Otto–von–Guericke–Universität Magdeburg

Diplomarbeit

Algorithm Development and Testing for Four Legged League Robot Soccer Passing

von

Florian Knorn
(* XXX. Dez. 1981 in Berlin)

14. September 2006

Eingereicht an die: Otto–von–Guericke–Universität Magdeburg
Fakultät für Verfahrens– und Systemtechnik
Prüfungsamt
Universitätsplatz 2
Postfach 4120, 39016 Magdeburg
Deutschland

Erstprüfer: Prof. Jörg Raisch
Zweitprüfer: Prof. Dietrich Flockerzi

Betreuer: Prof. Richard H. Middleton
Michael Quinlan

ARC Centre for Complex Dynamic Systems & Control
The University of Newcastle
2308 Callaghan, New South Wales,
Australia

Meinem Vater

Table of contents

Outline and objectives

For the Diplomarbeit *of Florian Knorn*
Otto–von–Guericke Universität Magdeburg
Fakultät für Elektrotechnik und Informationstechnik
Institut für Automatisierungstechnik

Algorithm Development and
Testing for Four Legged League
Robot Soccer Passing

Introduction

This project is motivated by experience at the University of Newcastle in participating in the RoboCup International, Four Legged Soccer competition. In this competition, to the proposers knowledge, practical algorithms for development of passing behaviours, whilst highly desirable, have not successfully been implemented. Previously, limited sensor information and kicking accuracy have restricted the ability of teams to develop higher levels of cooperative behaviour. It is now recognised that some of the lower level algorithms such as localisation and kick selection have developed to the point where higher level algorithm development is now needed. The proposed project tackles an important part of this in developing deliberate ball passing algorithms, both at a high level, and including relevant enhancements to lower level modules to facilitate this behaviour. The project will involve algorithm development, simulations studies, software implementation, testing and debugging on the Sony aibo.

The project

The project requires the student to make and integrate developments in several areas. Firstly, the student will need to make enhancements to the decision making and kick execution in the passing robot. This will involve the use of potential fields (possibly single dimensional) for strategic decision making, plus behaviour development for obstacle avoidance and kick orientation, followed by new kick executions with improved control of distance and timing if necessary. The second area of work is in relation to the kick receiver. The student will design new algorithms for positioning prior to the kick being executed, and correction of position whilst the ball is in motion. The student will also be required to conduct testing, redesign, debugging and tuning of existing extended Kalman filter algorithms for ball velocity estimation. The student should also develop and test ball "trapping" algorithms, capable of taking a moving ball, directly into a robot "grab". The third area of work is in relation to the cooperation needed to effectively complete the ball pass behaviour. This will include communication between the robots to coordinate behaviour, and also communication to improve ball position estimation by incorporating timing and velocity information derived from the kicking robot. The student will demonstrate the integration of these new algorithms in the existing software system to show deliberate team passing behaviour.

Magdeburg, 12 April 2006

Date of issue:	14 April 2006
Date of submission:	14 September 2006
Supervisors:	Prof. Richard H. Middleton Michael Quinlan
1st examiner:	Prof. Jörg Raisch
2nd examiner:	Prof. Dietrich Flockerzi

Prof. Dr.–Ing. J. Raisch	Prof. Dr.–Ing. A. Kienle	Prof. Dr.–Ing. E. Specht
1st examiner	**Head of institute**	**Examination committee**

Preface

There once was a swagman camped in a billabong
Under the shade of a Coolibah tree
And he sang as he looked at the old billy boiling
Who'll come a-waltzing Matilda with me ...

— The beginning of Banjo Paterson's
"Waltzing Matilda" song

Acknowledgments

First of all, I would like to thank Prof. Richard H. Middleton for all he has done for me, for the opportunities, the support, counsel and wise advice he has given me in the past six months. Rick, you are one of the few people I will always look up to, not only in an academic but also a personal way. Few people have achieved as much as you have, but even fewer managed to stay so friendly, supportive, caring, modest, yet funny along the way.

The person however I worked most closely with during my time here in Newcastle was Michael Quinlan, whom I also consider a dear friend of mine, not only for your important and indispensable help, support and patience with my *Diplomarbeit* as well as all those selfless small or big favours over the time — but especially for the awesome time you made me have, be it down in the lab, out at some rock concert, on the ski field or simply at lunchtime over in Jesmond.

When I mention Michael's name, I also have to thank my other lovely, charming and funny lab mates — Naomi Henderson, Robin Fisher, Steven Nicklin and Andrew Bevitt. I deeply appreciate all your help and assistance with my silly little (and big) problems along the way, and really enjoyed all the fun we had down here in our "windowless bunker". It was a pleasure to work in this friendly, relaxed yet productive atmosphere and I will greatly miss your company.

