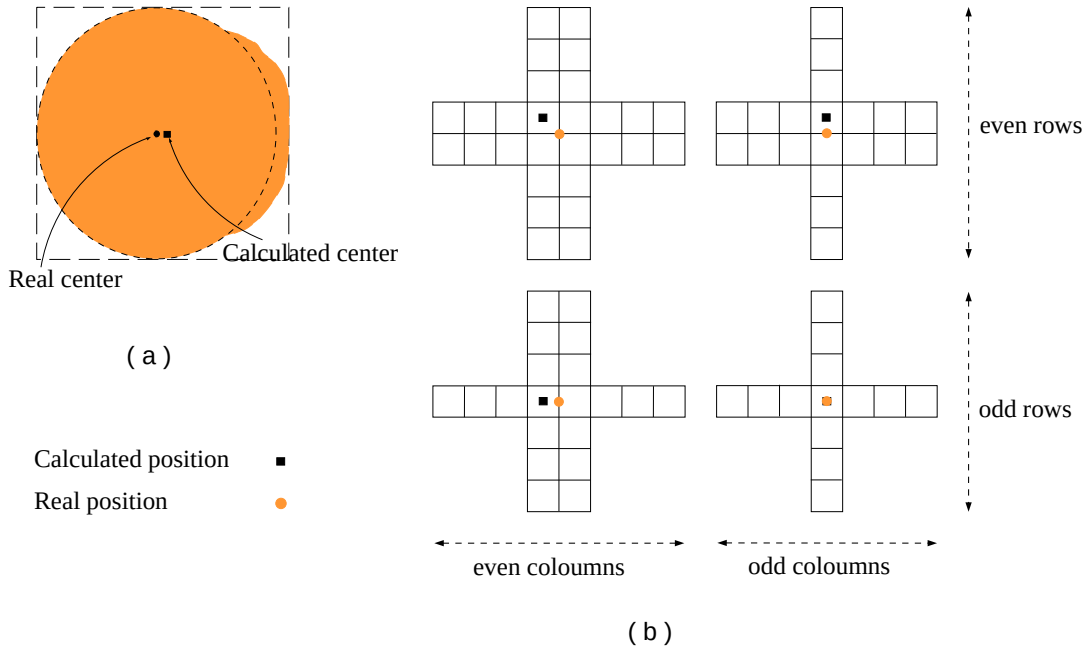


Figure 3.7: a)The center of the image is not the same as the center of the bounding box b)In three out of four possible cases the grid map show another center for the ball than the real center point

1×1	to	40×40	$\rightarrow 2$
41×41	to	50×50	$\rightarrow 3$
51×51	to	80×80	$\rightarrow 4$

Beyond the Desired Ray

Another thing to think of is to let the algorithm check whether there is going to be need of looking at the pixels, beyond the chosen rays. Because, as can be seen in Figure 3.8, it would be sufficient to make this control for the pixels from the angle $30°$ to $60°$, as marked in the same figure. This would make the solution even more efficient as we save time for comparing further pixels. The length l after a $30°$ turn will be reduced to $cos30° = \frac{\sqrt{3}}{2}l$. This means that for a 40×40-grid, the desired length of the ray is 20 and the $30°$ turn will reduce its image on the axis to 17.32 pixels.

Also, as can be seen in Figure 3.9 as long as the number of squares are less than 4, there is no need for such a test at all.

Digitizing the Rays

Going from 0 to x needs the *Digital Differential Algorithm*, DDA [1], to let each ray be *digitized* as accurate as possible.

Suppose we have a line segment defined by the endpoint (x_1, y_1) and (x_2, y_2), as in Figure 3.11a. These values are rounded to have integer values, so the line segment starts and ends at known pixels.

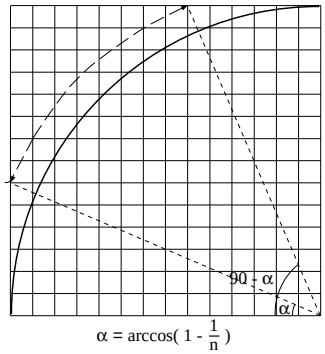


Figure 3.8: Only at the end of the rays with the angles $30°$ to $60°$, there may be pixels that could have other colors than the color of the ball, if it is a perfect circle

The inclination of the line is given by

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}$$

We assume that $0 \leq m \leq 1$ and for the rest of the circle we can use the same algorithm based on its symmetrical shape.

While traversing from (x_1, y_1) to (x_2, y_2) for any change in x equal to Δx, the corresponding changes in y must be $\Delta y = m\Delta x$.

As we move from x_1 to x_2, we increase x by 1 in each iteration. Thus, we must increase y by $\Delta y = m$. Rounding the value of $y + \Delta y = y + m$ together with x will give us the correct coordinate of the point to look at in the direction of ray; see Figure 3.11. By counting the number of pixels, out of those pixels checked by the routine, that are orange we can decide if a blob *is* a circle. This can also be used for the "shape criterion" in section 3.10.

Comparison Between Handling the Pixels for Alpha and Beta

We know that if $\beta = 90 - \alpha$, $sin\beta = cos\alpha$ and $cos\beta = sin\alpha$. This means yet another reduction in our calculations, since in each quadrant, half of the rays will have $sinus$-values that are equal to the $cosinus$-values of the other half.

Length of Ray

We could sum up the exact distances from all pixels on the perimeter of the circle being checked to the "center of region" and then divide them by the number of those pixels that are correct, meaning the pixels that are within the blob. This will give us an even better approximation of the radius of the ball.

At last, the total length of the projections of the rays on x and y axis, *totalLengthX* and *totalLengthY* respectively, are calculated separately and used to get an approximation of the length of the ray

$$\frac{\sqrt{totallengthX^2 + totallengthY^2}}{nbrCorrectRays}$$

This would be necessary, specially if there was no bounding box to decide the length of the diameter, which is of course not the case in our system.

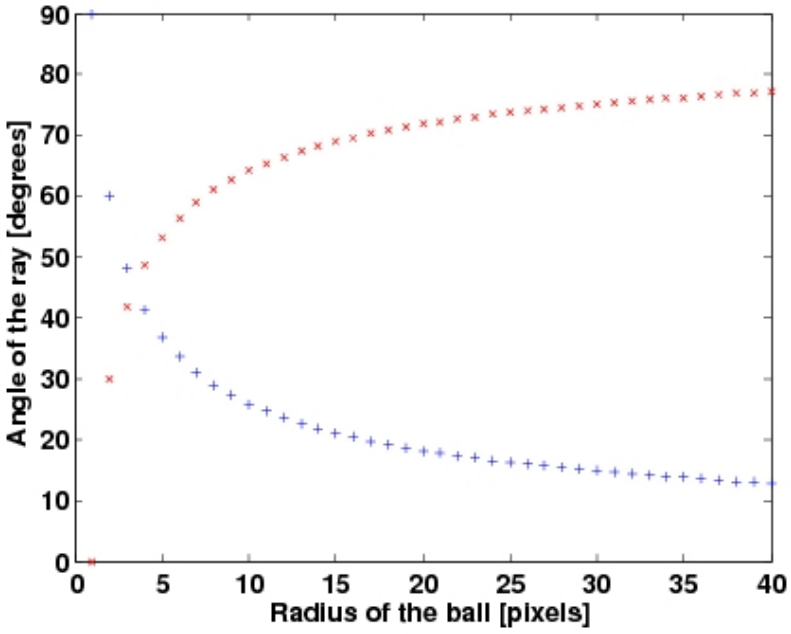


Figure 3.9: As is clear from the graph, if the length of the radius of a circle is at least 4 pixels, and up to 40 pixels, then we will need to check whether there are pixels with the same color as the circle. **+** shows the least and **x** the largest angle that a ray needs to have in order to be looked at

The Confidence

At the end of the session of "finding the ball", we can use the percentage $\frac{nbrCorrectRays}{nbrRays}$ also as a value of *confidence* for estimating the possibility of the property of being a ball or not.

Removing the Redundancy

There is still some more reductions to do, which would be to consider all the quadrants as if they are one and the same and use the same calculations with the values x+1 and y-1 replaced in each one as in Figure 3.12. But we rather use redundant code than use multiplications, and if- and while-statements, in order to make the procedure less time- and memory-demanding.

If the Ball Is Occluded

The ball can become occluded in a way that a part of it can not be visible in the image frame. The reason why this would occur is

1. The object covering it partly, is either another robot or
2. the situation is that the ball is inside the goal and we are looking at it from behind one of the net's side poles/walls (goal edges)
3. It is heavily shadowed by a robot

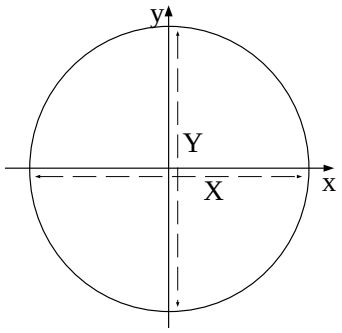


Figure 3.10: 1.8

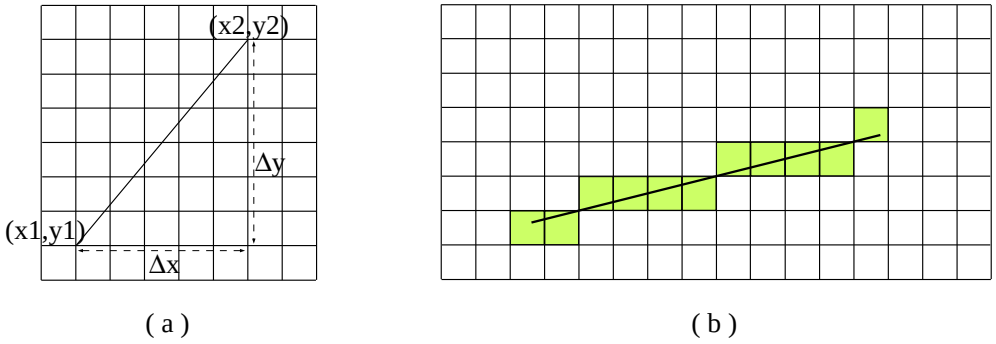


Figure 3.11: a)Inclination of an array. b)A digitized array

4. The lighting condition has changed.

In this case we will just need to add some tests to discard those rays that lie outside the image frame, as in Figure 3.13.

As can be seen in the figures and as been discussed, those parts that lie outside are automatically discarded. Besides, those rays that end inside the frame, are checked for and accepted, if the interval [rayMin, rayMax] is totally inside the frame, for that particular direction.

Finding the Radius

As an aid to estimating the distance to a ball, in cases where the ball is occluded, the following method was added: Count the number of ball pixels on all four edges of its bounding box. If on the horizontal line, the difference between the number of pixels are larger, then take the horizontal side for calculating the distance and vice versa; Figure 3.14. Note specially that in the case, where all the corners of the bounding box are free, the object still can be occluded.

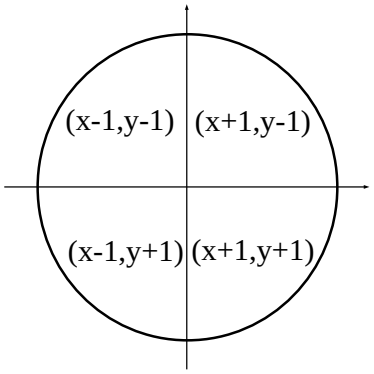


Figure 3.12: In each quadrant the calculations are all the same, other than the change that occurs in the (x,y) pairs

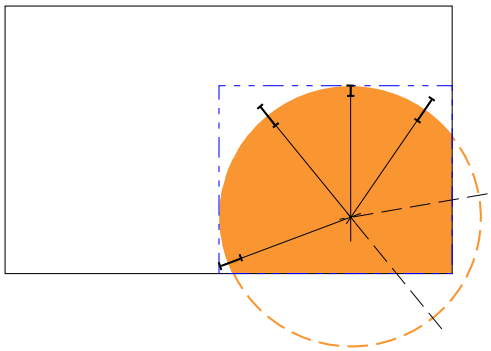


Figure 3.13: Some rays are to be discarded since they do not serve any purpose, particularly those that totally or partly lie outside the image frame

3.2.3 Facing "Close Range" - Considering another Approach

Still, choosing a point outside the image frame would cause us lots of trouble-looking at and testing points that probably are not within the frame wastes too much time. Besides, this is a long way to go and there are, as already noticed, too many factors to consider. Another approach was thus to be looked for. And it should work for all three cases, *long*, *close* and *very close* range, without any extra effort.

Consider Figure 3.15 for a complete covering of all cases that can occur when "ball pixels" fill a bounding box. There could be cases where the corners A and C or B and D are covered by the blob but those need not be considered, since that would presume the ball to be either non-circular or covered by two obstacles at the same time. Both these assumptions are rejected for the reason that the ball does not move fast enough to change shape in the images and that there are not obstacles on the field that would be able to cause a partial covering of the ball in this sense.

Compared to the case from UNSW [9], where the whole image was searched, the reduction to looking only within the blob, makes the task easier, since the computation effort is substantially less.

Of course, the more a ball is occluded the harder it becomes to estimate its center.

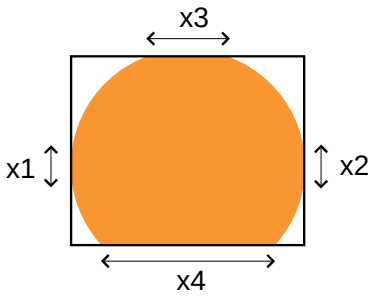


Figure 3.14: The difference between the number of orange pixels on the horizontal sides are larger than on the vertical sides. The horizontal side is thus used for calculating the distance

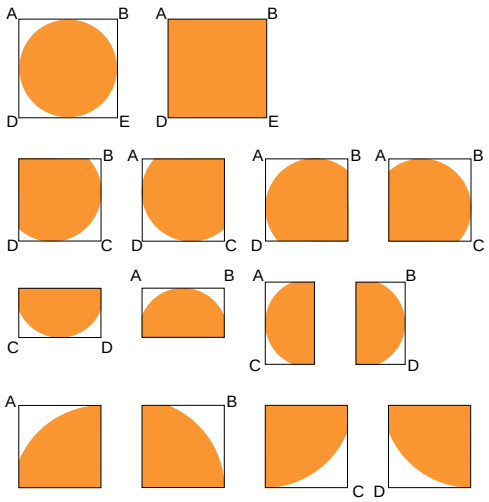


Figure 3.15: All possible cases of a ball-blob in a bounding box

But this is all we have to play with.

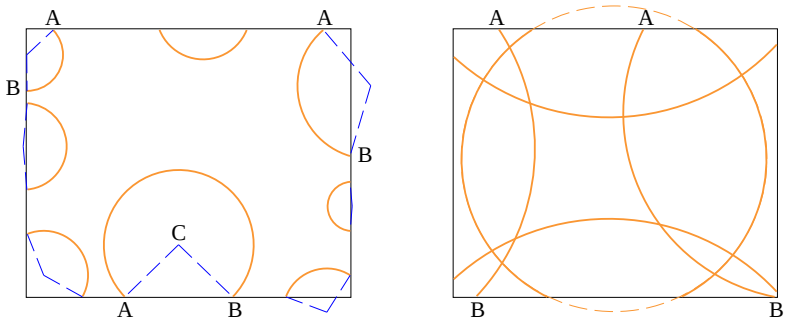
A major drawback-from time complexity point of view-would seem to be the risk for the cases where the lines can be horizontal and their k_{MA}^{-1} become infinitely large. But in this algorithm, that case is also easily and clearly taken care of; section 4.3.1.

As the algorithm looks mainly the same for all ranges, we go directly to the close range. In Figure 3.17 all possible cases that may occur, when a ball is not imaged as a whole circle, are illustrated.

If the circle is just partly inside the frame, as in Figure 3.16 then the procedure will be as in section 3.2.3 but first consider the following two cases

- In the close range we can always find the three points A, B and C, by running a procedure of traversing the edge points of the image frame, until we find the first orange point on the ball. The team of UNSW seems to have found a slightly better approach for this step of the method

  > As we traverse the boundary of the border, we keep track of the places where non ball colors and ball colors switch, if we come upon into a large consecutive run of ball colors then we go back to the last



(a) Close Range (b) Very Close Range

Figure 3.16: The ball is only partly visible.

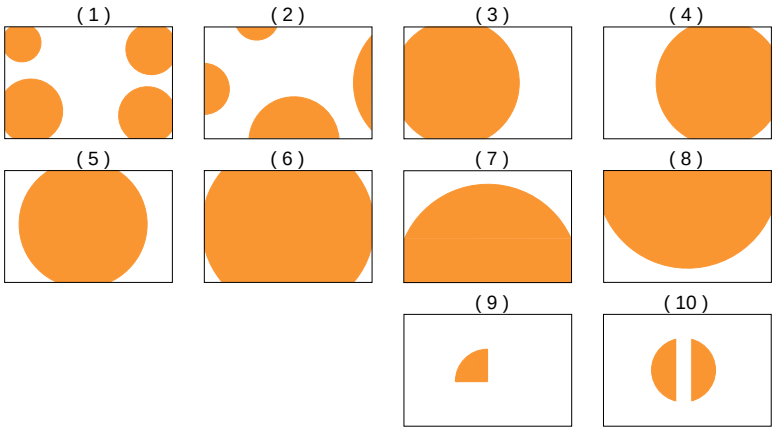


Figure 3.17: We found out that, the case (2) is just a special case of the case (1). This means that we will always have to take into account the possibility that the corners are open. Hence, for the 10 possible cases, except 2, 7, 8 and 9 we will have to introduce four variables, A, B, C and D, explained later in this section

> place where non-ball and ball colors switched and label that as the entry point into the ball region. All switching points that are not followed by consecutive run of the switched color are discarded as noise."[9]

But the reason why we need not follow this approach is that:

1. We already have removed blobs that are too small to be counted on
2. If a ball is divided into two, still adjacent parts (as with the beacon in Figure 3.23), and as a consequence of 1, not only there will be no need of discarding blobs, but also there will be a danger in doing so

- There is an easier way. Since the image is segmented into different colors and we have access to the maximum respective minimum of each blob in both the vertical and horizontal directions, we can always look at these points and choose them as the points A, B and C. This would be a reasonably fast process. Considering the images gives though another impression.

  As illustrated in Figure 3.16, the points A and B are the desired points. But they

are, however, not the extreme points of the blob, lying on the cross section of the boundary and the image frame. Hence, it is not equally simple to find the points, as it seemed to be in the beginning. We will have to choose the first strategy of iterating the points on the image frame. It is done much easier than the solution of the [9] though. Since the image does have boundaries, we will only need to begin from the boundary points.

For a circle that is heavily disturbed by, for instance, the image frame the boundary could be divided into several intervals and more points could be found on the perimeter-as in Figure 3.18.

Begin from the boundary that will be divided by 5, 10 or 20 on each side, not being on the frame edges. First point is the point on the edge and the last one on the corner where the two sides meet, if it is not already one of the edges. Then rays are sent toward the frame edges, till they meet points on the edge of the blob. All these points are saved in an array with the length, let us say n. Then points are coupled, two and two, and the PBS[3]s are calculated, M_1 and $M_{\lceil \frac{n}{2}+1 \rceil}$, M_2 and $M_{\lceil \frac{n}{2}+2 \rceil}$, ..., M_m and $M_{\lceil \frac{n}{2}+m \rceil}$, ...

This proved to be successful in spite of there being lots of noise in the image, resulting in faulty estimation of the three points.

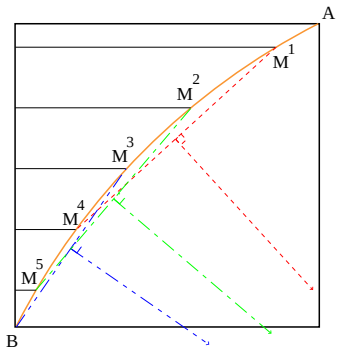


Figure 3.18: Using more than only two points for ball curves

The drawback is that this would prolong the execution, and perhaps not give any perfect result either, since in a very close range many lines will be almost parallel and give the same result.

Very Close Range

In the case Very Close Range-cases 3 through 6, plus 10, in Figure 3.17-we divide the problem at first into finding M for the two curves and then, when it comes to estimating the center of the circle, we use all the four points; see Figure 3.19. Nevertheless, the range of the ball is not deterministic, as noticed. What we use for estimating the size of the circle ray is only how many corners in the bounding box that are free.

[3] **P**erpendicular **Bis**ectors

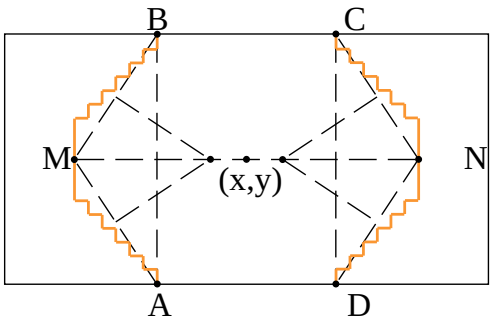


Figure 3.19: Searching for the center point of a circle, corresponding to the case (5) from Figure 3.17. The points are all marked.

3.2.4 Neighboring Colors of a Ball

Under a ball, in the image, there always have to be dark-blue, carpet, green, or white. If we are looking at the ball over the back of another robot, then other colors are to be expected too, such as red, robot blue, gray or black.

3.3 Finding the Nets

3.3.1 Shape of the Net

First case is if the goal is not occluded. Then the shape of the goal, on a long range, is rectangular. We will only need to clarify if the object has a rectangular shape, i.e., four straight lines with consequent lines perpendicular to each other. We could probably use the same procedure as for a partly visible ball, i.e., draw some lines, both vertical and horizontal, within the bounding box for the blob to find some points on its edges; Figure 3.18. There are two methods to use: 1)Finding points in a regular mesh, and 2)An algorithm for following the perimeter-both methods described in section 4.7. The second method can not be used for the *circle recognition*, because there are no straight lines in the perimeter of a circle, which makes the task a waste of time, bearing in mind that methods that could use all the points-such as the Hough Transform-are already rejected as solutions, by reason of their processing times.

The first method is therefore to use in the same manner as is done for ball recognition. Different from that procedure will be to find out if straight lines connect these points together. This can be done using Hough Transform of the points' coordinates[4].