It was Dianne Piefke who took great care of me — especially at the beginning of my stay here — and helped me with a number of issues, such as getting the correct visum, finding a place to stay, enrolment at the university, etc. Thank you for making my arrival and stay on the virtually other end of the world such a smooth and hassle free experience.

Besides all the lovely people and staff at *International House*, the *NUsport Volleyball Club*, the *Body Jam* class as well as the *Bar on the Hill*, *Customs House*, *The Brewery* and *Fanny's* (pubs and bars where I enjoyed copious amounts of Australian beer) I would like to send my regards to the following people, colleagues, friends and *mates* for all their company, support, welcome distractions and fun time they made me have (alphabetical order, titles omitted): Aaron Wong, Brady Hardcastle, Carla Piefke, Christian Friedrich, Craig Murch, Craig Sharpe, Cynthia Angelica and Lorena Pais Soto, Damon Keen, Dawn, David and Todd Pike, Dimit Yalu, Donato Pantalone, Drew Mellor, Eva Barbay, Evelyne Niederberger, Heather Luciani, Helen Falk, Henning Bulka, Ingo Preiml, Jason Chua, Jennifer Neils, Jeram Hyde, Jonathon Kirk, Josie Milner, Joyce Wong, Julie Caudron, Kara Hanwright, Milan Derpich, Natalie and Alistair Lockhart, Ngoc Mai Tran, Katrin Wolf, Kenny Hong, Kristen Ochs, Kristina Stiller, Luke Murray, Mary Stokes, Natalie Lockhart, Nicola Mays, Oliver Keßler, Rebecca Martin, Rebecca Quattelbaum, Robert King, Samuel Barns, Scott Fitzgerald, Steffi Klinge, Stephan Chalup, Timothy Moore, Timo Tenhunen, Valérie Faustin, Verena Wiesen, William Ireland, ...

Finally, I could never overstate my gratitude towards my family and close friends back home. Thank you for supporting me over the years, for being there when I needed you, for your patience with me and, most of all, for your love.

* * *

Declaration of originality

I hereby declare that this thesis and the work reported herein was composed and originated entirely by myself. Information derived from the published and unpublished work of others is acknowledged in the text and a list of references is given in the bibliography.

Newcastle, Australia, 14 September 2006

Florian Knorn

Introduction

The present day RoboCup competitions have probably exceeded many of the initial expectations made in the early 1990s when it was founded. Objectives like drawing attention to and promoting research and education in the field of robotics and artificial intelligence have certainly been achieved.

Over the years, several leagues have been formed inside RoboCupSoccer. Using the somewhat visually appealing Sony "entertainment robot" aibo, [7], it seems to be the Four Legged League in particular that is getting much media attention lately. Interestingly enough, this is so far the only league that uses standardised hardware, i. e. untouched third party robots. This means that only the software separates the different teams.

We can safely assume that the boundaries of the provided hardware are pushed to their limits. Also — as every year after the RoboCup tournament all the participating teams are to publish, share and comment on their code in their yearly team reports, [4] — most of the teams are roughly at the same level of capabilities. For instance, the maximum walking speed of the robots appears to have plateaued over the last years, and there is almost no difference between the teams any more.

It is clear that, with little room for significant improvements on the physical side but sophisticated low level behaviour at hand, the general focus shifts more and more towards the information processing capabilities and the high level behaviour of the robots. For example, efforts are made to further refine the vision data processing or introduce more complex team behaviours and strategies.

Looking at team tactics in particular, no team in the Four Legged League (to the best of our knowledge) has successfully implemented deliberate in–game passing behaviour. Reasons for that may be the limited information each individual player has available and/or the precision of that information. Besides, each player acts fully autonomous, i. e. there is no overall decision making instance — which would make the coordination of such complex behaviour much easier. Also, as mentioned earlier, mature and reliable low level capabilities were of primary concern.

As passes are an indispensable element in almost every team sport they would unquestionably enrich the quality of robot soccer and constitute a major advantage over other teams. It is the development and design of such desired passing behaviour that lays at the heart of this thesis. This thesis will touch a number of diverse areas of modern science, such as artificial intelligence, systems theory, control theory, fuzzy logic or computer science, having the following structure:

The first chapter will give an introduction to the hardware of the robot, briefly commenting on the input, processing and output equipment provided. We also state a few important rules from the official RoboCup rule set which are most relevant for this thesis.

Chapter 2 primarily focuses on the NUbot software in use, developed at the University of Newcastle, Australia, [3]. As our passing algorithm is integrated deeply into the existing frame work — and relies heavily on a number of low level capabilities — we shall describe each of the software modules run on the robot. At the end of the chapter, we discuss two of the most important modifications done to the existing software in the course of this project.

In the third chapter, we give a detailed explanation of the design of our algorithm that makes the actual passing decision — whether to pass at all and, if so, in what direction to play it. We also lay out the subsequent communication and cooperation between pass sender and receiver that is required to coordinate the actual passing action.

The objective of the last chapter is to show in simulation and real practice the workings and some of the features of our implementation of the passing algorithm, as well as highlight the potential gain to the overall team performance.

The appendices hold a proof, additional plots, the python™ source code of the developed passing algorithm as well as the concrete choice of parameters used in the tests.

CHAPTER 1

Hardware used

We will give a quick introduction to the robots' hardware, reviewing its input, processing and output equipment as well as mention a few RoboCup rules.

1.1 Introduction

This short chapter shall serve as a description to the NUbots hardware and also list some of the most important rules of the Four Legged League. Again, one particularity is that the robots used are not self built by the different teams. Rather, an already existing "entertainment robot", the Sony aibo ERS–7 is used, as shown in Figure 1.1 below.

Using the same hardware has a number of advantages: On one hand, teams do not have to worry about hardware issues, which allows smaller teams and research institutes (that might not have a large workshop backing them up) to participate in the competition, and also invites institutes that are more interested in software and algorithm development to get involved. On the other hand, with everybody using the same hardware, different financial situations

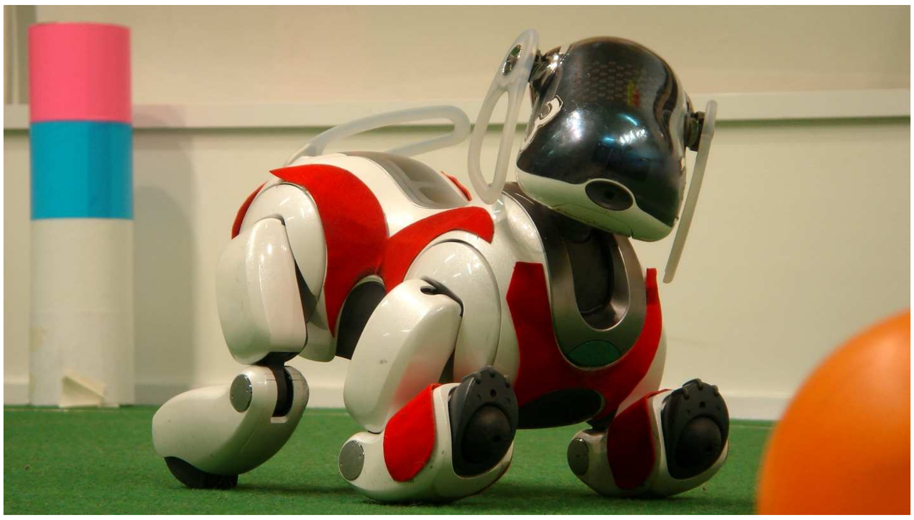


Figure 1.1: *A NUbot (playing Red Team) ready for action.*

are mitigated (i. e. as long as you can afford the retail price of the robots, you can compete on equal grounds — this is different in other leagues, where, loosely speaking, spending more money and using a bigger motors will make you win).

So let us take a look at the hardware and later at a number of rules from the official RoboCup rules.

1.2 Hardware

The robots are, in their standard stance, about 27 cm long, 20 cm wide and 23 cm heigh to give in idea of their size. In the following, we shall group the different components into input–, processing– and output related parts. The following specifications can be found for example in the robots' user manuals, [6].

1.2.1 Input

To perceive the outside world, each aibo ships equipped with

- a video camera in the head,
- infra–red sensors in the snout and on the chest,
- three touch sensitive buttons on the back, one on top of the head and one underneath the chin,
- four touch sensors in the paws,
- a microphone per ear,
- three accelerometers (x–, y– and z–axis),
- a vibration sensor.

The video camera delivers pictures at 30 frames per second and is probably the most important input element. Each camera frame processed consists of a 208×160 pixel YUV picture unfortunately covering only a very narrow a field of view ($55°$ horizontally, $45°$ vertically). Worse, the pictures are of poor quality, rather noisy, heavily vignetted and extremely sensitive to the particular lighting conditions, all that requiring a significant amount of software post processing. See Figure 1.2 on the next page for an example of the quality — a shot similar to the last figure, but taken directly from another robot.