Then, we will only need to control whether these lines are perpendicular. This is however too risky, since a net, seen not from the middle of the field *and* on a close range, hardly looks like a perfect rectangle with the sides showing all the characteristics of it; see Figure 3.20. Consequently the color-check was used to avoid complicating the task of *net recognition*.

[4]See [11], [13], and [46]

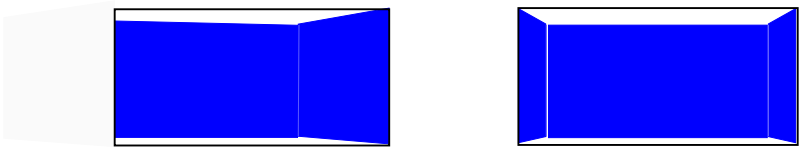


Figure 3.20: The nets viewed by the AIBO from two different positions, a) Standing close to the left beacon and b) standing in the middle of the field

3.3.2 Angled Image

Even if the head is panned to the left or right we will not have any particular problem recognizing the rectangles.

What we would need here to do is to look at the distances of the points found on the supposedly rectangular shape to the nearest side of the bounding box. Even if the shape is rolled, we can guess how long from the edges of the bounding box each point should be, using simple Trigonometry. But this method was discarded later on, because the head movements were simplified to be done in simple left-right and up-down directions; see 3.8 on how it otherwise is handled.

3.3.3 Merging Net Blobs

Sometimes, not only for the nets, but also for the other obstacles on the field, lighting condition or poor segmentation can make their images divided into several separated subparts. Therefore, an algorithm for merging these parts together could save us from false estimations and hence, seeing objects that are spread just anywhere in the field but their real places.

For merging two blue blobs, we can calculate the distances between the sides of the two. If there are two sides of these that are close enough, we can conclude that these actually are parts of a larger net. For an algorithm, see section C.3.

3.3.4 Marking the Nets

It is a fact that we need to find the yellow net before we look for the blue one. But because of problems with the segmentation, we could have a dark blue carpet. It was therefore necessary to recognize the blue net in the image, if any, to make sure that a blue part of the carpet would not be recognized as a blue net (BNET), by checking that no yellow net (YNET) is positioned above the blue blob - that we assume to be the BNET-as illustrated in Figure 3.21.

To see whether they coincide horizontally, we can simply check if the intervals between the vertical margins of each of BNET and YNET are overlapping in the horizontal direction.

3.3.5 Partial Visibility

If the rectangle is just partly visible, the 8 different cases in Figure 3.22, except case 5, may occur:

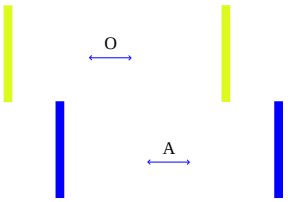


Figure 3.21: A possible case, when the yellow net seems to be over the blue one. A rule base, as discussed in C.2.1 or another procedure, could take care of such cases

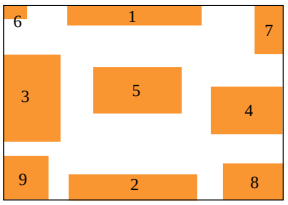


Figure 3.22: The 9 different poses a rectangle may be seen in

In cases 1 and 2, the width is used for the calculation of the distance, which in (4.9) gives the value

$$d = \frac{104}{tan(\frac{56.9}{2})} \times \frac{RealWidth}{ImageWidth},$$

where the RealWidth is the width of the object in mm and ImageWidth the width of its image in pixels.

In the cases 3 and 4, on the other hand, the width is not completely visible, so the height is used for the estimation

$$d = \frac{80}{tan(\frac{45.2}{2})} \times \frac{RealHeight}{ImageHeight},$$

where the Real Height is the height of the object in mm and ImageHeight the height of its image in pixels.

In the case 5, it seems at the first sight that either of the height and/or width could be used for this purpose, but there are two potential problems introduced here, specially for the blue net

- For the same reason as for the beacons and the ball-that is, the length can be divided by rays of light and make it futile-we could prefer to use the height of the image; Figure 3.23.
- Changes in lighting conditions can produce shadows in the image, where some parts of the field, adjacent to the net, can turn into blue. This results in lengthening the height of the net, which makes the width the one to prefer. This problem can not occur for the yellow net.

The final decision was nevertheless to use either the height-the choice of the Kyushu United Team ([30])-because

1. The second potential problem above, happens only for the blue net, which is the "own" net and not as crucial as the yellow one, although necessary for the localization

2. In case the AIBO looked at a net from one side of the field, then it could not see the whole width of it, which would result in estimating the distance to be much more than it actually was

or a smart combination of the two, depending on the relation between the sizes of the bounding box. Here we could have a much better control over the situation, if Localization could provide Recognition with information about where on the field the player is supposed to be in a particular frame. See the section C.2.1 for more discussion on this matter.

Whether this decision is going to lead to a successful estimation of the distances, depends, of course, highly on how well the color segmentation succeeds.

Finally, the cases 6 through 9 show those situations, where it is almost impossible to guess how far the net is unless we use the same procedure discussed in 3.9 (Imagine, for example, that a large part of a net on a long range can have exactly the same size in the image as a little part of it in a short range.)

Figure 3.23: There is a risk that the objects can be recognized easier in the vertical direction than in the horizontal. The reason is that the projectors point down from the ceiling, resulting in rays of light dividing the blob along the horizontal axis. This also shows clearly the necessity of implementing a method for merging neighboring blobs with similar colors. For a solution, care for section C.3

3.3.6 Correct Distance To a Net

What the AIBO can see of the net depends on from what angle it looks at it. To find the distance to the middle part of the net, for a better localization, a reduction of the distance with its depth is necessary; Figure 3.24.

This was unlike what the Kyushu United [30] and The University of Texas Austin Villa [42] did, where the goal post was used for the distance, because " The goal edges were much more reliable as inputs to the localization module than were the goal centers"[42].

3.4 Finding a Robot

Robots, either the "teammates" or the opponent players, have complex shapes compared to other objects on the field, as can be viewed in figure 3.25. Depending on the pose of an AIBO, it can have different shapes. Hence, a recognition based on the shape of the

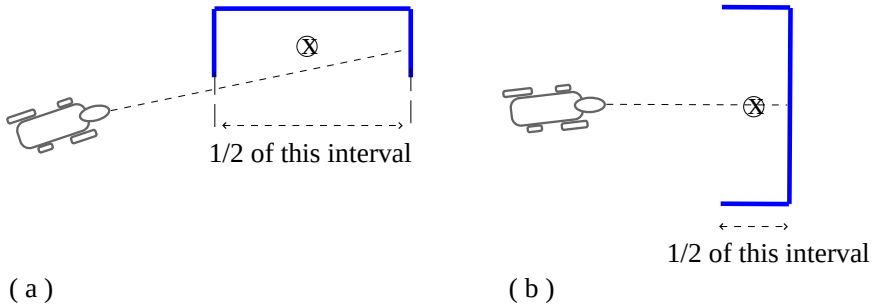


Figure 3.24: a)The net is seen from one side. Distance to the net is reduced with half of its width b)AIBO is standing in front of the goal. Distance to the goal is now reduced only by half of its depth

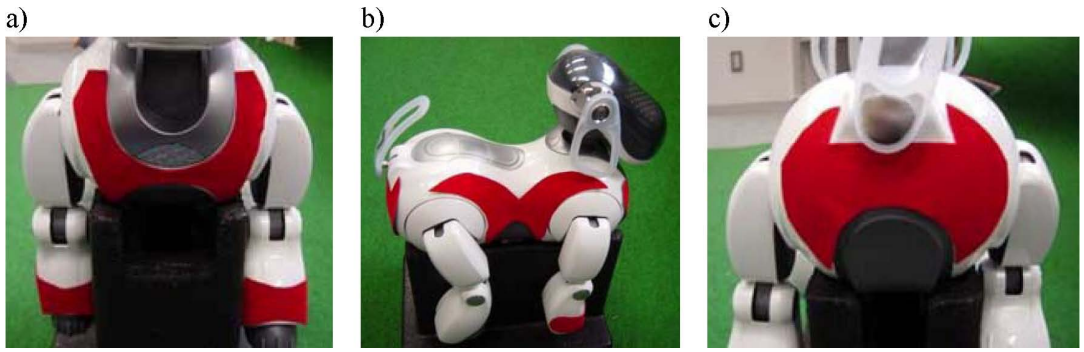


Figure 3.25: ERS-7 team markers. The images illustrate, from left to right, the front, side, and back view

whole object is not possible, or simply computationally too expensive. Even recognition of the shapes of the patches considering their special configurations is even more difficult. On the other hand, its distinctive RED color is recognizable easily compared to the other colors on the field.

What comes to other teams, there were no special strategies reported in this regard. Outcome of the studies of other works and preformed tests in this regard did not give rise to any ideas other than the heuristic to use the color as the only way for recognizing robots.

If a blob is segmented as red, it suffices to test whether it is standing on the carpet or if there is another color than CARPET or DARK BLUE below it. It is otherwise treated as a yellow net or a beacon. This simple check seemed to work well. But for an even better result, we should use a merging algorithm; see section C.3 for more discussion.

3.5 Recognizing Non-Predetermined Objects

Because of all the shortcomings in the recognition module, the recognition of all other unknown objects were also based solely on the colors and not their shapes or any other characteristics.

3.6 Angle and Distance To Objects

With the properties of an AIBO's imaging device it can be considered almost as a pin-hole camera. Since we know how large the ball, the goal, the dogs, and the beacons are, we can estimate the distances to these objects with good accuracy, using the view angle from section 3.2.3 to the respective object.

For this, we will only need to calculate both its distance to the center line, d_r, and its perpendicular distance to the camera, b. Consider Figure 3.26a.

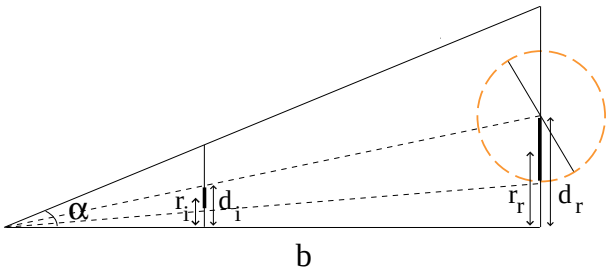


Figure 3.26: Distance and angle to an object can be calculated using a comparison between the image itself and its image on the image plane

At the first sight, it seems impossible to calculate d_r, because there is no information that can support our calculation of b. For finding d_r just imagine that there is an object whose width is d_r and we are trying to find this width. But there is nothing to compare the width with. Instead, we can compare the size of the ball in reality with its size in pixels. Then we apply the same relation on its distance to the center of the image.

If an object is heavily occluded, still some of its visible properties can be used for determining its distance from the camera. For example, with a net whose whole length is not seen we can use the height for estimating its distance. This part is thoroughly discussed for each object in its own section.

3.7 Median Filter

It is obvious that miscalculations of distances to different objects will occur every now and then, due to different causes, explained in respective sections - see section 4.4.2 for an example. To prevent undesirable reactions from the AIBO, a median filter was used.

Take into account the history of the distances of the ball and take the median of the last 10. In order to skip moving the distances in the array, use an index.

In the square that $index$ is pointing at, as in Figure 3.27, the new value can be set, estimated in the latest frame and then move the index to the next one, which naturally is the oldest input value, if the same procedure is followed all the time. Take the median of all ten and compare to the already computed and used distance earned from the last frame. This will give the new distance if it is not farther away than the distance the robot actually can move, between two frames. For better estimation of the frame frequency and velocity of the dog, World State-see Chapter 1, Introduction-should be a good source to look up in.

By calculating the median value of a neighborhood rather than the mean value, two main advantages are gained over the mean filter:

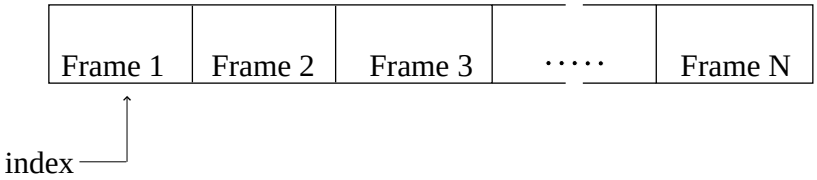


Figure 3.27: A median filter with N frames. Index shows where the next value is to be put in

- Median is less susceptible to declining values. The median is a more robust representative of changing values in this case, than the mean and so a single, very unrepresentative distance in a neighborhood will not affect the median value significantly.
- Since the median value must actually be one of the values in the neighborhood, the median filter does not create new unrealistic values.

3.8 Turning the Image

Images can be rotated, depending on the rotation of the head. This will not affect the recognition of the ball, since it is circular and hence symmetrical. But almost all other objects are affected.

Hence, we should be careful when the head is rolled to one side, because all quadratic shapes will do the same[5].

Figure 3.28 is an illustration of the possible translations and rotations of the camera made by the movements of the head-joints. This could lead to images, of objects on the field, that are not as simple as one usually senses when looking at the field from a normal upright position.

The degree of freedom of the head is said to be 3, as can be viewed in Figure 3.28; also refer to A.1. What is needed to be done to find the transform matrix for the image taken in the camera coordinates, relative to the coordinates of the neck-joint, is to look at what movements it can perform.

Let us define this transformation with the help of the 4 following coordinate systems

1. The CS_{camera}: the camera is an immobile part of the head. Its coordinate system, in what the image is defined, is 3-dimensional and described by
   - its origin at the shutter of the camera
   - the positive y, pointing down, perpendicular to the head,
   - the positive x, pointing to the right, seen from the middle point of the AIBO's body,
   - and the positive z, pointing out from the camera and in the direction of the head; this axis has actually no physical interpretation, but is necessary for the coming calculations,

[5]Close to end of the of the project, it was decided to constrain the head movements not to include the *tilt*. All code regarding this section was therefore removed.

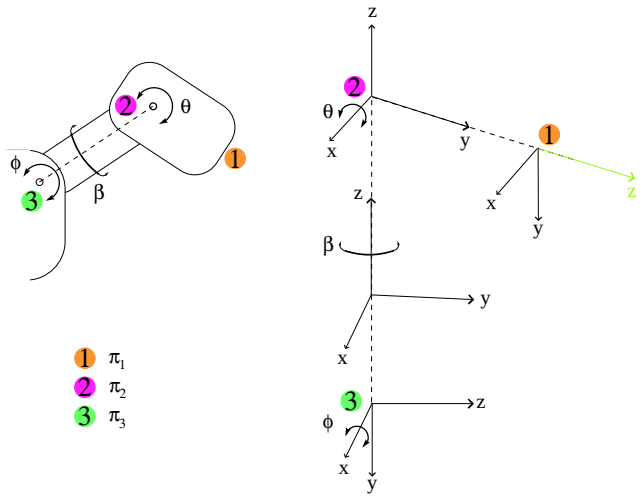


Figure 3.28: Those joints that are used to enable the head movements and the corresponding coordinate systems: 1)CS_{camera}, 2)CS_{head} and CS_{spin} (for the sake of being perspicuous, the latter one is moved a little bit down) and 3)CS_{neck}

2. The CS_{head} is in the same way described by
   - its origin at the joint that connects the head to the neck,
   - the positive y, pointing forward, in the direction of the head,
   - the positive x, pointing to the right, seen from the middle of the AIBO's body,
   - and the positive z, pointing up, in the direction of the neck.
3. The CS_{spin} is described by
   - its origin on the axis of spin of the neck-as will be shown, the vertical position of this point is not a determinative factor-
   - the positive y, pointing forward and out from the body,
   - the positive x, pointing to the right, seen from the middle of the AIBO's body,
   - and the positive z, pointing upward in the direction of the neck and perpendicular to the ground.
4. Finally, the CS_{neck} is described by
   - its origin at the joint that connects the neck to the body,
   - the positive y, pointing down, perpendicular to the ground,
   - the positive x, pointing to the right, seen from the middle of the body
   - and the positive z, pointing forward and out from the body.

Using the definitions above, three transformation matrices can be defined:

1. A 90-degrees, counterclockwise turn in the yz-plane of the CS_{head} and a translation, as much as the length of the head (L_h), in the negative direction along the z-axis, will transform the CS_{camera} to the CS_{head}. Taking into account that the camera may make additional turn with the angle θ around the x axis of the latter system, the transformation matrix is defined by[6]

$$\begin{aligned} T_{head} &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\,(90-\theta) & sin\,(90-\theta) & L_h\,sin\,(90-\theta) \\ 0 & -sin\,(90-\theta) & cos\,(90-\theta) & L_h\,cos\,(90-\theta) \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & sin\,\theta & cos\,\theta & L_h\,cos\,\theta \\ 0 & -cos\,\theta & sin\,\theta & L_h\,sin\,\theta \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{3.1}$$

2. The CS_{head} and the CS_{spin} are actually one and the same system, although the first system can spin around their common z-axis, in either direction, with the angle β, giving at hand the transform matrix

$$T_{spin} = \begin{pmatrix} cos\,\beta & sin\,\beta & 0 & 0 \\ -sin\,\beta & cos\,\beta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.2}$$

3. The CS_{spin} has to make a 90-degrees, clockwise turn around the x-axis of the CS_{neck} and then translate as long as the length of the neck, (L_n) in the negative direction of the y-axis, to be transformed to the CS_{neck}. Additionally, the neck may make a turn in the same plane with an angle of ϕ-degrees to generate the transformation matrix

$$\begin{aligned} T_{neck} &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\,(-90-\phi) & sin\,(-90-\phi) & -L_n\,cos\,\phi \\ 0 & -sin\,(-90-\phi) & cos\,(-90-\phi) & -L_n\,sin\,\phi \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -sin\,\phi & -cos\,\phi & -L_n\,cos\,\phi \\ 0 & cos\,\phi & -sin\,\phi & -L_n\,sin\,\phi \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{3.3}$$

We are now one step closer to our goal. A combination of these three matrices gives the coordinates of the image of any point relative to the CS_{neck}. Together with information about the position of the body and other information that can be extracted from these matrices, we can estimate the distance to an object, as will be shown in section 3.9.

To find the horizon in the image, we need only to find the angle between a horizontal line, in the CS_{camera} and a horizontal line in the CS_{neck}, if we suppose that the body is in the normal position, meaning that the xy-plane of this system is parallel to the field.

[6]For a discussion on how the image matrices are presented in consult [12] or any book on *Perspective Geometry*

For the sake of simplicity, suppose that the horizontal line passes through the points $A(x, 0, 0)$ and $B(-x, 0, 0)$. According to the rules of linear algebra, the coordinates of these points, in the CS_{neck}, are

$$M_{neck} = T_{neck} \times T_{spin} \times T_{head} \times \begin{pmatrix} x \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Proceeding from right to left, we will have

$$M_{head} = T_{head} \times \begin{pmatrix} x \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ L_h\ sin(90-\theta) \\ L_h\ cos(90-\theta) \\ 1 \end{pmatrix},$$

$$\begin{aligned} M_{spin} &= T_{spin} \times M_{head} \\ &= \begin{pmatrix} x\ cos\ \beta + L_h\ sin(90-\theta)\ sin\ \beta \\ -x\ cos\ \beta + L_h\ sin(90-\theta)\ cos\ \beta \\ L_h\ cos(90-\theta) \\ 1 \end{pmatrix} \end{aligned}$$

and

$$M_{neck} = T_{neck} \times M_{spin} =$$

$$\begin{pmatrix} x\ cos\ \beta + L_h\ sin(90-\theta)\ sin\ \beta \\ y_{M(spin)} \times\ cos(-90-\phi) + z_{M(spin)} \times \sin(90-\phi) - L_n\ cos\ \phi \\ z_{M(spin)} \times\ -sin(-90-\phi) + z_{M(spin)} \times \cos(-90-\phi) - L_n\ sin\ \phi \\ 1 \end{pmatrix},$$

where

$$y_{M_{spin}} = -x\ cos\ \beta + L_h\ sin(90-\theta)\ cos\ \beta$$

and

$$z_{M_{spin}} = L_h\ cos(90-\theta)$$

The same can be done for the negative x (x can be replaced with $-x$ in 3.4) to get the values

$$x_2 = x\ cos\ \beta + L_h\ sin(90-\theta)\ sin\ \beta$$

and

$$x_1 = -x\ cos\ \beta + L_h\ sin(90-\theta)\ sin\ \beta,$$

which means that the distance between A and B, in the horizontal direction is

$$\Delta x = x_2 - x_1 = -2x\ cos\ \beta$$

On the other hand

$$\Delta y = y_2 - y_1 = 2x\ sin\ \beta\ cos(-90 - \phi)$$

Hence, the inclination of the image of a horizontal line is

$$k = \frac{\Delta y}{\Delta x} = \frac{2x\ sin\ \beta\ cos(-90 - \phi)}{-2x\ cos\ \beta} = -cos(-90 - \phi)\ tan\ \beta$$

Knowing the fact that $-cos(-90 - \phi) = sin\ \phi$ we realize that the final equation for the inclination of a line in the image is

$$k = sin\ \phi \times tan\ \beta \tag{3.4}$$

Note that the inclination is independent of the angle θ. When we know how much the image is turned, we can decide whether an object seems to have a lower or higher position than another object. This has been discussed throughout the previous chapter, in subsection 3.3.2, and the comming section 3.9.

3.9 Using the Elevation

As mentioned previously, knowing the elevations of the blobs in an image, makes it easier to filter out those that are illogically positioned. Moreover, we can find whether an object is within a reasonable distance from the camera.

In order to know what object has the highest elevation we will have to turn the image upright. This is done with the help of the equations 3.1, 3.2 and 3.3, as in the previous section (3.8).

To find the distances to the objects in the image, still more needs to be done. The height of the camera from the ground, h and the angle β, in Figure 3.29, will help us with this task.