We do not use the long range infra–red sensor (in the head) to measure the distance of other objects as the results would not only be significantly less precise and reliable than those that can be obtained via image processing, but also as it would be difficult to attribute them to particular objects. We only fall back to their readings if the robot is very close to the goal and his entire field of view is filled with goal colour (resulting in the vision system not being able to calculate the distance correctly) or when dribbling and trying to dodge defenders. However, the close range infra–red sensor in the robot's chest is used much more frequently when determining whether the ball has been grabbed successfully, and the to keep checking whether it is still in possession.

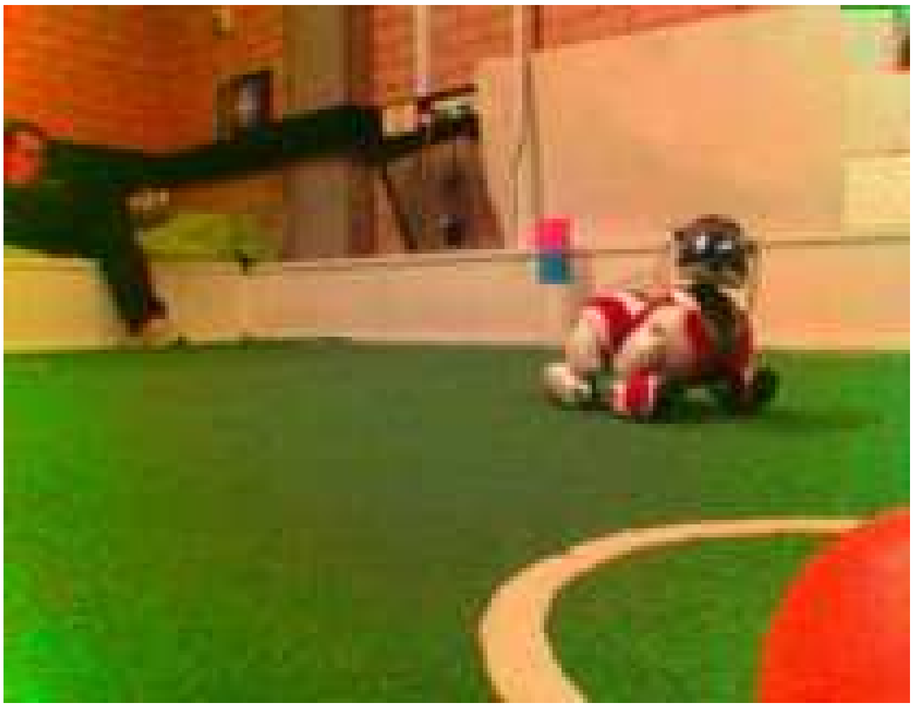


Figure 1.2: *The same arrangement as in Figure 1.1, this time taken with the camera of another robot. Note the low resolution, the pixel noise and the vignetting near the corners of the picture.*

Neither the paw sensors, the sensor under the chin, the vibration sensor nor the microphones are used in our set–up.

1.2.2 Processing

Now that the robot can more or less sense the outside world, this data needs to be processed. For that, the robots feature

– a 64 bit RISC processor at 576 MHz,

– 64 MB of RAM,

– an IEEE 802.11b wireless interface,

– a Memory Stick™ reader.

All the software resides on a Memory Stick™, and due to Sony's policies, only 16 MB Memory Sticks™ can be used, requiring a very careful and responsible handling of resources. The robot's firmware, operating system as well as all the device drivers are provided by Sony and cannot be changed or customised, confining all work to relatively high level programming.

The computationally more expensive task are written in the C programming language using a Software Development Kit provided by Sony. However, as shutting down a robot, recompilation on the PC, transfer of the binaries to the Memory Stick™ and rebooting the robot takes at least two minutes, debugging or parameter tuning can be very tedious. For that reason, most of the behaviour code, and code that needs a lot of situation dependant tweaking was ported to the much more high level programming language python™, [8], which is also run on the robot. Here, files and changes can be uploaded almost instantly via File Transfer Protocol (FTP) over the wireless interface, and only the python™ interpreter needs restarting, which takes about two seconds.

Although python™, being an scripting language, is much slower than compiled C++ and thus anything but suitable for computationally more involved tasks, it is a major advantage of having development cycles cut down from two minutes to a couple of seconds. Also, one has always the option of porting functions back to C++ once they run as desired (fortunately interfacing C++ and python™ functions is very easily done).

1.2.3 Output

Once the "brain" has processed all the inputs it wants to interact with the outside world. More or less imitating a dog, the robot has

– three motors per leg,

– three motors to move the head,

– two motors to move the tail,

– a motor to open the mouth,

– a motor per ear,

– a speaker (plus a small system beeper),

– a large amount of LEDs in many different colours[1].