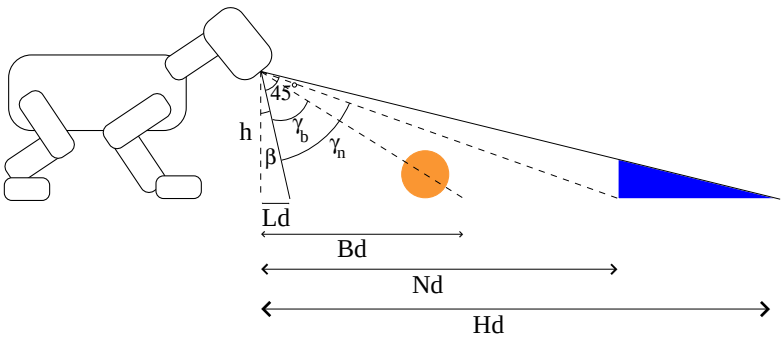


Figure 3.29: How to use the dimensions of the dog and the image frame to calculate distance to an object. Angles are supposed to be already calculated-as mentioned earlier in this section. Here, a completely visible ball and a partly visible BNET are viewed by the robot

Height of the camera, h, can be found in two different ways:

Joint Values: Joint angles are always available in the World State. Fetching these values and following the calculations below gives the height of the camera. Consider part (d) of Figure 3.30. The lower part of the legs lean $45°$ back. This makes the whole height of the leg

$$H_l = 69.5 \times cos(45) + r + x \quad = 69.5 \times \frac{\sqrt{2}}{2} + 9 + 33.15 = 91.3\ [mm],$$

where H_l is the height of the whole front leg. The distance between the shoulder-joint and the neck-joint is $19.5mm$. Added to the H_l we will have

$$H_n = 19.5 + 91.3 = 110.8\ mm,$$

where H_n is the height of the head-joint, which can be rounded to 111 without affecting the precision of the calculation that much. Next step is to find the height of the camera compared to the neck-axis.

Height of the head-joint relative to the height of the neck-joint can vary between 80 and $80 \times cos80 \approx 14.0mm$. Adding even this value to the H_n the total will lie somewhere in the interval $[111 + 14, 111 + 80] = [125, 191]$ in millimeters. Drawings of the AIBO in [44] show that last joint is at the same height as the IR, when the Head Tilt is zero. Hence, the height of the camera from the ground is $[125 - 14.6, 191 - 14.6] = [110.4, 176.4]$. Added to all these, can be the height of the camera-depending on how much the head is tilted.

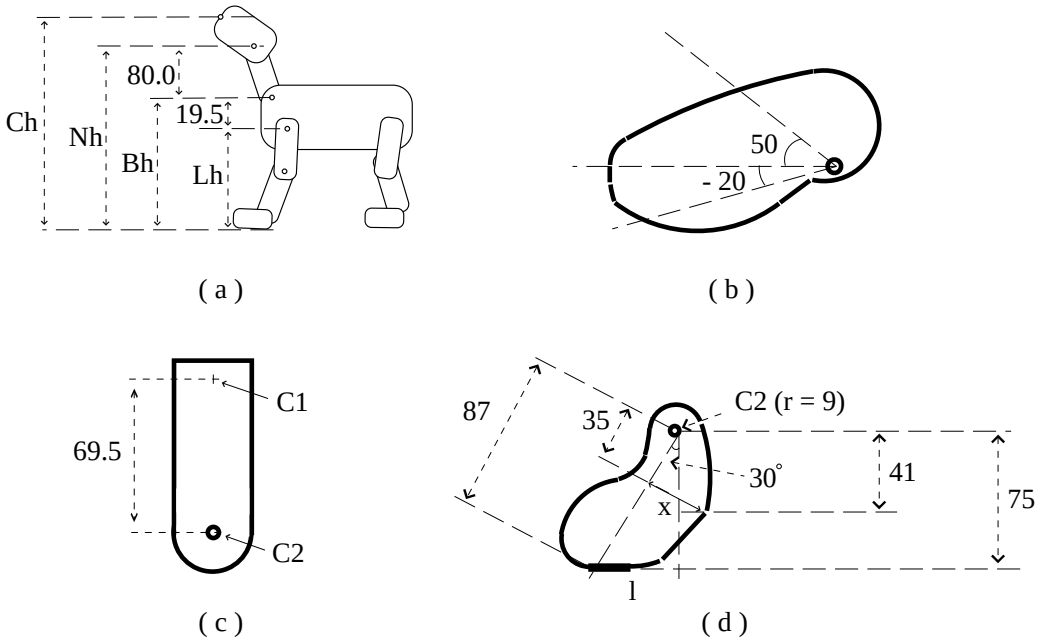


Figure 3.30: Those joints that are used to enable the head movements and the corresponding coordinate systems, used for the calculation of a matrix for mapping

Kinematics: Next alternative is to use the rules of the *Kinematics*[7] and the fact that the height of the camera from the ground is equal to the perpendicular distance

[7]Kinematics is the science of motion. In a two-joint robotic arm, given the angles of the joints, the

of the camera shutter to it, which has the coordinates (0, 0, 0) in the CS_{camera}. This means that the elevation of the camera is equal to the y-coordinate of the camera in the CS_{neck} added to the height of the leg, H_l, which is computed in the following way[8]

$$\begin{aligned}
\pi_1 &= T_{neck} \times T_{spin} \times T_{head} \times \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \\
&= T_{neck} \times T_{spin} \times \begin{pmatrix} 0 \\ L_h\, cos\, \theta \\ L_h\, sin\, \theta \\ 1 \end{pmatrix} \\
&= T_{neck} \times \begin{pmatrix} L_h\, sin\, \beta\, cos\, \theta \\ L_h\, cos\, \beta\, cos\, \theta \\ L_h\, sin\theta \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} L_h\, sin\, \beta cos\, \theta \\ -L_h sin\, \phi\, cos\, \beta\, cos\, \theta + L_h sin\, \theta\, cos\, \phi - L_n\, cos\, \phi \\ -L_h cos\, \phi\, cos\, \beta\, cos\, \theta - L_h sin\, \theta\, sin\, \phi - L_n\, sin\, \phi \\ 1 \end{pmatrix}
\end{aligned}$$

The same goes for point 2, π_2, with the difference that this point has an x-coordinate that is constantly 0. Therefore, only one multiplication is needed to get its coordinates in CS_{neck}, that is

$$\pi_2 = T_{neck} \times \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -L_n\, cos\, \phi \\ -L_h\, sin\, \phi \\ 1 \end{pmatrix}$$

The computations above, in addition to our knowledge about the height of the shoulder-joint, gives the value of h. What is needed now is the angle, with what the head is pointing to the ground.

As is illustrated in Figure 3.31, if the coordinates of the points 1 and 2 in CS_{neck} are found, then basic rules of trigonometry will give the desired angle.

Suppose that the coordinates of the camera (point 1) are (x_1, y_1, z_1) and the coordinates of the head-joint (point 2) $(0, y_2, z_2)$, both in the CS_{neck}. Then the distance between the projections of these points on the xz-plane is

$$\Delta xz = \sqrt{(x_1)^2 + (z_1 - z_2)^2}$$

kinematics' equations give the location of the tip of the arm. *Inverse kinematics* refers to the reverse process. Given a desired location for the tip of the robotic arm, what should the angles of the joints be so as to locate the tip of the arm at the desired location. There is usually more than one solution and can at times be a difficult problem to solve. [Demos -Modeling Inverse Kinematics in a Robotic Arm, retrieved from http://www.mathworks.com/products/fuzzylogic/demos.html]

[8]It seemed that the rules of the Kinematics and its predefined calculations are far more complicated than what is done here. Hence, an intuitive computation with the help of the definitions in *Linear Algebra* and *Perspective Geometry* were preferred.

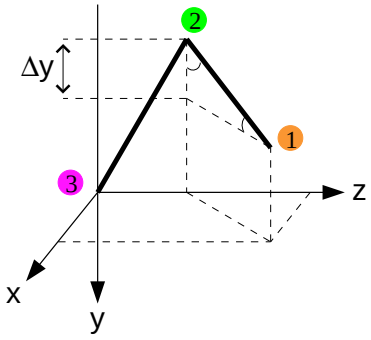


Figure 3.31: Calculating the position and angle of the camera

and the desired angle β, from Figure 3.29, is thus equal to

$$\beta = arctan(\frac{\Delta xz}{\Delta y}) - \frac{\gamma}{2}, \tag{3.5}$$

where γ is the real angle of the camera, in the vertical direction. This angle has been marked with $45°$ in Figure 3.29, for the case where the image is not rolled to either of the sides. As already known, the horizontal view angle of the camera is more than this value, and the more the head is rolled the more the view angle will be. This is thus to be taken into consideration in our calculations.

To hold the procedure on a simple level, we can use the relation

$$\rho = \frac{1}{2} \times (\frac{FrameHeight}{vertical\ ViewAngle} + \frac{FrameWidth}{horizontal\ ViewAngle}),$$

which will be multiplied by the vertical size of the image, in pixels. We saw earlier that the horizon is found from the equation (3.4). Length of the vertical line that goes through the image and is perpendicular to the horizon-labeled in Figure 3.32 as L_v-depends on its inclination, α. Note that if the horizon is inclined with an angle of less than $arctan(\frac{ImageWidth}{ImageHeight})$, the L_v is

$$L_v = \frac{ImageHeight}{cos\ \alpha}$$

pixels high and if it is more, then its length is equal to

$$L_v = \frac{ImageWidth}{cos(90 - \alpha)}$$

Its corresponding angle, γ is thus equal to

$$\gamma = \rho \times \frac{angle}{pixel} \times L_v$$

This is exactly what we need for the equation 3.5.

As an example of how this can help us find the distance to a partly visible net, discussed in section 3.3.5, first consider the section 3.3 to realize that the distance to a net, when the whole of neither the width of the net nor its height is presented in the

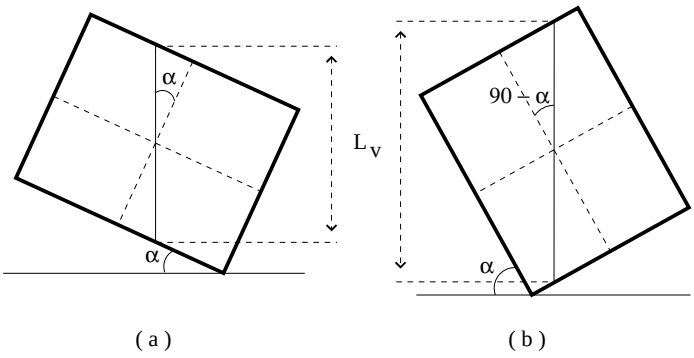


Figure 3.32: Calculation of the value of L_v is depended on whether it is greater or less than 90 degrees

image frame, is impossible. Now, by looking back to Figure 3.29, we notice that if the angle to the lower edge of the BNET is computed in the same way the L_v was, then the distance to the net Nd is

$$Nd = h \times tan(\beta + \gamma_n)$$

This is of course true as long as the lower part of the object is in the frame and $\beta \leq 45\,^\circ$. Nevertheless, this can at least be used to check whether the calculations seem to be correct or not.

If this method is to be used to find the elevation of objects, the constraint of, for instance, the blue blob to be above the green field, cannot be released, because the blue blob can be beyond the boundaries of the field. This method rather verifies whether the object is on the ground, if we think it is a goal or a ball, or whether it is suspended in the air, if we have guessed it to be a beacon.

3.10 A Voting System for the Confidence

The whole recognition can be organized as a voting program or weighted sum of credits from each part of the recognition process[9]. The idea is that the whole recognition should work as a "voting" system, or to express it mathematically: a weighted sum of the confidences earned from the functions defined below. This means, all blobs will be checked in the following functions:

1. How many pixels is it? Could it be noise?
   There is a minimum size for every known object. At the farthest distance, it is at least as large as *MinSize* pixels. The less (than MinSize) it is the lower the confidence. Even for the blobs that are too large, the confidence drops with growing size, until it covers the whole image. Figure 3.33 shows how the function may look like.

[9]As the irregularities in the color blobs were too much, particularly the items 2, 4, 6, 8, and 9 were malfunctioning and hence could not be used. Consequently the function is not implemented exactly as is mentioned above. Only the other remaining items were implemented and in use. Even the neural network was not implemented for the same reason.

Intuitively, the increase of the confidence can be described as

$$f = \frac{1}{e^{(MinSize-size)}}$$

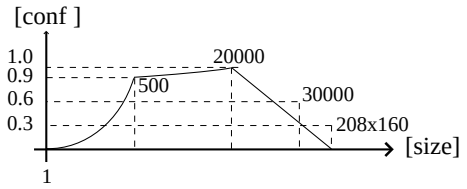


Figure 3.33: The exponential function that produces a confidence value for an object consisting of less pixels than it is allowed to have

We need the program to handle the computation of f in a reasonable amount of time. Knowing that the MinSize is usually between about 20 to 500 pixels for all known objects on the field, the exponent can be divided by 100. As can be considered in the following table this reduces the output value with an order of up to 10^{-172}.

BlobSize	**exp(MinSize - BlobSize)**	**exp((MinSize - BlobSize)/100)**
100	1.9E-174	0.02
200	5.2E-131	0.05
300	1.4E-87	0.14
400	3.7E-44	0.37

Table 3.1: An approximation of the parameters in the above equation

Additionally, we can normalize every object in the image, relative to the size of it. This means that all objects will have the same confidence for covering the same percentage of the image frame, relative to their respective MinSize.

Also, for blobs within a closer range, the larger a blob is, the higher its confidence is to be. Another exponential function is employed to describe this increase.

$$conf = \begin{cases} e^{\kappa(\frac{BlobSize-MinSize}{100})}, & BlobSize \leq MinSize \\ \kappa + (1-\kappa)e^{\frac{BlobSize-MaxSize}{1000}}, & MinSize < BlobSize < MaxSize \\ \frac{FrameSize-BlobSize}{FrameSize-MaxSize}, & BlobSize \leq MaxSize \end{cases} \quad (3.6)$$

where κ is used to decide the maximum confidence. This test showed to be even more necessary, when lack of a good color segmentation, section 2.2, resulted in seeing a blob covering the whole screen, which means that seeing a too large blob should be met with suspicion. Of course, there are rare cases when the dog pushes the camera against an object. But there should be other solutions in the Localization module for that.

Note that a beacon is the only landmark with a maximum number of pixels. It can, in normal conditions, never cover the whole image frame, for the reason that is mentioned in section 3.1.

2. What color is the blob?
   The range of the colors is divided into different classes depending on what object we think we are looking at. The more the class of that specific color is close to the desired one-the color that it is supposed to be viewed in-the higher the confidence.

   For the blue net the color has to be blue or black, and for the yellow one in a range that includes yellow, such as white-yellow-orange-red. The farther from yellow, the lower the confidence.

   When it comes to the ball, we will have to let the colors we are looking at in the progress, to shift between several different colors. As laboratory experiments showed, in strong light, the ball may look pink. Also, as UNSW reported in [9], "a shadowed ball takes on the same color as a red robot", and as the German Team, [35], reported "Shaded orange often appears as red", besides The University of Austin Villa, [42], claiming in their report that "The orange ball, whose color vary from orange to yellow to brownish-red depending on the amount of lighting available at that point" and at last, as members of Cerebrus have experienced it, "Bright light on an orange object some of it might look like yellow or even white" [24].

3. Is it totally in the image?
   If the whole object is not visible, which means that its bounding box touches some of the edges of the frame, deciding whether it *is* the object we think, and the distance to it, is more likely to be inaccurate. Hence, its visibility is marked with lower confidence, the more edges of its bounding box are in contact with the edges of the frame. This depends of course on what the object is. The closer to the "image center", the higher accuracy. Consider the following equations

$$\begin{aligned}\Delta x_{left} &= xMin \\ \Delta x_{right} &= FrameWidth - xMax\end{aligned}$$

$$\begin{aligned} BlobWidth &\leq min(\Delta x_{left}, \Delta x_{right}) \leq max(\Delta x_{left}, \Delta x_{right}) \\ &\Rightarrow conf_h = \frac{BlobWidth}{max(\Delta x_{left}, \Delta x_{right})} \\ min(\Delta x_{left}, \Delta x_{right}) &\leq BlobWidth \leq max(\Delta x_{left}, \Delta x_{right}) \\ &\Rightarrow conf_h = \frac{min(\Delta x_{left}, \Delta x_{right})}{max(\Delta x_{left}, \Delta x_{right})} \\ min(\Delta x_{left}, \Delta x_{right}) &\leq max(\Delta x_{left}, \Delta x_{right}) \leq BlobWidth \\ &\Rightarrow conf_h = \frac{min(\Delta x_{left}, \Delta x_{right})}{BlobWidth}, \end{aligned}$$

where BlobWidth is the width of the bounding box of the blob, $conf_h$ the degree of certainty of the object being what we expect it to be, at least in the horizontal

direction, and ..... . The same is done for "vertical confidence", $conf_v$, and the average of these two will be the total confidence.

4. What shape does it have? (square, round, rolled square, ...)
   This part is best described by an example. Suppose that we are looking at pink / red / orange / yellow blobs-the colors that are most probable for a ball to appear in, in different light conditions. With rating how near the shape of a ball a blob is-in form of varying confidence-we may choose the best one to be our desired object. The closer to the shape of a square, the higher the confidence. Besides, after taking into account the roll-angle, the more it is rolled, compared to the vertical orientation of the camera, the less the confidence. This is to be decided by the algorithm, explained in section 4.7

5. What is the ratio of its width and height?
   Depending on what the shape is, the more the ratio ($\frac{height}{width}$) is close to the correct value the higher the confidence.

6. Is the information about its real size and the blob size in accordance with the distance given by IR-sensor?
   Comparing the distance of the object with the distance that the IR-sensor may give at hand-for the same object-could also be a source for defining the confidence.

   The closer to the value of IR, the higher the rate. This presumes, of course, that the blob is positioned in the middle of the image for the IR to be able to read the correct value.

$$
\begin{aligned}
WS \quad & \longrightarrow IRvalue \\
if \quad & (blob.xMax \geq \frac{X\_Max}{2} \& \\
& blob.xMin \geq \frac{X\_Max}{2} \& \\
& blob.yMax \geq \frac{X\_Max}{2} \& \\
& blob.yMin \geq \frac{X\_Max}{2}) \\
& return(1 - \frac{|distance - IRvalue|}{IRvalue} = 2 - \frac{distance}{IRvalue})
\end{aligned}
\tag{3.7}
$$

   It may however be used for deciding the confidence for the "distance". For the model at hand this part is not to be taken into consideration. There are two reasons

   - It presumes that we mark what blob is "IR-sensed" at what distance in each frame
   - The IR-sensor is - as experiments showed - not accurate enough

7. How high from the ground is the object?
   Actually depends on where the "ground" is. This will make us to first look at the position of the beacons to see whether it is on a higher level than the blob we are testing. The values of pan and tilt also play a major role in deciding whether a blob is too high or too low. This information is possible to fetch from the World State, Chapter 1.

There are two possibilities

- To get the elevation, we can choose to decide some reference or look at the tilt-pan angles, or why not both. Looking at the focal plane of the camera, being above or under this point, will give us a hint about the blob being or not at the distance we think it is. See sections 3.8 and 3.9.
- Knowing the size of a known object makes it even possible to calculate the distance between its lowest point and the ground. This gives another fact to decide the confidence upon. The method is presented in section 3.9, Using The Elevation.

8. What are its neighboring blobs?
   The idea behind it is simply to check the neighboring blobs to see if the blob we currently are looking at really should be in that "neighborhood". The more undesired neighbors the less the confidence. To answer this question, we will have to take the order of the objects being tested into consideration. This means, we will test it with respect to those blobs that are already identified; Section 3.1.

9. What are the colors of its neighboring blobs? Color combinations around each object can lead us to weather the object is correctly identified; discussed previously. The ratio of number of "correct" neighbors and the total number of neighbors is yet another parameter to consider.

10. Is it at a reasonable distance?
    This last criterion can be tested in a particular order. For instance, first look at the one that is supposed to be a ball, and then the net, the landmarks, and the other objects. We add another criterion, that verifies how connected to its neighboring colors the object is. This means, as in a *Rule Base* we can look at the neighbors on the four sides of the blob and for each correct neighbor give it a 25 percent confidence

Now the blobs with highest scores are chosen to be the objects that are important to us. For instance, if there are four blobs qualifying for being the ball, they can be sorted in "credit" order, and the one with the highest credit will be chosen as the ball. This voting system will give us the possibility to decide the confidence of each blob being what it is estimated to be.

Another advantage of this approach is its modularity. If the criteria are to be added for classifying the objects, it suffices to add a function that gives every blob some points, which are added to the rest of the points they have earned from other criteria.[10] The great benefit of this modularity is that, even if some functions do not do their work properly, due to disturbances, other criteria will still make the recognition possible.

To decide the weights of these criteria is actually a matter of doing many experiments. But there is also another solution and that is a Neural Network.

In this approach, the training set will be consisted of images taken by the AIBO camera, with different objects on the field and in all conceivable positions, and presented to the software.

Usually, before such processing, images are to be enhanced in order to make them manageable by a neural network. Most of the time, this is carried out by applying one

[10]For what criteria to be chosen see for instance the UNSW's report from 2003 ([9], p.28).

of the methods of clustering, region growing, or filtering, all mentioned previously- as previous experiments has shown this is preferably not done on the AIBO. In our case though, as also frequently mentioned, the objects in each image are already presented as blobs. Each blob is then evaluated by every one of the 10 above-mentioned tests, from which a separate binary or real presentation $i_0 = x_{00}, x_{01}, x_{02}, ..., x_{0m}$, $i_1 = x_{10}, x_{11}, x_{12}, ..., x_{1m}$, ... , and $i_9 = x_{90}, x_{91}, x_{92}, ..., x_{9m}$ of the probabilities/confidences is earned, where m is the number of digits each value is presented with. A parallel combination of these values can serve as *input* to the network.

At the output, an ad-hoc evaluation of every object in the images can be presented, by the operator, and compared with the output, by the software. Running the program over a large set of images will eventually adjust the weights for each $i_n, n = |tests|$.

Chapter 4

Solutions

> Intelligence is determined by the dynamics of interaction with the world.
>
> Brooks [7]

Some parts of the system solution that are explained theoretically in the previous chapter, 3, often contain details that are not explained throughout the respective sections. The present chapter is specifically dedicated to those solutions to make the abstract notations more explicit. Therefore, it may look as disjoint set of methods but in the bigger picture outlined by the previous chapter all these parts fall easily into place.

In this regard, the next section, 4.1, outlines the structure of the module to give a big picture of the solution. Section 4.2 explains how a bipartite object in general and a beacon in particular is recognized by using the adjacency of the color blobs. Section 4.3 extracts the equations that are needed for calculation of the center of mass of a circle. The chapter continues with the section 4.4 discussing all possible aspects of computing the distance to an object. Utilization of a median filter in stabilizing the calculation results, through consequent image frames, is another topic presented in 4.5. Confidence for the recognized objects is named in section 4.6 and confidence, primarily used in Localization module, in section4.6. Finally *shape recognition* is discussed in 4.7.

4.1 Class Hierarchy

As noticed in the previous chapters of this report, although a recognition module is a single unit it still needs to be divided into different sub-modules, each taking care of different objects. The recognition process uses nevertheless common methods that are gathered in their respective classes to prevent redundancy and provide effectivity as is expected in a software module. In this section the subdivisions of the module are presented briefly.

Figure 4.1 show that all detected blobs are divided in level A, to different blocks at level B, depending on their colors; this does not necessarily mean that each blob is sent to only one block of level B. In each block, the blobs are then treated to find the desired object, using the special functions implemented in blocks at the level C. At the level D lies the block “Pixel Rec”, that contains methods for search and verifying the colors of the pixels.

All blocks have access to the “Rec Info”, which contains the overall information needed in the recognition system, such as the length and height of the image in number of pixels, and so on.

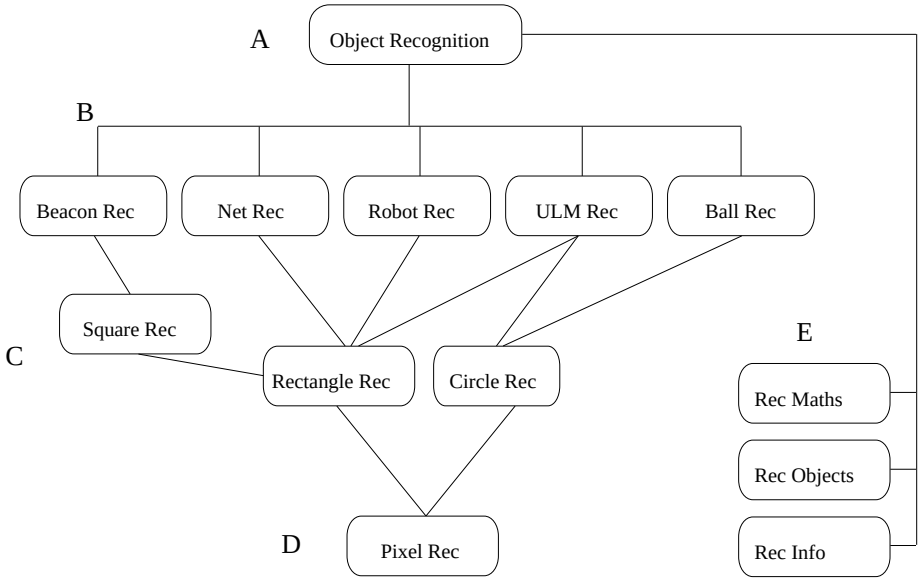


Figure 4.1: The hierarchy of the classes constitutes the whole "recognition system"

"Rec Maths" contains the mathematical methods, as the "distance filter", methods for computing the inclination and such.

When an object is recognized by a block at level B, all necessary information that can be used later on is saved in the private *struct* of that object in block "Rec Objects". This gives us the possibility to change the information about an object when desired, although this is to be done before all information about the found objects are saved in the World State and the token (page 7) is passed over to the Localization module. A good example is when a blob is merged into the blob that we believe is one of the nets. This schema has shown to be practical, for both coding and debugging.

4.2 Explaining the Method "Identify Neighbor"

As was explained before, we first look for pink blobs suitable to choose as candidates for being the main part of the beacon. This means we first look at the sizes and discard those pink blobs with $number\ of\ pixels \leq N$, where N is an experimentally found integer, as is mentioned in section A.2. For each pink blob we follow this procedure

- Find the center point of the blob
- Look at pixels from the upper side of the bounding box and as long as its height, in a straight line right above the middle point of the blob.
- Do the same for the pixels under the bounding box.
- Count the number of yellow and blue pixels, both over and under, and compare the results for all four.
- The dominating side-color combination, if now there is any that is over an experimentally predetermined value, decides if this is to be classified as a beacon and in that case which one of them.

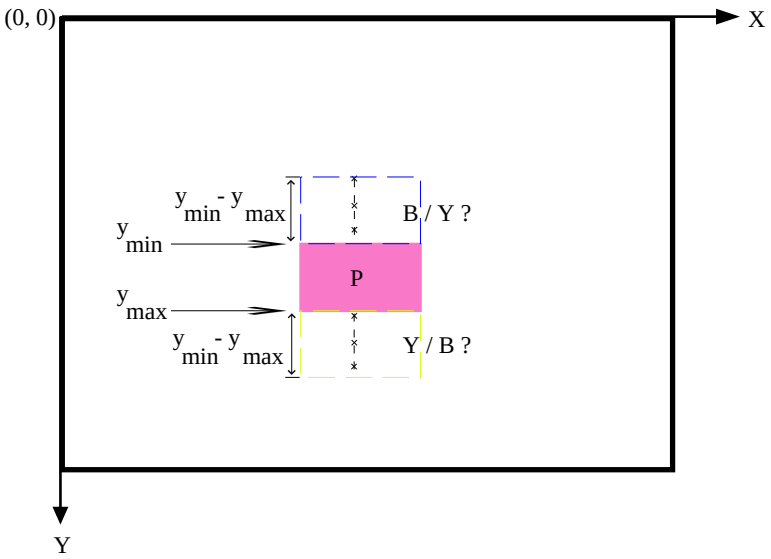


Figure 4.2: A pink blob in the middle of the image frame is being checked. The crosses show some of the points whose colors are checked in order to find out if there is a valid blob close to the pink one, making the second part of a beacon

Attempts were also made to add a method to: *always check whether there are white pixels under the lower colored part of the beacon*. Since the white section of the beacon, could be segmented as other colors than white-yellow in particular-this test was not successful. The method is illustrated in Figure 4.2.

4.3 Facing Close Range

Looking again at Figure 3.15, illustrating different cases of a ball at close range, one can see that in all cases but the second one on the first row, at least two points on the blob can be used for finding a third point C.

When the iteration is finished, depending on whether the Δx ($= |x_A - x_B|$) or Δy ($= |y_A - y_B|$) is the largest, a bisector is drawn perpendicularly through the largest one. This bisector is continued over the whole bounding box, to cut its opposite side in a point, called C; see Figure 4.3.

Now there is a good chance to find the origin of the circle, if the three points A, B and C, are all connected together with the lines AB, AC and BC, through which new *perpendicular bisectors* (PBS) are drawn towards the inside of the bounding box. Where these three meet is the origin of the circle.

When the inclinations of the lines are found, we can calculate where their PBSs cross each other.

Suppose an arbitrary point M to be somewhere on the xy-plane. Let us call the point on the middle of the line, connecting point A to M, $AM(x_{MA}, y_{MA})$, and likewise, the middle point of the line, connecting B to M, $BM(x_{MB}, y_{MB})$. Any line, perpendicular to the lines AM and BM, with the inclinations k_{MA} and k_{MB}, will have the inclinations $\frac{-1}{k_{MA}}$ and $\frac{-1}{k_{MB}}$ respectively; see Figure 4.4.

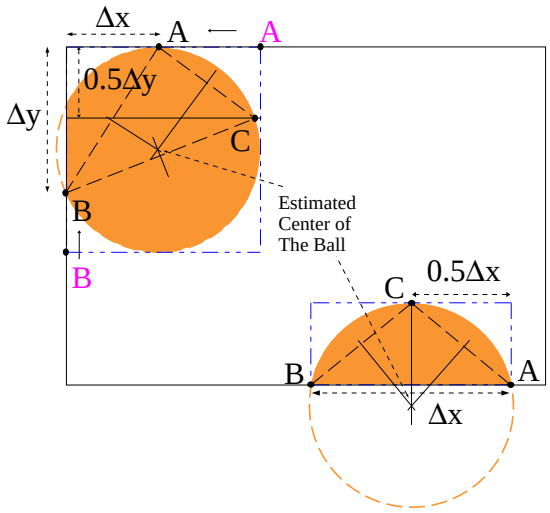


Figure 4.3: The two possible cases that can occur when comparing the lengths of Δx and Δy

This gives that the equations for the lines MA and MB are

$$y - y_{MA} = -\frac{1}{k_{MA}}(x - x_{MA}) \tag{4.1}$$

$$y - y_{MB} = -\frac{1}{k_{MB}}(x - x_{MB}) \tag{4.2}$$

Combining the two yields

$$\begin{aligned}
& \frac{-1}{k_{MA}}(x - x_{MA}) + y_{MA} = \frac{-1}{k_{MB}}(x - x_{MB}) + y_{MB} \\
\Leftrightarrow \quad & \frac{-1}{k_{MA}}x + \frac{1}{k_{MA}}x_{MA} + y_{MA} = \frac{-1}{k_{MB}}x + \frac{1}{k_{MB}}x_{MB} + y_{MB} \\
\Leftrightarrow \quad & (\frac{1}{k_{MB}} - \frac{1}{k_{MA}})x = \frac{1}{k_{MB}}x_{MB} + y_{MB} - \frac{1}{k_{MA}}x_{MA} - y_{MA} \\
\Leftrightarrow \quad & (\frac{1}{k_{MB}} - \frac{1}{k_{MA}})x = \frac{x_{MB}}{k_{MB}} - \frac{x_{MA}}{k_{MA}} + y_{MB} - y_{MA} \\
\Leftrightarrow \quad & x = \frac{\frac{x_{MB}}{k_{MB}} - \frac{x_{MA}}{k_{MA}} + y_{MB} - y_{MA}}{\frac{1}{k_{MB}} - \frac{1}{k_{MA}}}
\end{aligned} \tag{4.3}$$

We thus have the position of the cross-section of the two PBSs of the lines MA and MB, which is nothing but the center point of the circle passing through the points A and B.

$$x = \frac{k_{MB}^{-1}x_{MB} - k_{MA}^{-1}x_{MA} + y_{MB} - y_{MA}}{k_{MB}^{-1} - k_{MA}^{-1}} \tag{4.4}$$

$$y = y_{MB} - \frac{1}{k_{MB}}(x - x_{MB}) \tag{4.5}$$

In the same manner, the cross-section for the PBSs of the lines MA-AB and MB-AB are calculated. When all the three points are decided, we check whether all are on the

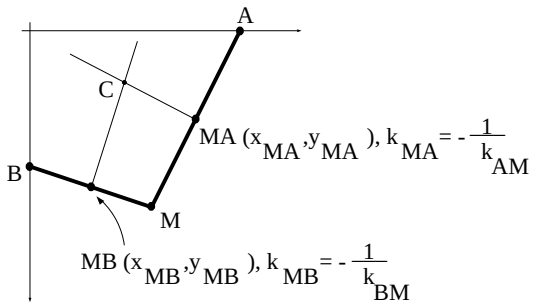


Figure 4.4: Finding the center of a circle. A, B and M are three different points on the perimeter of a circle, MA and MB middle points of the lines AM and AB respectively, and C is the cross point of the perpendicular bisectors C-MA and C-MB

same-or almost the same-pixel on the grid map. To find a point that instead would match all three we just calculate their average, in both directions, and let this average point be an estimation of the origin of the circle.

The existence and uniqueness of the circle is guaranteed by the theorems of Euclidean geometry. For proofs of these theorems, presented by Scott Sutherland, please refer to the website of the State University of New York at Stony Brook, [43].

4.3.1 Infinite Inclination

There is one case where we would not follow exactly the same procedure, namely when one of the PBSs has an infinite inclination; a sketch and the solution are presented in Figure 4.5.

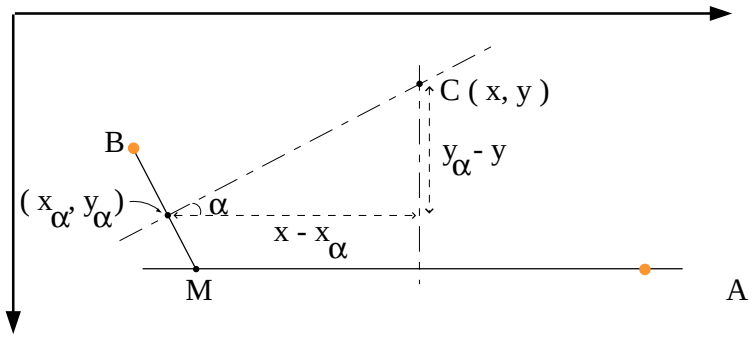


Figure 4.5: If AM is horizontal, its perpendicular bisector has an infinitely large inclination. This means we just need to put the X-values of the middle point of AM in the equation of the line C-BM

4.3.2 A Shortcut To the Third Point

Specially, when the blob is relatively large or both A and B lie on the same border, to make the procedure of finding the point M even faster, we use an alternative way, where

starting point for the search, as can be seen in Figure 4.6, is the point between A and B.

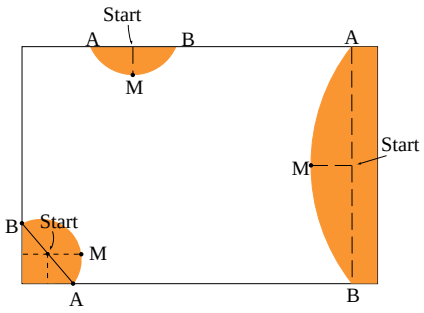


Figure 4.6: Time used for finding the start point can be slightly reduced if just the points A and B are connected together, virtually

4.4 Angle and Distance To Objects

Suppose that, in image 3.26, the size of the radius of the object is r_r $[mm]$ and r_i in number of pixels, ($[pixel]$), and its distance to the center is d_r $[mm]$ and $d_i[pixel]$, then the real distance d_r is

$$\frac{r_r}{d_r} = \frac{r_i}{d_i} \Rightarrow d_r = \frac{r_r \times d_i}{r_i} \tag{4.6}$$

Now, we need values to put in d_i and r_i to reach an answer for d_r. To have a reliable source for these measures, a simple experiment with a ruler is done, as the sketch in Figure 4.7 shows. At The distance $RealDistance$, $2 \times RealWidth$ of the ruler is visible in the horizontal direction. Imagine a ball being held in front of the camera in a way that its perimeter touches the upper and lower edges of the image frame. Since the size of the frame is $FrameWidth \times FrameHeight = 208 \times 160$, the diameter of the ball is also 160 $[pixel]$ which makes $ImageWidth$ to 80 $[pixel]$. The same value, $ImageWidth$, measured in unit length is

$$ImageWidth\ [mm] = \frac{ImageWidth[pixel]}{\frac{FrameWidth}{2}[pixel]} \times RealWidth\ [mm] \tag{4.7}$$

Then, we put the ruler aside and compare the real size and distance of the ball with the new values we have for its image, as if we have taken two images at two different distances, $ImageDistance$ and $BallDistance$. What is needed to do now is just a simple comparison, that is

$$\begin{aligned} \frac{BallWidth\ [mm]}{BallDistance\ [mm]} &= \frac{ImageWidth\ [mm]}{ImageDistance\ [mm]} \\ \Rightarrow BallDistance\ [mm] &= \frac{BallWidth\ [mm] \times ImageDistance\ [mm]}{ImageWidth\ [mm]} \end{aligned} \tag{4.8}$$

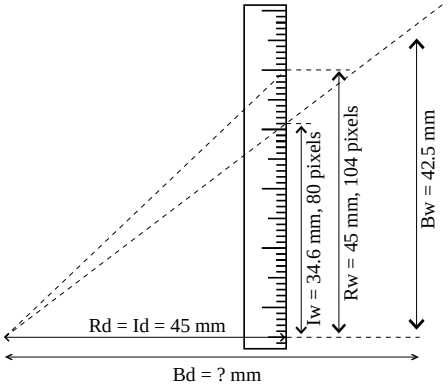


Figure 4.7: An object, *ruler*, at the distance D from the camera.

Putting the two equations, 4.7 and 4.8, together gives

$$\begin{aligned} BallDistance &= \frac{BallWidth \times ImageDistance}{\frac{ImageWidth \times RealWidth}{\frac{FrameWidth}{2}}} \\ &= \frac{ImageDistance}{2 \times \frac{RealWidth}{FrameWidth}} \\ &= \frac{FrameWidth}{2 \times \frac{RealWidth}{ImageDistance}} \times \frac{BallWidth}{ImageDistance} \end{aligned}$$

And the fraction $\frac{RealImage}{ImageDistance}$ is nothing but the $tangent$ of the half of the *maximum horizontal view angle of the camera*, $HFrameAngle$. This means that the distance to a ball is calculated as

$$Distance = \frac{FrameWidth\ [mm]}{2 \times tan(\frac{HFrameAngle}{2})} \times \frac{ObjectWidth\ [mm]}{ImageWidth\ [pixel]} \tag{4.9}$$

Equation (4.9) can actually be used to calculate the distance from any *known* object to the camera and thus, this is the one that is used for all objects throughout the report and the vision system.

An Example

The image of the ball with 80 pixels from the center of the image is $ImageWidth = \frac{80\ [mm] \times 45}{104} = 34.6\ [mm]$ from the center of the ruler at the distance $45\ [mm]$ from the camera but the real size of the ball is $86\ [mm]$. Hence, it must be situated at the distance

$$\frac{86\ [mm]}{34.6\ [mm]} = \frac{d}{45\ [mm]} \Rightarrow B = \frac{86 \times 45}{34.6} = 135.2\ [mm]$$

away from the camera.

Attempts were made to calibrate the *Infrared sensor*, IR, with the camera, in order to know where in the image the IR beam is pointing. The objective was to let the IR help us with estimating the distance to the objects in the image, as it could do that more accurate than the calculations. But this cooperation was not successful and hence discarded from the program.

A Standard Overhead

Since the angle unit is defined in $\mu\Omega$ (Micro Radian), we will have to convert the angles of the camera, 45.2 and 56.9 to $\frac{45.2}{180} \times 1570$. But in order to skip the time overhead for calculating $\frac{1570}{180}$ (each time we try to calculate ϵ and Θ), we put it as a global variable in the header file, WS (page 7). The result is shown in Figure 4.8.

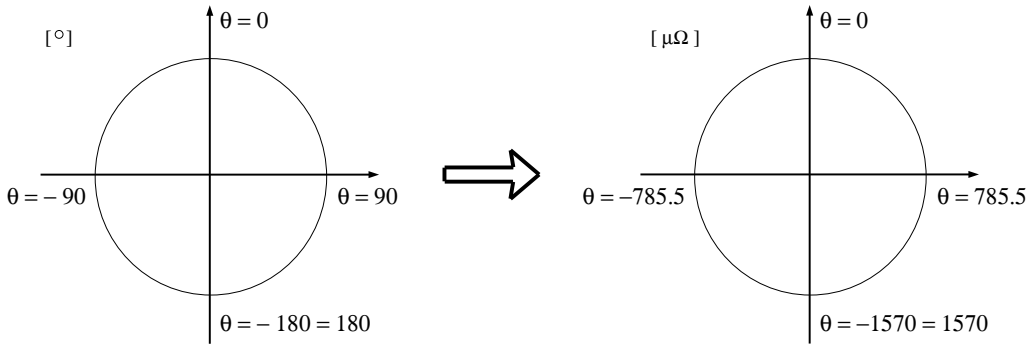


Figure 4.8: 1.37

But there are more. The value calculated above has to be multiplied by the maximum angle of the camera in either horizontal or vertical direction and then divided by the maximum pixel size in the same direction. This, in equation (4.9), gives

$$\begin{aligned} D &= \frac{\frac{208}{2}}{tan(\frac{CamAngle}{2})} \times \frac{ObjectWidth}{ImageWidth} \\ &= \frac{\frac{208}{2}}{tan(\frac{Cam_{Angle} \times \mu R}{180 \times Cam_{Size}})} \times \frac{ObjectWidth}{ImageWidth}, \end{aligned}$$

where the $ObjectWidth$ is the real size of the object, $ImageWidth$ is its size in the image, in number of pixels, Cam_{Angle} is the view angle of the camera, and Cam_{Size} is the size of the image frame in the same direction. Since these values are known for the current platform, all variables, μR, $ObjectWidth$ and $ImageWidth$, but the last two can be replaced by constants leaving D equal to

$$D = \frac{104}{tan(\alpha \times \mu R)} \times \frac{ObjectWidth}{ImageWidth},$$

where α is some constant equal to $\frac{Cam_{Angle}}{180 \times Cam_{Size}}$

4.4.1 Deluded by the Image?

What seemed to be very important in the beginning, only in the case of calculation of distance to the ball, is that since the imaging device is almost a pinhole camera[1] ([34], p.24), it does not image the largest circle of the ball; Figure 4.9.

[1] A pinhole camera consists of a box with a pinhole through which light enters and an image plane onto which the light is projected. Because light travels in straight line, an object is projected to a unique location on the image plane. ([11], p.84)

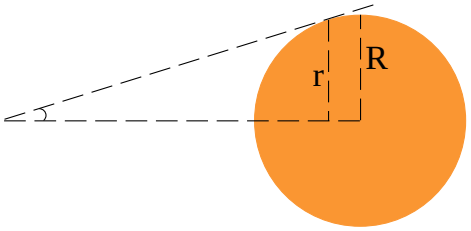


Figure 4.9: A pinhole camera can, theoretically, never image the largest circle of the ball, in particular from a close range

This problem becomes even more conspicuous, if the camera gets closer and closer to the ball. On the other hand and from the calculation point of view, the ball cannot be closer to the camera than the position, where the image of the ball covers the whole image frame. Since the accuracy of the information about the distance is even more important the closer we are to the ball the above-mentioned effect needed to be investigated.

By looking at Figure 4.10, one can see that

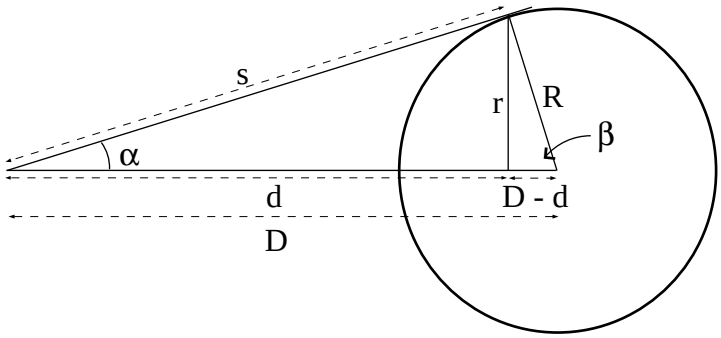


Figure 4.10: The situation from a geometric point of view

$$cos\ \alpha = \frac{R}{D}, \qquad sin\ \alpha = \frac{r}{R}$$

Using the equations above, we simply realize that the relation between "false radius of the ball", r, and the distance of the camera to the real center of the ball, D, is

$$\begin{aligned} \frac{R^2}{D^2} + \frac{r^2}{R^2} &= 1 \\ \Rightarrow\ r^2 &= (1 - \frac{R^2}{D^2})R^2 \\ \Rightarrow\ r &= R\sqrt{1 - (\frac{R}{D})^2} \end{aligned} \tag{4.10}$$

The graph in Figure 5.1 shows how r changes when the ball moves closer to the camera. We notice that, for example, on a distance of $D = 100\ [mm]$, the radius of the circle we

see in the image frame is only $r = 38.5\,[mm]$, on $D = 150\,[mm]$, $r = 40.8\,[mm]$, and on $D = 250\,[mm]$, $r = 41.9\,[mm]$. This is a result of combining the general equation for distance to an object, (4.9), and the equation (4.10) above. Consider that in (4.9) *Distance* to the object and its $ImageWidth$, in pixels, can change places to give

$$ImageWidth = \frac{FrameWidth}{2 \times tan(\frac{FrameAngle}{2})} \times \frac{ObjectWidth}{Distance}$$

and since the ObjectWidth is no longer the same as the "largest diameter" of the ball, one can replace it with (4.10) to compute it, using the information we have at hand. This produces the combination

$$ImageWidth = \frac{FrameWidth}{2 \times tan(\frac{FrameAngle}{2})} \times \frac{R\sqrt{1 - (\frac{R}{Distance})^2}}{Distance},$$

which is the length of the radius to appear in the image, in pixels, when it is situated on the actual distance, *Distance*, from the camera. For the case of the ball, the $ImageWidth$ can be replaced with this new value to give the distance that we, by mistake, will estimate, when we use 4.9

$$BallDistance = \frac{FrameWidth}{2 \times tan(\frac{FrameAngle}{2})} \times \frac{R}{\frac{FrameWidth}{2 \times tan(\frac{FrameAngle}{2})} \times \frac{R\sqrt{1-(\frac{R}{Distance})^2}}{Distance}}$$

Most of the variables can be elimniated to yield the final equation

$$BallDistance = \frac{Distance}{\sqrt{1 - (\frac{R}{Distance})^2}} = \frac{Distance}{\sqrt{1 - (\frac{42.5}{Distance})^2}}$$

The result of this part is presented in Chapter 5.

4.4.2 Sensitivity in Counting the Number of Pixels

Results from the tests, presented in Chapter 5 show that the accuracy of the estimation of distance decreases with increasing distance to the ball (see Figure 4.11). The reason was thoroughly investigated and is presented in this section with the help of a numerical example.

For the final decision of the distance to the ball, when the size of its radius in the image (*RaySize*) is calculated the following equation is used for estimation of the real distance

$$d = \frac{1}{2} \times \frac{\frac{208}{2} \times \frac{42.5}{tan(\frac{56.9}{2})} + \frac{160}{2} \times \frac{42.5}{tan(\frac{45.2}{2})}}{RaySize} \tag{4.11}$$

Results from tests are presented in Chapter 5. As expected, they show less accuracy of the estimation, the more the ball is distanced from the camera. The reason why we will not arrive at a correct value all the time is because the curve, as mentioned earlier in

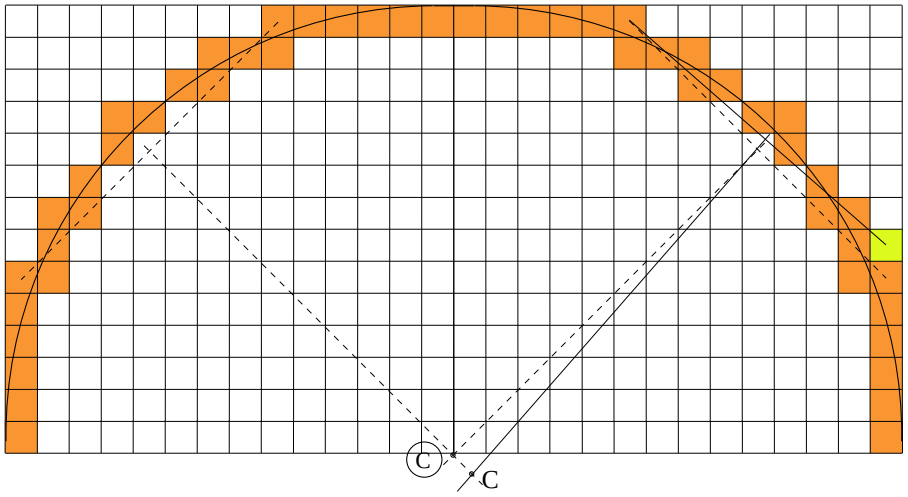


Figure 4.11: Making a mistake with only one pixel, for a ball blob, 28 pixels wide, leads to a false estimation of the origin of the circle with one pixel. This in turn gives 11 percent estimation error for a distance of only $55cm$

section 3.2, is reshaped to (partially) straight lines and finding points and coupling them with each other would give lines that are almost parallel, resulting in too large blobs and hence too close distances, or the other way around. For an example suppose that the distance to the ball in Figure 4.11 is to be estimated. The dashed line shows the correct radius of the circle. In this case the length of the radius is $ImageWidth = \frac{10}{cos\ 45} = 14.1$, which in (4.4.2) gives the value

$$d_{10} = \frac{1}{2} \times (\frac{104}{tan(\frac{56.9}{2})} \times \frac{42.5}{14.1} + \frac{80}{tan(\frac{45.2}{2})} \times \frac{42.5}{14.1}) \simeq 581.0\ [mm]$$

And in the same manner, for the false radius

$$ImageWidth = \frac{11}{cos\ 45} \simeq 15.6mm,$$

which in (4.4.2) gives

$$d_{11} = \frac{1}{2} \times (\frac{104}{tan(\frac{56.9}{2})} \times \frac{42.5}{15.6} + \frac{80}{tan(\frac{45.2}{2})} \times \frac{42.5}{15.6}) \simeq 523.3\ [mm]$$

The difference is thus $d_{11} - d_{10} = 581.0 - 523.3 \simeq 57.7mm$, or about $6cm$. That is, for a distance of $55cm$, there will be 11 percent error in the estimation.

4.4.3 Edge Margin

An edge margin of size 5 is used to make a cleaner environment while finding the ball boundaries. The reason is that near the frame edges, the pixels are often corrupted, which results in poor estimations. This was experienced many times when the code was being tested. It is also reported in [18]

> Distortion to color is significant around the edge of the image and much more noise than usual is introduced around the edge of the frame, which

makes it non-ideal for our purpose. Thus we shrink the border upon which we perform edge detection by 5 pixels from the frame border

A set off *by 5 pixels* is to be performed carefully, noting that some procedures or methods use the information about length and height, and hence it is crucial for them to keep the changes transparent.

4.5 Median Filter

The length of the *History* is calculated using

$$\frac{maximum\ velocity\ of\ the\ dog}{nbr\ of\ frames\ per\ second}$$

Since the maximum length walked by AIBO is $\frac{180\ [mm/s]}{14[frames/s]} \approx 12.857\ [mm/frame]$, after 5 frames the distance to the object has changed with $\frac{180}{14} \times 5 = 64.3mm$. This means if the AIBO is walking to an object with the maximum velocity, its estimation of the distance should not change more than $64mm$, if we compare for each 5 frames.

On the other hand, since the filter used is a median filter, permanent changes in the distance will not settle in faster than $\lceil \frac{length\ of\ the\ filter}{2} \rceil$ frames, presumed that the length of the filter is an odd number.

Hence, it was decided to put the maximum change of estimation to $64mm$, and the length of the filter to $1 + nbrOfFrames \times 2$, which in this case makes a length of 5.

This also means that it will only take 5 frames-about $\frac{1}{18} \times 2 \approx 0.14$ seconds-to start making an estimation of the distance to the ball and after 5 frames, it will start giving stable estimations of the distances to objects. Longer filters could be used but then it would delay the start of the estimation, that is to take place whenever an object is invisible for more than 5 frames before being observed again.

4.6 ”Confidence” for the Localization Module

Primarily, the idea was to let the *voting system*, presented in 3.10, provide the localization module with the necessary confidence value. Due to a malfunctioning voting system, in the version sent to the competitions after a frame is color-segmented and the objects are recognized, each one is marked with an age-counter, initially set to 1.0, that shows how old, counted in number of frames, it is. This counter is used for deciding the confidence of the object in the *Localization Module* and decreased by a certain amount for each image frame in which the object is not present.

4.7 Shape Recognition

For a four legged robot, it could be beneficial to be able to distinguish between the three different shapes, rectangles, triangles and circles. It is obvious that triangles and rectangles can form identical shapes, if they are not completely in the image frame; Figures 4.13b and 4.13c. The exception is if the “free edges” of the blob—those that do not lie on either of the frame edges—are totally parallel. And in a digital image, with the poor quality of the images, this is nothing to rely on. The same is valid for the

circles, when they are too far away. Hence, we have to presume that the whole object is present in the image frame and it is within a reasonable distance, to be recognized with high accuracy.

For a circle, a condition for beginning the "circularity check" should be that the blob must occupy at least half of the bounding box. It is easy to realize that this *is* the case, knowing that the blob would not be a circle, if its boundary is not convex. And, since all four sides of the bounding box are to be in contact with the blob, one cannot find a case where the blob is circular but does not cover at least half of its bounding box (Figure 4.12). This is a good point to start with for recognizing circles.

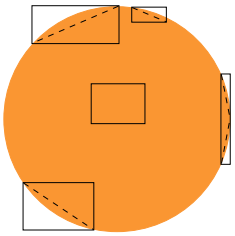


Figure 4.12: No matter what part of the ball is visible, at least half of the area of its bounding box must be covered with ball-colored pixels. Since the edge of the ball is always convex, there is a good reason to presume that the number of orange pixels is to be larger than half of the number of the pixels of the bounding box

There are in general no good, easy alternatives for checking whether the blob edges follow a circular path, unless we use a more computationally expensive circle detection algorithm. The only easy way that remains, is to use the second method, discussed in section 3.2.2.

When it comes to the other shapes, there cannot be any constraints in the same manner. From Figure 4.13a, we notice that there is no way we can classify the triangles and rectangles, based on the number of edges or corners that are occupied by the blob. According to the *first fundamental theorem of projective geometry*[2], a general convex quadrangle is assumed to be the projection of a rectangle[3]. Hence, we can not demand a fixed number of color changes along the edge of the bounding box in order to distinguish whether the blob is of the one or the other type.

Despite of that, in general, if we suppose that the whole object is visible, there is a good chance to distinguish circles from rectangles and triangles [4].

[2]Denote E and $E^{'}$ as two vector spaces. First fundamental theorem of projective geometry states

> *Let* $P(E_n)$ *and* $P(E_n^{'})$ *be two projective spaces of same (finite) dimension* n, m_i, $i = 1, ..., n+2$, *and* $m_i^{'}$, $i = 1, ..., n+2$, *two projective bases of* $P(E_n)$ *and* $P(E_n^{'})$, *respectively. Then there exists a unique homography* $g : P(E_n) \longrightarrow P(E_n^{'})$ *such that* $g(m_i) = m_i^{'}, i = 1, ..., n+2$.
>
> ([12], p.86)

A *homography*, is any transformation H of $\mathbb{R}^n$ which is linear in projective coordinates (hence the terminology *linear projective*) and invertible (thus $H(\mathbb{P}^n) = \mathbb{P}^n$). It can be described by an $(n+1) \times (n+1)$ non-singular matrix $\mathbf{H}$, such that the image of x is $x^{'}$.([12], p.67)

[3]Proof of this theorem is out of the scope of this report

[4]The method was discarded after the stage of Matlab implementation, as it showed not to be successful probably for the same reason pervaded the whole work.

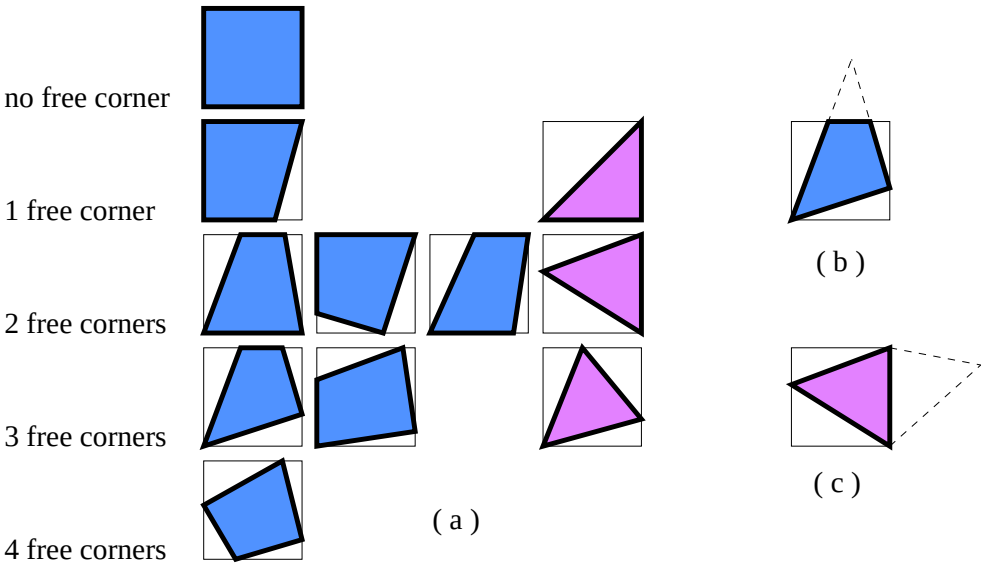


Figure 4.13: Sketches of all different poses the image of a rectangle (triangle) can have within its bounding box. a)Note that according to the First Fundamental Theorem of Projective Geometry a general convex quadrangle is assumed to be the projection of a rectangle. b)A section of a triangle can be mixed up with the image of a rectangle and c)vice versa

4.7.1 Edge Detection

To proceed with the recognition, two stages are ahead. In the first one, we have to find the edge of the blob. As been discussed before, there are at least three ways to finding pixels on the perimeter of a shape[5].

1. *Convolution masks*[6]; ([15], p.48)
2. Using rays in horizontal and/or vertical direction(s); Figure 4.14a
3. Following the perimeter of the blob; Figure 4.14b

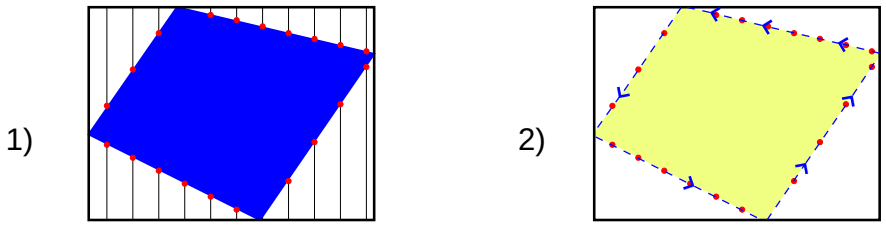


Figure 4.14: The two methods for collecting edge points a)Using grid lines b)Following the edges

The first method, is used in many of the edge detection applications and is based on a mathematical ground. The idea behind the mask operations is to let the value assigned

[5] A very usual and relatively easy method to limit the information in an image. What is left is a matrix with binary or bipolar values for each pixel wherever there is a contrast between two adjacent areas.

[6] Also known as *templates, windows, or filters*.

to a pixel, be a function of its gray level and the gray level of its neighbors. Usually, Sobel[7] and Canny[8]-see [32]-are renowned edge detection masks in this context.

As a second alternative and perhaps the easiest and most reliable one, parallel lines are drawn along the vertical or horizontal direction with start points on the edges of the bounding box. Colors of all points on each line are checked with the expected color of the blob, as the line proceeds. When the expected color is encountered, the search is interrupted and the coordinates of the last visited point are saved.

The third method is an intuitive, yet simple and functional algorithm:

Blob Pixel	$\rightarrow$	Go to the left
		Continue until Empty Pixel & not on boarder
Empty Pixel	$\rightarrow$	Go to the right
		Continue as long as Empty Pixel
Blob Pixel & On Boarder	$\rightarrow$	Go to the right
		Continue until Empty Pixel
Empty Pixel & On Boarder	$\rightarrow$	Check every fourth pixel for a Blob pixel
		if(not) do as usual from last Empty Pixel

Finish when you come to an already visited pixel

All three methods have of course gone through simple tests and seem to work well. However, due to the restrictions we have on time usage, the one with best result and least computation time is to be chosen. In this regard, the first alternative would not be our best choice, considering the computing resources. For an $M \times N$ blob, $9 \times M \times N$ multiplications are to be carried out. This would mean $299,520$ multiplications for a blob with 160×208 pixels.

Originally, the third method was meant to be used for ball recognition. But this is of course costly. Additionally, because the ball blob is not always convex, the procedure is not as simple as in the above mentioned procedure. Hence it was discarded already before the final implementation was done. Still, compared to the second method, and of course the first one, this is the least time consuming procedure to use for the two other shapes. It is easy to realize that the largest number of pixels to visit, is for a rectangle that covers the whole frame. That is $2(FrameWidth + FrameHeight) = 2(208 + 160) = 834[pixel]$, while for the second method, the maximum number is for a rectangle positioned along one of the diameters of its bounding box, having a width of only a pixel-in other words a straight line. If the bounding box is as large as the image frame, and the grid lines are drawn vertically, from every 10th pixel on the frame edge, then $\lfloor \frac{208}{10} \rfloor \times 160 = 3200$ pixels are to be visited.

4.7.2 Shape Verification

When enough points are found, two methods can be used to find the shape they construct

[7] One of the easiest methods for Edge Detection. It is based on calculating the first derivative of the image matrix at hand using the simple convolution masks/kernels or Fourier transforms followed by thresholding the data.

[8] Another method used for this purpose. Its larger convolution kernel smooths the input image to a greater extent and so makes the operator less sensitive to noise. The major problem with this method is that it is time consuming and hence not suitable for a real-time procedure, unless very powerful platforms are used.

Hough or Radon Transform , are methods that could preferably be used on a platform with no restrictions on time. As been mentioned before, this is not the case

Side Angles Simple comparison of edge inclinations is an alternative that, although simple, can work effectively. In this method, all edge points are arranged in the order they are encountered, if the perimeter of the "shape" is followed, (counter-) clockwise. Let us call the points $\pi_1, \pi_2, ..., \pi_n$. A straight line is drawn through each couple of points (π_j, $\pi_{j+\delta}$)-where δ is an integer-constituting the lines

$$l_1, l_2, ..., l_m$$

-where $m = \lfloor \frac{n}{2} \rfloor$-with the inclinations $k_1, k_2, ..., k_m$, respectively. Going through the latter values, one can assume every sudden persistent change, that is more than a predetermined constant, to be a sign of an existing corner.

The great advantage of this procedure is that it can be done at the same time as the perimeter-points are being detected using the second or third method, whereas the edge detection using the mask operations is to be finished before the Hough transform is performed. This means that as soon as the corners are detected, the procedure can be interrupted, if desired.

Chapter 5

Evaluation

As discussed in 4.4.1 the effect of the pin-hole camera needed, theoretically, to be dealt with. The result presented in section 5.1 is meant to show why the attempt to find a solution was revoked.

In section 5.2 an evaluation of the whole vision module is presented. Discussions related to the results follows then in the next chapter.

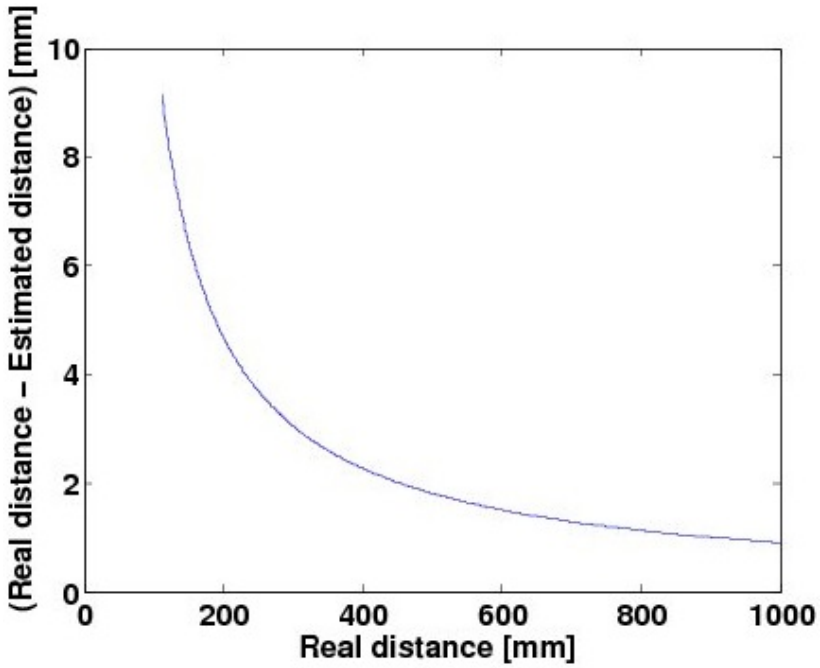


Figure 5.1: The pinhole camera does not affect the estimation of distance remarkably

5.1 Effect of the Pinhole Camera on Distance

From beginning, it seemed to be crucial for shorter ranges, where kicking or grabbing the ball can be affected. But the calculations in section 4.4.1 showed that this is not an

issue to worry about, compared to the poor estimation that may be made as a result of orange-non-orange-pixel mix up; section 4.4.2. Figure 5.1 illustrates the reason for this: already after a distance of $300\ [mm]$ this difference is down to only $3\ [mm]$.

5.2 The Distance Estimation

The recognition module showed to be successful in recognizing objects and providing the necessary information to the whole system, with sufficient precision and accuracy.

A comparison between this system and the other implementations from other teams is though not possible due to the fact that non of the reviewed teams from the previous year, had reported any explicit results from evaluation of their respective vision modules. Another issue making a comparison almost impossible is that the platform used in the previous year was different, ERS-210. The only remaining method for evaluating the systems is hence based entirely on the overall performance of each team.

Nevertheless, an attempt was made to evaluate the system based on what it could perform in recognizing the ball as an indication for the performance of the module. Recognition of the ball and the distance to it are the most delicate issues in the whole recognition module. As has been discussed throughout the report, the only moving object on the field is the ball. Other objects, from beacons and nets to opponent robots and unknown objects, are all static. These are used only as landmarks and presence or absence of one does not affect the performance of the localization module crucially as absence of a landmark can easily be compensated for by considering another landmark. But the ball and information about its position and distance are irreplaceable. The robot needs to be aware of this piece of information in any moment and the only source that can provide the necessary information is the ball itself. The idea was that if the robot can successfully manage the task of localization then there will be no need for testing the recognition and distance information of each and every landmark explicitly. This does not mean that the statistics related to those objects were not considered, but it needed not to be presented as an evidence for a well-tuned recognition module.

Therefore, many tests were run to verify the module's accuracy in this regard. Of course, there are many different situations to test the ball in, but here, the easiest and the most difficult positions are presented.

Two different sets of tests were conducted that are mentioned in the following, one before the competitions and one after they were over.

5.2.1 Pre-Competition Tests

Five different cases were chosen for the tests, *long range*, *partly visible*, *hardly visible*, *occluded by opponent's leg*, and *close to opponent's body*. The error values yielded were as follows

Long Range: 25.19% of error if the ball is not occluded and at the distance of 2250 mm. For a ball occluded with a dark obstacle up to half of its radius and positioned at 1000 mm from the camera, the error is only 5.57%.

Partly Visible: For a ball occluded up to half, with a dark obstacle, if it is positioned at 250 millimeters, the error is 9.94%, whereas at 200 millimeters, the error is 7.30%.

Hardly Visible: If at least 75% of the ball is occluded with a dark object, at 560 millimeters distance, the error is 23.14% and at 550 millimeter the error is equal to only 2.24%. The difference can be motivated by the shape of the visible part of the ball. See Figure D.3.

Occluded by opponent's leg: With the opponent's leg covering the middle part of the ball, at 600 mm's distance the error is 8.33%, but covered on the lower part, in the "UNSW position" and at the same distance, the error is as much as 18.17%.

Close to opponent's body: If the body of the opponent is put on the ball, for the same distance of 600 mm, the error is 4.7%, but if it is behind the ball, the error decreases to 2.75% for the same distance. For the previous position a distance of 700 would surprisingly decrease the error to 3.43%.

In figures, presented in appendix D, every position is illustrated together with a sketch, in what the real distance and the number of pixels, according to the segmentation module are shown. Also the *median* (m), *mean* (μ), and *standard deviation* (σ) are calculated and shown; Figure D.1 through D.3. In order to eliminate the risk for reflection of the color orange, obstacles with non-metallic black color were chosen to cover the ball-except where another AIBO ERS-7 with red patches was used; Figure D.4 and D.5.

Last in the series of figures, a graph showing the overall performance of the vision system for the estimation of the distance to the orange ball is added, Figure D.6. For relative constant lighting conditions, the estimation works hence reasonably well.

5.2.2 Post-Competition Tests

According to another evaluation of the same system[1] [31], performance of the recognition module can be estimated as follows:

Ball: Around or less than 10% of error. The absolute error is $10cm$ at maximum. The error in measurement of the angle is less than 0.1 radians, except for the range closer than 0.5 meters, where the error is 0.5 radians. The overall average of the error is only 0.01 radians except for close range where it is as low as 0.05 radians.

Landmarks: The average error for the distance is 4 to 30% and for the angle about 0.01 to 0.1 radians

Nets: In average the error is 20 to 25% for the distance and 0.1 to 0.8 radians for the angle

Robot: "Then the distance was estimated based in the height of the blob. This approach was tested but it did not work very well, yielding a lot of bad measures. It was mainly because of no precise segmentation but also because the blobs are seen in different ways depending on from which side the robot is seen."

The results from the ball-recognition are more or less consistent for both tests. A discussion regarding the results will follow in chapter 6.

[1] Whether the conclusions made in that thesis are in accordance with what can be concluded from the overall system performance needs a thorough discussion in the presence of the author of that report.

Chapter 6

Conclusion and Future Works

A discussion of the results presented in the previous chapter follows in 6.1 and based on that and what has been reported in this thesis, some desired changes and improvements of the system are outlined in section 6.2.

6.1 Discussion

According to the evaluation made before the competitions, reported in chapter 5, and the evaluations reported in [5], and [31] the recognition module showed to be successful in recognizing objects in the laboratory environment and providing the necessary information to the localization module, with sufficient precision and accuracy particularly for the recognition of the ball. Despite of that, in the competitions, the whole system collapsed as a consequent of failures in all modules and sub-modules, from segmentation to planning.

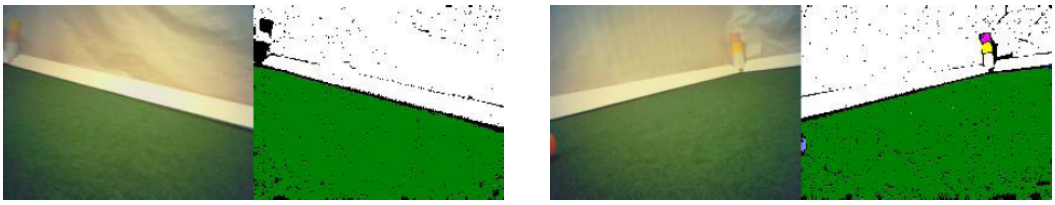


Figure 6.1: *Left*: no information retrievable from the image. *Right*: what we should hope to be a rare case: a blue ball!

A comparison of the conclusions in [5] and [31] together with the images provided in the second report, added in Appendix D.2 shows that besides lack of needed complexity in all the modules, a poor color segmentation easily "brought the AIBO to its knees".

As described in the beginning of this report a lighting-independent object recognition had never been concerned in previous years and hence a "regular" vision module was not designed to handle these conditions which during the development of the module was specially cared for. As the presented results show, the distances of the objects to the robot were also estimated very well and with high accuracy and provided to the localization module for further operation.

Figure 6.2: *Left*: only a beacon, the AIBO can be anywhere in at least one fourth of the field. *Right*: only a goal, the AIBO can be anywhere in a circle close to the goal, although this would be a contradiction to its long distance

The Overall Performance

Despite of the promising results pointed above, the overall performance of the module seemed to be far from the expected. To investigate the reason other reports from the same team have been consulted.

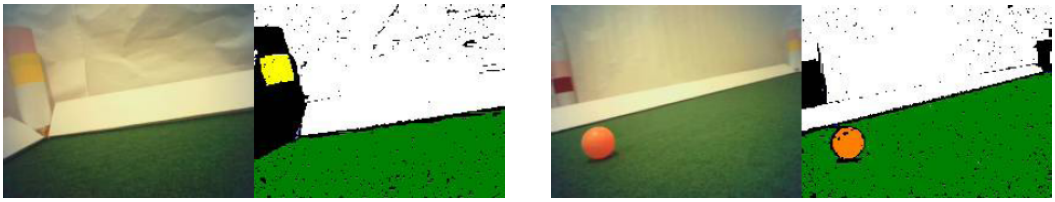


Figure 6.3: *Left*: only a yellow blob, could be the goal if its neighboring colors are not considered. *Right*: no landmarks at all, it can be anywhere, close to the wall.

After the competitions, a clearly extensive evaluation has been performed by Navarro in [31]; most of the images used in this evaluation are presented in the Appendix D, out of which some are shown in figures 6.1 to 6.6. In over 50% of the 85 cases—49 images with segmentation failure(s) and 36 without—one or more of the colors/objects are not correctly segmented in the image. Still, the final conclusion about the color segmentation is, amazingly, that "*As a conclusion it can be said that the segmentation with these current color tables works rather well and most of the colors are segmented correctly in most of the frames.*"

Figure 6.4: *Left*: no information about the ball can be extracted. How can AIBO know if it has scored or if it should look for the ball somewhere else on the field? *Right*: the only available information about the blobs can be retrieved using their sizes. A rule-based reasoning would be the solution.

It is an inevitable fact, that a poor color-segmentation will in no way give a recognition module any chance to correctly classify objects on and around the field and extract the necessary information about them.

The rules of the challenges, B.4.2, states clearly that:

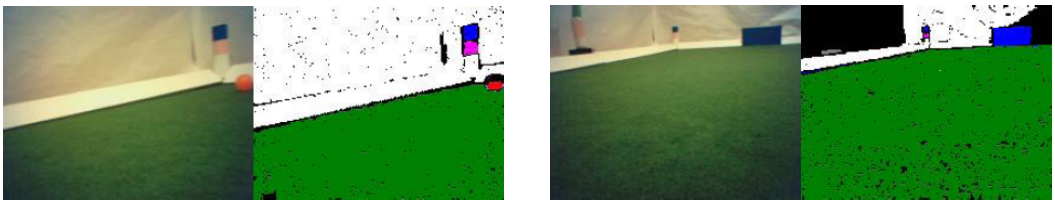


Figure 6.5: *Left*: color of the ball is red and its shape closer to rectangle than circle. Very probable to be recognized as a robot. *Right*: the unknown landmark is not presented in the image and consequently no information will be returned about it

> *This second challenge is intended to encourage teams to increase the robustness of their vision to illumination changes.*

And illumination changes are what the segmentation algorithm should take care of, unless the raw image is provided to the vision module for further work.

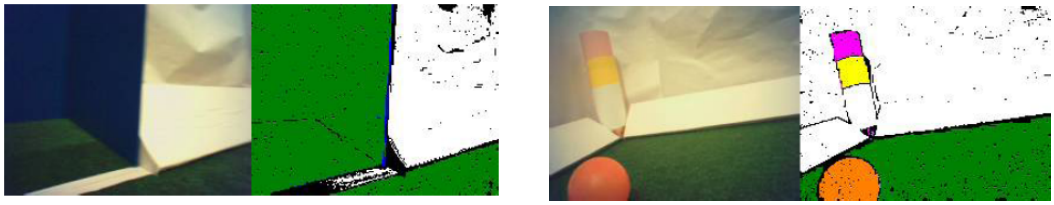


Figure 6.6: *Left*: blue net segmented as carpet and white stripe as black. *Right*: a rare case, a correctly segmented image!!

The participating Dutch team's, [33], comment

> *In June 2004 one Bachelor student started a study on color invariance algorithms for the Variable Lighting Challenge (Pieterse, 2004)... In this study a comparison was made between algorithms based on a Gaussian color model... or a color-histogram... The conclusion of this study was that the algorithm based on color-histograms is applicable on the Aibo platform. The computational resources on the ERS-7 are clearly not enough to facilitate an algorithm based on a Gaussian color model.*

shows on the importance of the evaluation of the color segmentation. And the following statement from a German team, [21], shows why the color segmentation of TCC could not be successful, specially for the variable lighting challenge:

> *A very common preprocessing step for vision-based object recognition in color coded scenarios is color segmentation using color tables... Such methods directly map colors to color classes on a pixel by pixel basis, which has some crucial drawbacks. On one hand, the color mapping has to be adapted when the lighting conditions change, on the other hand, the mapping results in a loss of information, because the membership of a pixel in a certain class is a yes/no decision, ignoring the influences of the surrounding pixels.*

Despite of the failure in the color segmentation and the methodology used, the recognition rate is relatively high. In as many as 90% of the images, the distance to the ball

is computed successfully. The result is lower for landmarks with 96 down to 70%, and for the nets 80 to 75%. The reason for the robots not being tested explicitly is that they were only obstacles. Their position in the field would not change the localization of the robot. Of course, it would affect the planning but this was obviously not the main reason for the system to fail. As quoted by the TCC captain:

> ... we decided not to participate in Open Challenge... In SLAM challenge, phase 2, all our robot did was just standing in place wagging its tail... it didn't manage to get sufficiently close to the target locations... In the Variable Lightning challenge, since our penalty kicker completely hopeless, we decided to use standard Team Chaos penalty kicker - it barely managed to touch the ball, so at least here we outperformed teams who didn't decide to participate.

In another comment, not included in any report:

> ... For example, we do not use *any* odometry, so only objects currently seen are represented with any accuracy. Head movement is quite bad, so we cannot be certain that we will see anything not in front of the dog. Even simple behaviors don't seem to work and are incredibly difficult to use. Errors in localization are large enough to make it mainly useless for most tasks.

These statements show clearly that even using a functioning system, such as the Team Chaos penalty kicker, without a good color-segmentation is doomed to failure.

Finally

Other factor that plays an important role is probably the short time compared to the character of the tasks, not only for the vision but also for planning and localization. Usually, other, more frequently successful teams use the same architecture, and make minor changes and add new local features and modules depending on the type of the challenge, from year to year. But TCC's decision to recreate a whole system from scratch was obviously not a good idea.

Nonetheless, it is always easier to blame on other factors instead of trying to compensate for shortcomings. After all, the vision system lacks the necessary complexity and a future team would need to make some drastic changes and also incorporate some advanced features into the existing code. The following sections present some of these features.

6.2 Shortcomings and Improvements

A glimpse on the whole report reveals the fact that there is still some work left to do to make this module and the whole TCC-system a winning contribution to the competitions.

- Eliminating the defect and, at the same time, managing the changing-light conditions, requires an ability to check the lighting conditions by looking at the white walls and defining constant parameters for different YUV-values. That could help

us change our expectations when looking for different colors. The more lighting values are lowered the finer the thresholding should be, when doing the color segmentation. This calls for a real-time color calibration.

- An alternative filter can be used in place of the simple Median Filter, explained in the section 3.7, where depending on the velocity of the robot and the head pan, the filter length can be changed, if the velocity is available.

- By comparing two consequent frames we should be able to update the information about distance and angle to objects. If the information is not reasonable, we may discard one or the other, based on our confidence comparing the same object in the two frames. The other way around, we could higher the confidence of our estimation of what objects there are in every image. In general, cooperation between localization module and recognition module would lead to a higher accuracy and confidence. For example, if the AIBO sights the blue net at a distance of 4 meters, it will never be able to see the same net at the distance of 2 meters, in the next frame-no matter how good pace it keeps. This goes back to how cooperative the two modules are. On the other hand, we can also estimate the velocity of the AIBO by comparing two images, using the displacement of the objects and the time between two frames.

- As the lenses used in AIBO cameras are obviously not the most precise type of lenses, using the method discussed in C.1 for camera calibration may help to remove some distortions in the image frames leading to better color segmentation.

- For verification of the outcome of the recognition module, for every single frame, increasing its reliability there could be a lookup table with what other objects are reasonable to be seen in the same frame as the blob/object to which the credit is given. For a discussion on how *reasoning* could help to make this improvement see C.2.1.

- If the same color-blob-based method is used in future implementations, a function for recognition and merging of shattered objects can be useful, specially if applied before estimation of distance and angle. Some hints on this method are presented in C.3.

- There are yet many more changes that could be made in segmentation and localization to make the whole system robust to variations in the conditions of the environment. But this is per definition not within the scope of this project and needs to be discussed in the relevant reports.

- The recognition module, presented in this paper, has been coded in such a way that makes it independent of the platform. This means that it can easily be ported to other platforms without any considerable changes. To make the porting of code easier for an operator a *Graphical User Interface* could make the procedure of porting easier. In that interface, a table should appear, where the operator may insert the information, necessary for the vision system to operate. This would eliminate the necessity of going through the entire code and changing values whenever the vision system is to be used on a different platform. The information is then saved in a header file to be available for the whole system, as is already done.

Bibliography

[1] E. Angle. *"Interactive Computer Graphics: A top-down approach with OpenGL,"* Addison Wesley Longman, August 2000.

[2] R.C. Arkin. *"Behavior based robotics,"* MIT Press, 2000.

[3] C. Axelsson and J. Törner. *"Color Calibration in ERS-210,"* October 2003.

[4] C. Axelsson and J. Törner. *"Color Calibration: Software for color calibration of Sony AIBO robots in a RoboCup environment,"* 2003.

[5] S. Bie, J. Persson. *"Behavior-Based Control of the ERS-7 AIBO Robot,"* Master's Thesis, Lund University, Lund Sweden, 2004.

[6] V. Braitenberg. *"Vehicles: Experiments in Synthetic Psychology,"* The MIT Press, Cambridge, Massachusetts, 1986.

[7] R.A. Brooks. *"Cambrian Intelligence: The Early History of The New AI,"* The MIT Press, Cambridge, Massachusetts, 1999.

[8] P. Buschka, A. Saffiotti, and Z. Wasik. *"Fuzzy Landmark-Based Localization for a Legged Robot,"* Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS) pp. 1205-1210. Takamatsu, Japan, 2000.

[9] J. Chen, E. Chung, R. Edwards, N. Wong, B. Hengst, C. Sammut, W. Uther. *"Rise of the AIBOs III - AIBO Revolutions,"* November 2003.

[10] D. Cohen, Y.H. Ooi, P. Vernaza, and D.D. Lee. *"The University of Pennsylvania RoboCup 2003 Legged Soccer,"* Team, 2003.

[11] G. Dudek , M. Jenkin. *"Computational principles of mobile robotics,"* Cambridge University Press, New York, NY, 2000

[12] O. Faugeras and Q.T. Luong. *"The Geometry of Multiple Images,"* MIT Press, March 5, 2001.

[13] R. Fisher, S. Perkins, A. Walker, E. Wolfart. *"Hough Transform,"* Retrieved from: http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm .

[14] D.A. Forsyth and J. Ponce. *"Computer Vision: A Modern Approach,"* Prentice Hall, 1st edition, August 14, 2002.

[15] R.C. Gonzalez, R.E. Woods. *"Digital Image Processing,"* Addison-Wesley, Longman Publishing Co., Inc. Boston, MA, USA.

[16] M. Green. *"Recognizing a Pattern-Featured Object in a RoboCup Environment,"* Sweden, August 2003.

[17] K. Gurney. *"An Introduction To Neural Networks,"* UCL Press, 1997, Reprinted by Taylor & Francis Books, 2004.

[18] B. Hengst, D. Ibbotson, S.B. Pham, C. Sammut. *"The UNSW United 2000 Sony Legged Robot Software System,"* November 2000.

[19] L. Iocchi and D. Nardi. *"S.P.Q.R. Legged Team Report from RoboCup 2003,"* 2003.

[20] M. Johansson. *"A primer on Fuzzy Control,"* Lund Institute of Technology, Sweden, January 1996.

[21] M. Jüngel, J. Hoffmann, and M. Lötzsch. *"A Real-Time Auto-Adjusting Vision System for Robotic Soccer,"* Institut für Informatik, LFG Künstliche Intelligenz, Humboldt-Universität zu Berlin, Germany, 2003.

[22] D. Kortenkamp, R. Bonasso, and R. Murphy. ed. *"Artificial Intelligence and Mobile Robots,"* AAAI Press, 1998.

[23] K. LeBlanc, S. Johansson, J. Malec, H. MartŠnez, and A. Saffiotti. *"Team Chaos 2004,"* Sweden, 2004.

[24] H. Levent Akın, M. Kaan Baloğlu, Hatice Köse Bağcı, Suzan Bayhan, Çtin Meriçli, Damla Poslu, Ömer Sever, O. Taner Yıldız, Svetlozar Argirov, Boris Marinov, Petya Pavlova, Nikola Shakev, Jordan Tombakov, Andon Topalov. *"CERBERUS 2003 TEAM REPORT"*.

[25] X. Li, Z. Zhang, X. Liu, Q. Li, K. Xu, Y. Feng, L. Jiang, X. Chen. *"WrightEagle 2003 Sony Legged Robot Soccer Team,"* 2003.

[26] *"Mi-Pal 2003"*.

[27] *"Mi-Pal 2004"*.

[28] M. Mukaidono. *"Fuzzy Logic For Beginners,"* World Scientific Publishing Co, 2004.

[29] R.R. Murphy. *"Introduction to AI Robotics,"* MIT Press, 2000.

[30] H. Najima, Y. Yasutake, T. Kato, T. Kadowaki, K. Matsumoto, K. Oda, T. Ishimura, T. Ohashi. *"The Kyushu United Team 2003 in the Four Legged Robot League,"* 2003.

[31] I.N. Oiza. *"Reactive Cooperation of AIBO Robots,"* Master's thesis. Lund University, Lund, Sweden, 2004.

[32] J. NG and H. Cheung. *"Dynamic local feature analysis for face recognition,"* Titanium Technology Research Center, Hong Kong, PR China.

[33] S. Oomes, P. Jonker, M. Poel, A. Visser, M. Wiering, W. Caarls, S. Leijnen; S. van Weers, N. Wijngaards, and Frank Dignum. *"The Dutch AIBO Team Report on RoboCup 2004,"* The Dutch RoboCup team, Holland, January 2005.

[34] S. Russell, and P. Norvig. *"Artificial Intelligence: A Modern Approach,"* Prentice Hall, Upper Saddle River, New Jersey, 1995.

[35] T. Röfer, O.V. Stryk, R. Brunn, M. Kallnik, M. Kunz, S. Petters, M. Risler, M. Stelzer, H.-D. Burkhard, U. Düffert, J. Hoffmann, D. Göhring, M. Jüngel, M. Lötzsch, I. Dahm, M. Wachter, K. Engel, A. Osterhues, C. Schumann, J. Ziegler. *"German Team RoboCup 2003,"* 2003.

[36] A. Saffiotti. *"Handling uncertainty in control of autonomous robots,"* Reprinted in: M. Wooldridge and M. Veloso, eds, Artificial Intelligence Today, LNAI 1600 (Springer, DE, 1999) pp. 381-408.

[37] A. Saffiotti and K. LeBlanc. *"Active Perceptual Anchoring of Robot Behavior in a Dynamic Environment,"* Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 3796-3802. San Francisco, CA, 2000.

[38] A. Saffiotti and Z. Wasik. *"Using Hierarchical Fuzzy Behaviors in the RoboCup Domain,"* In: C. Zhou, D. Maravall and D. Ruan (eds) Autonomous Robotic Systems, pp. 235-262. Springer, DE, 2003.

[39] F. Serra and J.-C. Baillie. *"AIBO programming using OPEN-R SDK; Tutorial,"* June 2003.

[40] *"Sony Corporations American Online Shop,"* Retrieved from: http://www.sonystyle.com/

[41] H. Stewénius, *"Studies in Mobile Camera Calibration,"* KFS AB, Lund, Sweden, 2003.

[42] P. Stone, K. Dresner, S.T. Erdogan, P. Fidelman, N.K. Jong, N. Kohl, G. Kuhlmann, E. Lin, M. Sridharan, D. Stronger, and G. Hariharan. *"The UT Austin Villa 2003 Four-Legged Team,"* 2003.

[43] S. Sutherland. *"MAT 200 Course Notes on Geometry,"* Stony Brook Mathematics Department. Fall, 2002. Retrieved from: http://www.math.sunysb.edu/ scott/mat200.fall02/Geometry/Main/index.html

[44] *"Tekkotsu: An open source project created & maintained at Carnegie Mellon University,"* Retrieved from: http://www.cs.cmu.edu/ tekkotsu/

[45] Z. Wasik, and A. Saffiotti. *"Robust Color Segmentation for the RoboCup Domain,"* Proc. of the Int. Conf. on Pattern Recognition (ICPR-2002) Quebec, Canada, August 2002. To appear.

[46] D. Young. *"Sussex computer vision teach files,"* School of Cognitive and Computing Sciences, University of Sussex. Brighton January 1994. Retrieved from: http://www.cogs.susx.ac.uk/users/davidy/teachvision/

Appendix A

Facts

A.1 SONY AIBO Key Specifications

- Dimensions :180 (W) x 278 (H) x 319 (D) mm
- Weight Approx. 1.65kg (including battery & memory stick)
- CPU : 64bit RISC Processor
- CPU Clock Speed : 576MHz
- Memory: 64MB SDRAM, Memory Stick for Program Storage
- Movable Parts
- Mouth - 1 degree of freedom
- Head - 3 degrees of freedom
- Leg - 3 degrees of freedom x 4 legs
- Ear - 1 degree of freedom x 2
- Tail - 2 degrees of freedom
- Camera :CMOS Image Sensor 350,000 pixels
- IEEE 802.11b (Integrated)Wireless LAN Card
- Audio: Miniature Microphones, Miniature Speaker, 20.8mm,500mW, MIDI, Volume Switch
- Built-in Sensors
- Infrared Distance Sensor (head, body)
- Acceleration Sensor
- Electric Static Sensor (head, back)
- Pressure Sensor (chin, paws (4)

- Vibration Sensor
- Temperature Sensor
- Power Consumption: Approx. 7W (Standard operation in autonomous mode)
- Operating Time: Approx. 1.5 Hours (Standard operation in autonomous mode)
- Charging Time: Approx. 2.5 Hours
- LED mode and emotion lights :
- Illume face: 24 LED (white 12, red 4, blue 4, green4)
- Ear : 2 (left & right)
- Head sensor : 2 (white and amber)
- Head (wireless LAN on/off) : 1(blue
- Back sensor : 16 (white 8, red 3, blue 3, orange 2

A.2 Some Constants

These are the constant values that are used throughout the implementation. For computations that are presented in this report, these are to be used. For more information on how to use these values, please consult the theory .

- The image frame has the maximum size of 208×160 pixels.
- The horizontal view angle of the camera is $56.9°$ and the vertical view angle $45.2°$.
- The diameter of the ball is $85\ [mm]$.
- Least (or largest) number of pixels a blob may have to be recognized as this very object
  - This is the maximum size of a ball whose diameter is as large as the height of the frame, reduced with a margin of 5 pixels that usually blur the blob near the edge of the frame so the maximum will be: (this blurring effect is reported even by the team of UNSW [9]) $\pi \times 70^2 = 15394$
  - Experiments have shown that blobs occupying less than 10 pixels are most likely nothing but noises; to be sure let's take as the minimum $\lceil \pi \times 3^2 \rceil = 28$
  - For the same reason as with the maximum size for a ball, the maximum area should be:
    - $$\begin{aligned} NET\_MAX &= MAX\_PIX - 2{\times}EDGE\_MARGIN \\ &\times \left(X\_SIZE + Y\_SIZE - 2{\times}EDGE\_MARGIN\right) = 29700 \end{aligned}$$
    - $NET\_MIN = 200$

- And for the same reason as above, the maximum number of pixels for a beacon part may be:
  - $BEACON\_MAX = \left(Y\_SIZE - 2{\times}EDGE\_MARGIN\right)^2 = 22500$
  - $BEACON\_MIN = 15$;
  - $ROBOT\_MIN = 8$;

Appendix B

RoboCup Rules

B.1 Sony Four Legged Robot Football Field

Following are the rules about the setup of the environment, from the rule book of the *RoboCup Technical Committee*, as of May 20, 2004; the text is copy-pasted without any changes in the context:

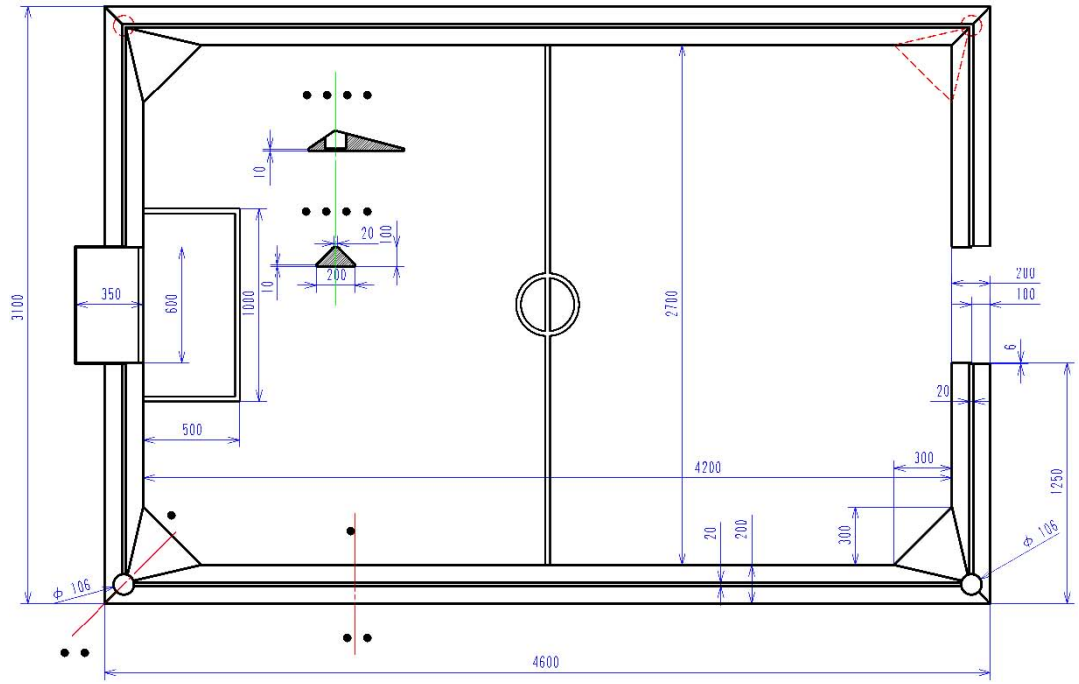


Figure B.1: Field dimensions in mm

B.2 Setup

B.2.1 Field Construction

The dimensions of the soccer field are shown in Figure B.1. The construction of the goals and the beacons is depicted in Figure B.2. Please note that the upper border of the

beacons is at 400 mm, because there is still 10 mm of the field wall below them. Also note the special connection between the field wall and the goals (cf. Figure B.3).

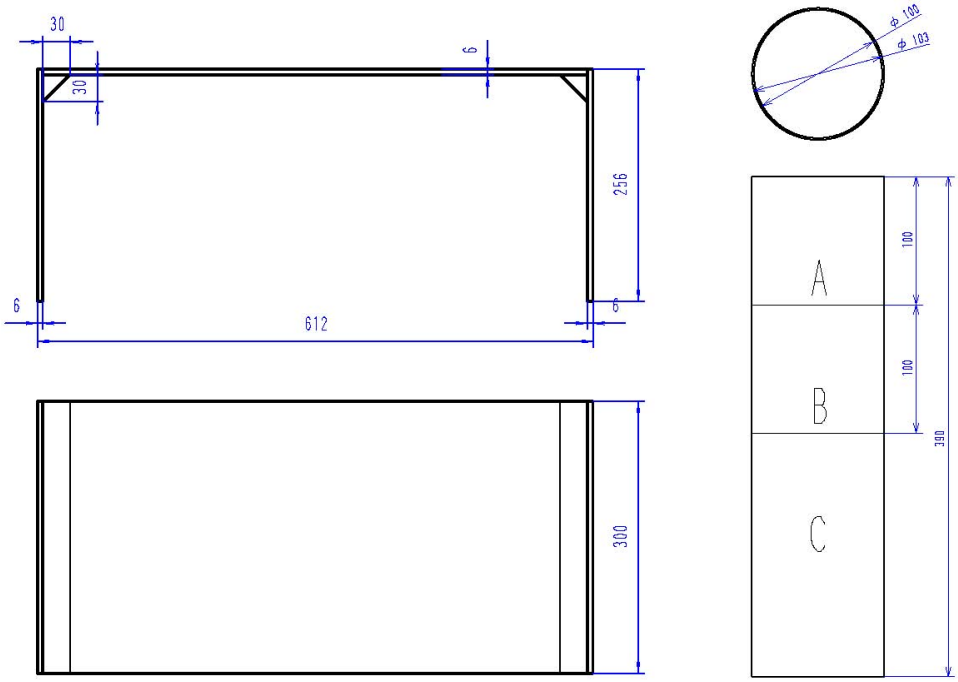


Figure B.2: The top and side views of the goals-on the right side-and the beacons-on the left side

In addition to the field shown in Figure B.1, there is an outer white wall with a height of 30 cm surrounding the field. The distance between the wall and the field is not defined.

B.2.2 Lines

All the lines on the soccer field (the halfway line, the lines surrounding the penalty areas, the goal lines, and the center line) are drawn with white stripes of 25 mm in width. The circle on the midfield line has diameter of 300 mm from the middle of the white stripe on one side to the middle of the white stripe on the other side, i. e. its outer diameter is 325 mm and its inner diameter is 275 mm.

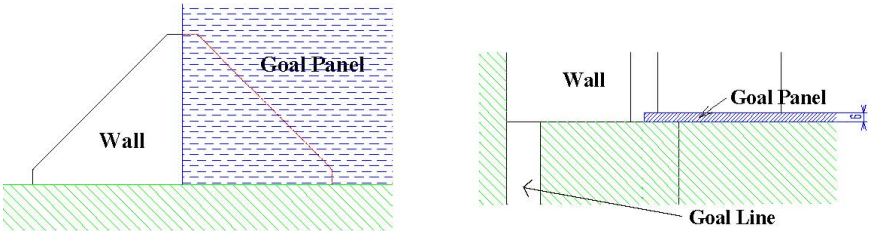


Figure B.3: The connection between the field wall and goal; side view on the left and the top view on the right side

B.2.3 Field Colors

The colors of the soccer field are shown in Figure 2.4. All items on the RoboCup field are color-coded:

- The field (carpet) itself is green.
- The lines on the field, the field wall, and the outer wall surrounding the field are white.
- The red team defends the yellow goal.
- The blue team defends the sky blue goal.
- In each corner of the field, there is a cylindrical beacon (cf. the right side of Figure B.2). Part C is always white. When looking from the yellow goal toward the sky blue goal, part B of the two beacons on the right of this axis shares the color with the neighboring goal, and part A is pink. On the left side of the field, part of the beacons is pink and part shares the goal color.

B.3 Lighting Conditions

The lighting conditions depend on the actual competition site. In 2004, they may differ significantly from previous years, because only ceiling lights are used. The event organizers guarantee a brightness of at least 500 lux on the field.

B.4 RoboCup Challenge Rules

B.4.1 The Open Challenge

This challenge is designed to encourage creativity within the Legged League, allowing teams to demonstrate interesting research in the field of autonomous systems. Each team will be given three minutes of time on the RoboCup field to demonstrate their research.

Each team should also distribute a short, one page description of their research.

The winner will be decided by a vote among the entrants. In particular:

- Teams must describe their demonstration to a designated representative of the organizing committee at least one day before their demonstration.
- Each team may use any number of Sony AIBO robots. Teams must arrange for their own robots.
- Teams have three minutes to demonstrate their research. This includes any time used for initial setup. Any demonstration deemed likely to require excessive time may be disallowed by the organizing committee.
- Teams may use extra objects on the field, as part of their demonstration. Robots other than the AIBOs may not be used.

- The demonstration must not mark or damage the field. Any demonstration deemed likely to mark or damage the field may be disallowed by the organizing committee.
- The demonstration may not use any off-board sensors or actuators, or modify the AIBO robots.
- The demonstration may use off board computing power connected over the wireless LAN. This is the only challenge in which off board computation is allowed.
- The demonstration may use off board human-computer interfaces. This is the only challenge in which off board interfaces, apart from the Game-Controller, are allowed.

The winner will be decided by a vote among the entrants using a Borda count. Each entering team will list their top 10 teams in order (excluding themselves). The teams are encouraged to evaluate the performance based on the following criteria: Technical strength, novelty, expected impact and relevance to RoboCup. At a time decided by the designated referee, within 30 minutes of the last demonstration if not otherwise specified, the captain of each team will provide the designated referee with their rankings. Each ranking is converted to points: ten points for the top ranked team, nine for the team ranked second and so on down to one point for the team ranked tenth. Any points awarded by a team to itself will be disregarded. The points awarded by the teams are summed and the team with the highest total score shall be the winner.

B.4.2 The Variable Lighting Challenge

This second challenge is intended to encourage teams to increase the robustness of their vision to illumination changes. It is based on a penalty shoot out. The team attempting the challenge places a single blue robot (robot with a blue uniform) on the field. That robot must score as many goals as it can into the yellow goal in three minutes (Note: this is the opposite goal from a normal penalty shootout). The team that scores the most goals wins. If two teams score the same number of goals, the result is a draw.

In addition to the single blue robot, two red opponent robots are also placed on the field. Both of these robots are paused, frozen in the UNSW stance. Neither of them shall move during the challenge. One is placed in a goalie position on one side of the yellow goal. The other is placed in the third of the field nearest the yellow goal, at least 30cm away from the edge. The exact locations of all the robots shall be determined by the referee, and will be the same for all teams.

There is a single ball upon the field. Initially it is placed in the center kickoff position. Upon each score, the ball is moved back to the center kickoff position. The robot is not moved by the referee and must make its own way back to the center of the field to reach the ball again. The robot will have its back button pressed when the ball is moved back to the center to indicate a score. The game controller shall not be used.

Teams are warned that the normal game use of the back button, penalization, does not apply in this challenge. Since the field does not have borders anymore, it is very likely that the ball may be kicked out of the field. In this case, it is placed back at the center of the field.

The main challenge in this task is that the illumination shall be different from standard RoboCup lighting. Some additional lights and suitable equipment shall be brought

in to supply variable lighting conditions (we envisage variable strength lighting using theatrical lighting equipment). These additional lights shall be white light of deliberately unspecified color temperature. Lights may also be covered to achieve variable lighting conditions.

Before the challenge the referee shall prepare a schedule of illumination changes. This shall include periods of constant illumination, periods of slow change in lighting and periods of rapid changes in lighting. It is envisaged that the additional lighting will be non-uniform across the field and hence the lighting changes will be non-uniform. This lighting schedule, though unknown until right before the challenge, shall be the same for all teams.

Unless otherwise specified, normal penalty shootout rules apply. There will be no penalty for charging the opponent robots. But, it is not allowed to help a robot stop charging and move away from or around another robot.

An example lighting schedule:
Time (min) Lights
0:00 - 0:30 All lights
0:30 - 1:00 Lights at blue end of field only
1:00 - 1:30 Lights at yellow end of field only
1:30 - 2:00 Background illumination only
2:00 - 2:15 Blue end lights solid, yellow end lights flashing
2:15 - 2:30 Yellow end lights solid, blue end lights flashing
2:30 - 3:00 All lights flash

B.4.3 The SLAM Challenge

The SLAM challenge is intended to help the league move away from strictly defined beacons to more generic localization information as in a soccer stadium. In order to achieve this, additional landmarks are placed around the borders on a RoboCup field.

The challenge consists of two stages. In the first stage, the robots are given time to explore the field. In the second stage, the normal beacons and goals are covered up or removed, and the robot must then move to a series of points on the field, using the information learnt during the first stage.

These additional landmarks shall have the following constraints:

- They shall all be outside the playing area but on the green field.
- They shall be of varying size and color.
- They are guaranteed to be unique when color and orientation are taken into account.
- There shall be at least three of these landmarks containing a patch of pink at least 10cm across.
- There shall be at least six landmarks.
- They shall be at least 15cm apart.
- They shall contain no white or black, although they may have a stand that is black or white.

- Each dimension shall be between 10cm and 50cm.
- They shall be no more than 50cm above the field.

Before the challenge the referee shall choose appropriate landmarks, five points on the RoboCup field and one restart point (and angle). The selected points shall be written to each team's memory-stick as a text file in the topmost directory: points.cfg. The format of the file has one target point per line, the x coordinate followed by the y coordinate. There is an example file available at the league website, in the "Downloads" section. The coordinates are given in cm, and the origin of the coordinate system is at the center of the field. The x-axis points from the blue goal (negative x) to the yellow goal (positive x). The y-coordinates to the right of this axis are negative; on the left they are positive. Each team is responsible for writing code to read the file with the target locations. Points are guaranteed to be at least 15cm from the nearest obstacle, and at least 100cm from any other point.

In the first part of the challenge, all normal landmarks and the additional landmarks are visible. The robot performing the challenge must start paused. The referee will place the robot at a point on the field (same for all teams) and then activate the robot by touching its head sensor. The referee will then leave the field area. The robot shall move about the field for less than one minute. It will then pause itself. If the robot takes over one minute to pause itself at this point, it will be disqualified.

Between the two parts of the challenge, the referee will cover or remove all the normal landmarks and goals. The referee will move the robot to the restart point and angle. The referee shall then activate the robot by pressing its head sensor and then start the timer.

Upon activation, the robot shall start moving to one of the target points. When it thinks that it is close to the target point, the robot shall pause itself and indicate to the referee that it believes it is near a target point (usually by wagging its tail). At this point the referee will pause the timer, place a small marker underneath center of the robot, and then re-activate the robot and re-start the timer.

The second stage ends when the robot has had two minutes, or when it has stopped five times. At the end of the second stage, all robot position markers more than 50cm from any field point are disregarded, and if there are multiple markers within 50cm of a single point then only the closest is kept. Teams are then awarded 150-d points for each visited marker, where d is the distance from the marker to the point in centimeters. They are then awarded 5x (120-t) points, where t is the total time used in the second stage measured in seconds.

Teams will be ranked as follows: First, they will be ranked by the number of markers they reach (within 50 cm). When two teams reach the same number of markers, the score determines their rank.

Another way of looking at the scoring is as follows:

- You start with 600 points.
- You lose 5 points per second.
- You get 100 points for reaching a marker (within 50cm). At 5 points per second, this means you need to reach that 50cm circle within 20 seconds to make it worth your time.

- For each 1cm improvement in accuracy you get another point. At 5 points per second, this means you need to increase your accuracy at 5cm/s to make it worth your time.

Appendix C

Investigations

C.1 Eliminating the Distortion

According to [41], in computer vision, mainly 2 types of calibration are investigated. First the problem of *camera calibration* and secondly the problem of *hand-eye* calibration. Camera calibration deals with how to determine a set of parameters of a camera. This is discussed in the next sub-section and could be a good way to eliminate yet another source of error.

In hand-eye calibration the camera is rigidly attached to a robot hand for which the motion is known, the task is to determine the translation and the rotation from the hand to the camera as well as the internal parameters of the camera. This procedure, the way it is conducted in [41], is in fact too much power consuming and is more applicable for larger platforms and therefore out of the scope of this thesis.

C.1.1 Camera Calibration

Very often distortions in camera itself causes failures in calculating the motion, and distances to and sizes of the objects. The distortions are mainly caused under the process of manufacturing the body of the camera lenses or arise from the their physical properties, such as *radial distortion*, in which image magnification decreases/increases with distance from the optical axes. Parameters have to be taken into account and dealt with in the software, in order to eliminate the distortion.

According to the rules of linear algebra and projective geometry, an object in the *3D* space and its image on the retinal plane of a *2D* pinhole camera can be expressed as a 4×1 vector, U, and a 3×1 vector, u, respectively. The relation between these two vectors is then

$$\lambda u = PU \tag{C.1}$$

with λ as the scaling factor, where P-the *projection matrix*-is a 3×4 matrix having the quality of being able to transform the object U to the image u, under a rotation, using a quadratic 3×3 matrix, $K \times R$, and a translation using a 3×1 vector, $-K \times R \times t$.

Hence the matrix P can be expressed as

$$P = [KR| - KRt] = KR[I| - t]$$

The 3D quadratic matrix K-even called the *Calibration Matrix* ([14], p.33)-is an upper triangular matrix, holding the camera depended *Intrinsic Parameters*, affecting the form and the dimensions of the images on the retinal plane.

Assuming a 2D camera being used for measuring the size of the different objects in a 3D space and their respective distances to the camera, there will be 5 parameters in the matrix K,

$$\begin{pmatrix} \alpha_u & \gamma & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}$$

that has to be found, before the camera is used for any purpose, see Figure C.1. This is called *Camera Calibration*.

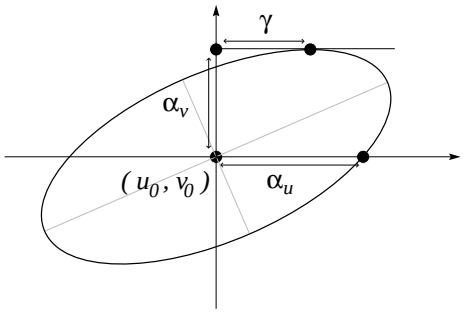


Figure C.1: The effects of the intrinsic parameters on an image. ([12], p.56, fig.1.35)

C.2 Strange Blue Rings

Besides what have been mentioned in the context of this report, yet another impediment to our recognition system to work as planned is bluish rings that appear in the images, taken by the camera of the AIBO. This makes the whole system fail in many cases and many frames. The ”defect” was discovered quite late, as it is hard to notice, and as a temporary solution we had to ignore the corners. The cost was, of course, less accurate recognition as a cause of the limited visual angle. One of the most important tasks to do in a future implementation of the vision system would be to find a way to eliminate the defect, if the same platform is to be used in the future.

C.2.1 Rule-Based Verification

After recognizing all blobs-or maybe after classifying each one of them-the information gained from each particular blob could be stored as Horn Clauses[1]. Then, with respect to the rules that we have settled, we can decide whether the latest piece of information about the object is valid. This eliminates the need for checking the blobs with each other, which for a large amount of blobs being in the image frame at the same time, would be an unnecessarily huge waist of time. An example of the most conceivable rules are

[1] Appendix C.2.1

- All objects besides the goals are lower than beacons
- A goal that is completely in the image is always lower than a beacon.
- A ball is always lower than the walls, the beacons, the goals and the head of the opponent dogs (used in the challenges).
- Goals are always surrounded by white walls, except the lower side, where it is in a green field's neighborhood.
- Two goals can not be seen at the same time
- Three beacons can not be seen at the same time
- Every goal is situated between two particular beacons. On the other hand, there can not be any goal between two particular nets.
- Each beacon occurs on a particular side of each net.
- As mentioned in the previous page, some properties for each blob have to be mentioned in the fuzzy decision maker (section C.2.1), for instance, BALL_NEAR, BALL_COMPLETELY_SEEN, BALL_OVER_LEFT (in the image frame)
- The head can be tilted HEAD_HIGH, HEAD_MIDDLE, HEAD_LOW
- If the head is LOW only the ball, lower side of the goal-in the upper part of the image-are visible, or the white walls-also in the upper part of the image-or the legs of another robot.

These are general rules for the relation between objects on the field. Combined with properties for each object, they could decide whether an object really is present in the image. An example of the properties of an object is the following table[2].

Figure	Left	Right	Up	Down	Position on the field
1	yellow	yellow	yellow	green	BACK_IN_YELLOW_NET
2	green	white	yellow	green	RIGHT_IN_YELLOW_NET
3	white	green	yellow	green	LEFT_IN_YELLOW_NET
4	green	green	yellow	green	MIDDLE_IN_YELLOW_NET
5	white	white	white	green	CLOSE_TO_WALL
6	green	green	green	green	ON_THE_FILED

Table C.1: Different poses of an orange ball on the field and, thereby, the different color combinations around it

As a clarifying example imagine the case where the own net is recognized with the accuracy BNET_HIGH from the three alternatives BNET_HIGH, BNET_MIDDLE

[2]This part was in the beginning of its implementation, to be used for filtering ball blobs outside a rule base but was removed later on, because of the uncertainties in the color segmentation. Some of these cases are illustrated in the table

and BNET_LOW and the ball is recognized with BALL_LOW. At the same time the neighborhood table says that the same ball_blob is BACK_IN_YELLOW_NET and we know that the head position is HEAD_LOW. Then we simply discard the recognized ball since the BALL_LOW contradicts the facts in the rule base, if we ever have foreseen this.

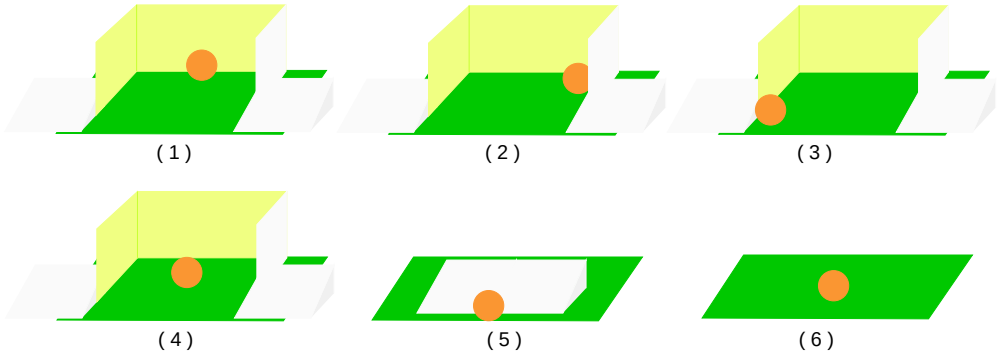


Figure C.2: Some of the positions a single ball can be found in, on the field. Numbers follow the same order as the rows in table C.2.1

The drawback of such a system could be the effort that must be put on generating as many rules as possible to cover all different cases that may occur.

On the other hand, as already known, there are rules that are not possible to define as Horn Clauses; C.2.1. For instance, the ball being close cannot be defined in any way by a Horn Clause. "'The object is close" is no statement that can be definitely right or wrong. That depends on what our definition of closeness is. Instead, we can choose to use the *fuzzy*[3] definition of the rules. A combination of the rules could then give a more precise result with more possibilities and less effort to put on defining the rules.

The method could be useful when defining the partial results from recognizing objects on the field to produce reliable final decisions about what objects really exist in every image. Concepts as HEAD_LOW, HEAD_HIGH, BNET_HIGH, BNET_MIDDLE, ..., as mentioned previously, are examples on vague statements that can preferably be represented by fuzzy statements.

There was a report from Mi-PAL Team Griffith, [26], showing their attempt on creating a rule base. It was obviously not finished at the time the report was published on the Internet and no later paper showed if it has been successful ([27]). There is though no reports that show a fuzzy object recognition system being used or even suggested the way it is here.

C.3 Merging Rectangles

If the blobs match horizontally or vertically, but are too far apart, we may better not merge them into one single blob; Figure C.3. We will have to parameterize the distance, compared to the size of the blob. This is easily done by multiplying the likelihood of the blobs matching in either direction, by $\frac{the\ distance}{the\ smallest\ one\ in\ that\ direction}$.

[3] For discussions on fuzzy logic, refer to [28], [20], and [34], and for some applications to [8], [17], [22], [23] [36], [37] and [38].

Multiplication will result in decreasing the matching possibility faster, when the blobs have an increase in their distances to each other.

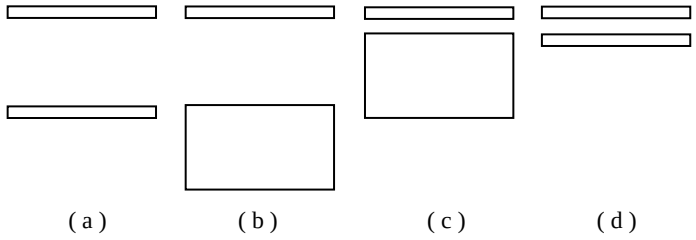


Figure C.3: Distance between blobs *plus* the size of the blobs, decide whether they belong to one and the same object. Here, the blob pares in c and d are more likely to be merged together than those in a and b

Assume that

$$dx_1 = max(x_1) - min(x_2)$$
$$dx_2 = min(x_1) - max(x_2)$$

Then, if $dx_1 \times dx_2$ is less than or equal to zero, the two blobs overlap horizontally- otherwise, the match is not accepted and they are supposed to belong to two different objects. As a measure for how well they match in this direction, we may use the least, out of dx_1 and dx_2, which is then added to 1 and divided by the width of the smallest blob

$$r = \frac{1 + min(dx_1, dx_2)}{min(width(blob_1), width(blob_2))}$$

the likelihood is then (Figure C.4)

$$likelihood = min(r, 1)$$

Now, the same is done for the vertical direction

$$dy_1 = max(y_1) - min(y_2)$$
$$dy_2 = min(y_1) - max(y_2)$$

and if $dy_1 \times dy_2 > 0$, then the minimum of these two is divided by the height of the shorter one. The result of this division is then subtracted from 1 which is the value of likelihood, in the vertical direction, if it is greater than zero. Otherwise, it is just put to zero and the blobs are supposed not to be matching. In the case that $dy_1 \times dy_2 \leq 0$, the blobs are certainly close enough to be parts of the same blob.

If there are more than two blobs with the same colors in the image frame, a procedure is followed, where

- The blobs are arranged by the number of pixels
- The largest blob is tested with all other blobs, using the method above

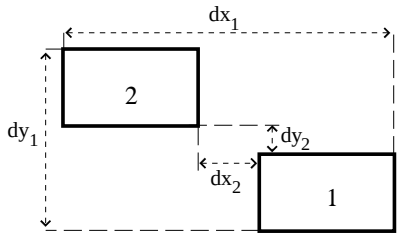


Figure C.4: The decision to merge two blobs depends on how their widths, heights and distances are related to the size of the blobs themselves

- It is then put aside and the same procedure is gone through, for all the remaining blobs

This way, the set of all blobs is divided into several, perhaps overlapping, subsets. The *intersection* of these subsets will produce new blobs, that hopefully are the objects we are looking for.

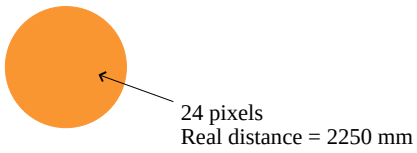
Note that the methods above are based on intuition and experiments and not on an actual, mathematical ground. Despite of that, this has shown to work well in simulations.

Appendix D

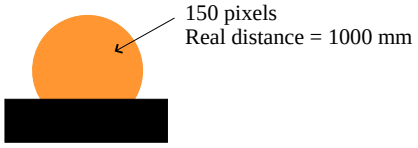
Evaluation Results

The first part, D.1, presents the result from evaluations made before the competitions and the next, D.2, shows images used for an extensive evaluation in a work conducted after the competitions but with the same vision module, [31].

D.1 Statistics from Evaluations

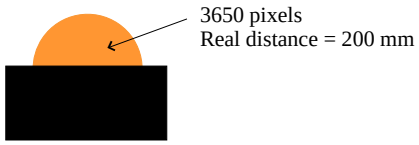


(a) $m = 2481.0$, $\mu = 2816.3$, $\sigma = 670.5$

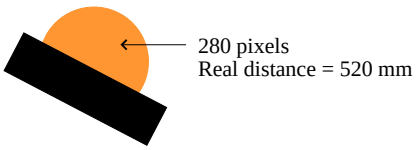


(b) $m = 1063.0$, $\mu = 1055.7$, $\sigma = 21.7$

Figure D.1: Long Range

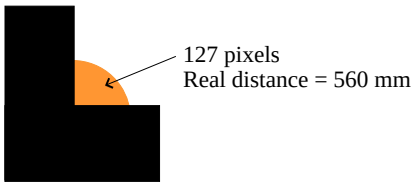


(a) $m = 186.0$, $\mu = 185.4$, $\sigma = 1.0$

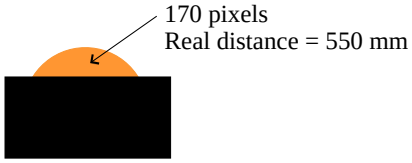


(b) $m = 480.0$, $\mu = 468.3$, $\sigma = 109.2$

Figure D.2: Partly Visible

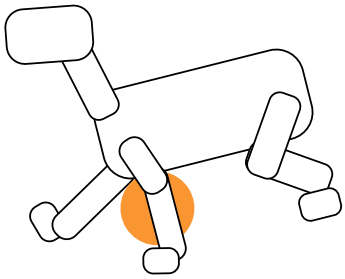


(a) $m = 425.0$, $\mu = 430.4$, $\sigma = 53.9$

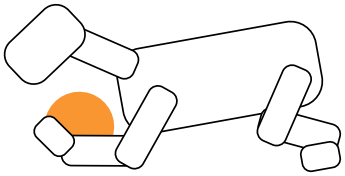


(b) $m = 539.0$, $\mu = 537.7$, $\sigma = 15.6$

Figure D.3: Hardly Visible

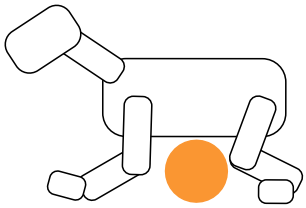


(a) Real D = 600, $m = \mu = 550$, $\sigma = 0$

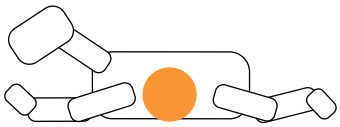


(b) Real D = 600, $m = \mu = 709$, $\sigma = 0$

Figure D.4: Occluded By Opponent's Leg

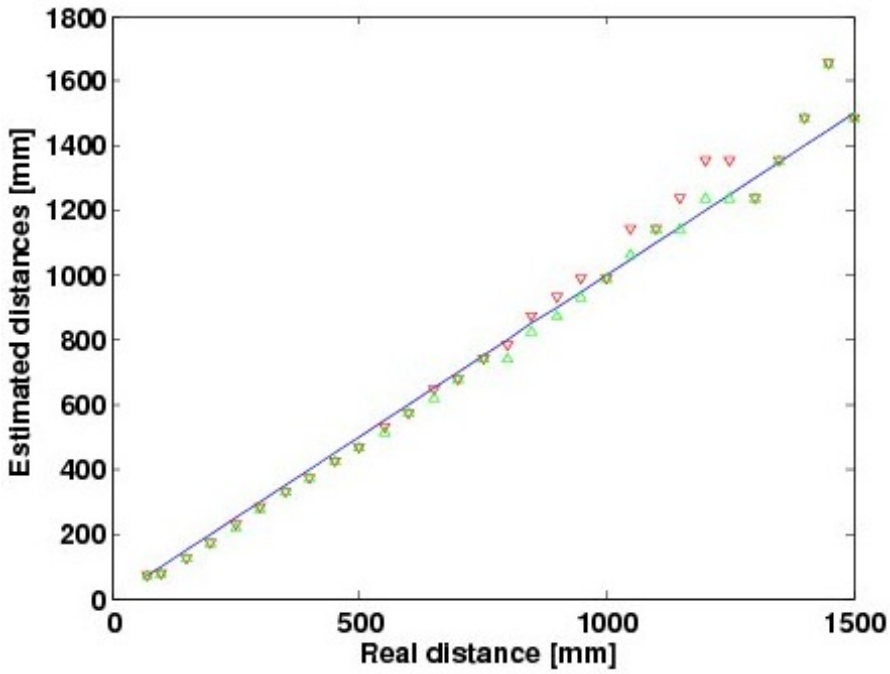


(a) Real D = 600, $m = \mu = 572$, $\sigma = 0$ Real D = 700, $m = \mu = 676$, $\sigma = 0$



(b) Real D = 600, $m = \mu = 583.5$, $\sigma = 16.3$

Figure D.5: Close To Opponent's Body



(a) Original Image

Figure D.6: The overall performance of the recognition module for estimation of the distance to the orange ball, within the range 70 mm through 1500 mm. ∇ : maximum estimated value, - : the real distance, $\triangle$: minimum estimated value

D.2 Images from Evaluations

The following images are a very good reference for evaluation of the vision module and the reason behind failure when running the TCC's robot system. Please note that titled images are only to be considered as examples. Head-tilt did not occur in the implementation of the original code of the TCC for the competitions.

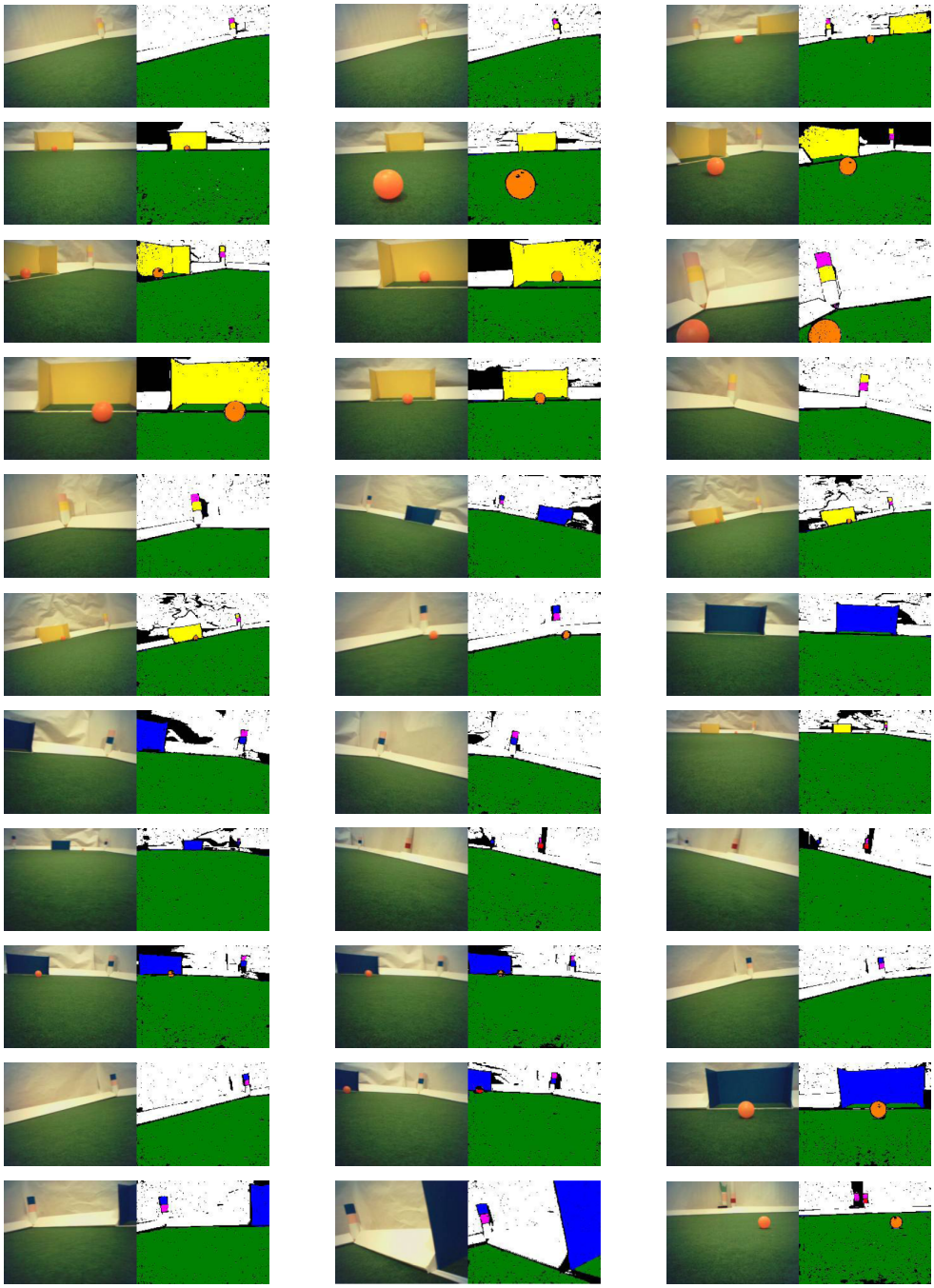


Figure D.7: There were many cases in which the images were correctly color-segmented.

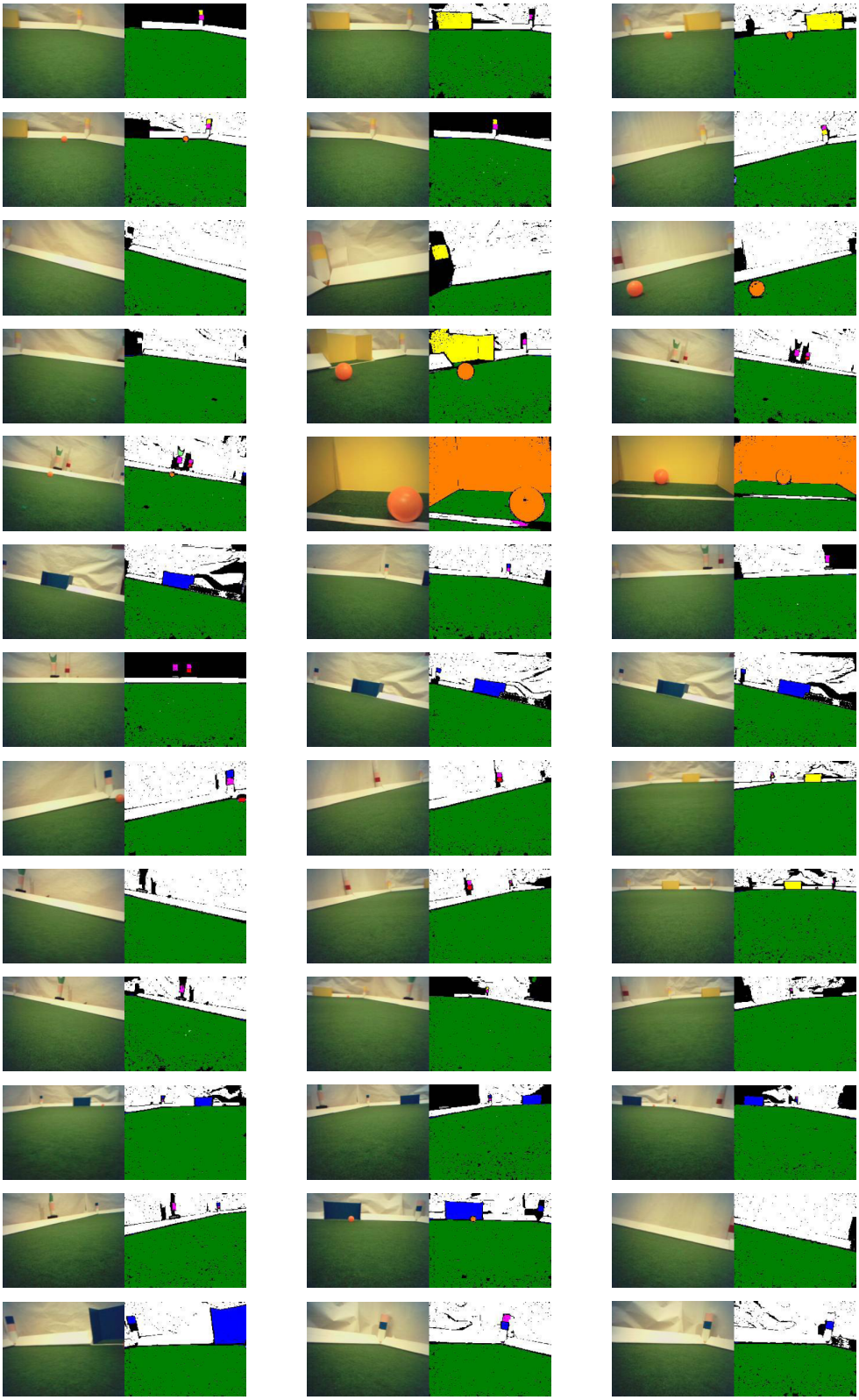


Figure D.8: There were even more cases in which the images were not correctly color-segmented. And this is just in normal lighting conditions.