Each leg can be rotated and spread out at shoulder level, as well as be bent at elbow level. The head can be rotated in the horizontal plane, pitched up and down as well as moved up and down at neck level. We do not use the robots tail or ears,[2] nor the speaker. However, the mouth is opened when trapping or dribbling the ball as it allows for a slightly higher head position so that the robot gets to see more of the field (again, the camera only delivers a very narrow field of view, vertically and horizontally).

The numerous LEDs are mainly used for debugging purposes, such as indicating the robot's overall status, whether it can see the ball, what team it is playing for (software wise), and the like.

With these relatively rich lists of parts it is easy to understand why these entertainment robots are now at the centre of serious research. Although the processor may sound quite powerful, the large amount of tasks it has to take care of (to start with, just consider processing 34,000 pixels thirty times per second) demand for extremely efficient algorithms and software design.

[1] It is an *entertainment* robot after all ...

[2] In fact, we usually take the rubber ears and tail off as (for us) they do not serve any other purpose than randomly falling off.

It is precisely this required ingenuity that is one of the key motivations behind RoboCup. Let us now take a look at some of rules involved in the Four Legged League.

1.3 Some RoboCup rules

The following rules (considered the most relevant for this thesis) are a subset of the official RoboCup rules that can be found in the official rule book for the Four Legged League, [5].

1.3.1 The field

A graphic of the field is shown in Figure 1.3 below. The field dimensions are 540 cm × 360 cm, with a 30 cm resp. 20 cm margin beyond the base– and side lines. The goals are 80 cm wide, 30 cm deep, and the uniquely colour coded beacons — indispensable for the robot's localisation on the field — are 40 cm heigh, sit 15 cm off the sidelines and 135 cm from the half way line.

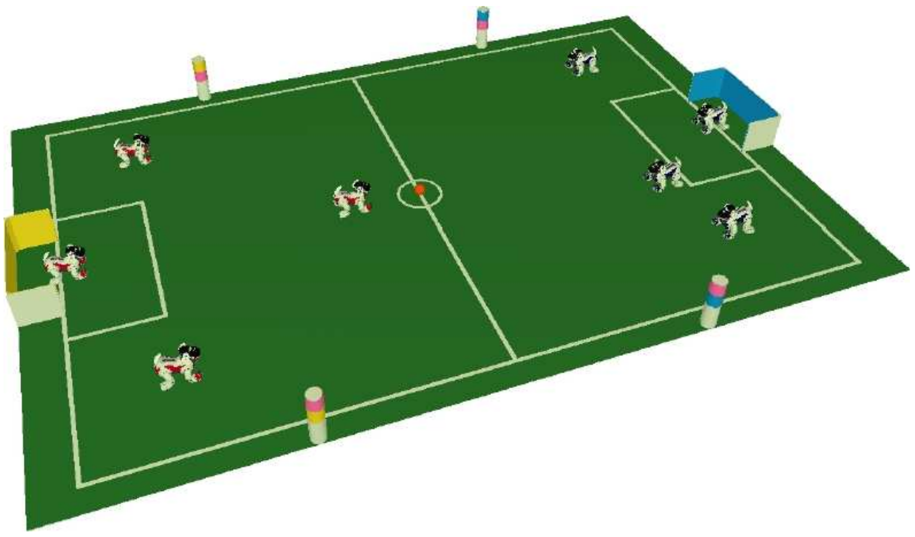


Figure 1.3: *A RoboCup Four Legged League soccer field, [5]*

Our absolute, or "fixed" coordinate system depends on the direction of attack. The x–axis points "up–field", that is, the goal keeper (usually) looks along the fields x–coordinate, the y–coordinate pointing to his left. So in Figure 1.3 for instance, if the team plays Red, i. e. defending its own yellow goal and attacking the blue goal, the field's x–coordinate would point towards the right hand side of the figure, the y–coordinate upwards.

The origin of the coordinate system lays always in the centre of the field (kick–off point). A player's heading ϑ corresponds to the (signed) angle measured in the mathematical positive sense from the absolute x–axis of the field. So, for example, if the Red Team robot near the kick–off point would look down towards the caption of the figure, it would have a heading close to $-90°$.

1.3.2 Game process

The game consists of two ten minute halfs, with a ten minute half time break. Each team has four players, one of which is the designated goal keeper. He is the only player allowed inside his own penalty box. Any other player that enters it with more than two legs is called an illegal defender, and will be penalised by being taken out of the game for 30 seconds (it will be put back into game on the middle of a side line).

Any player grabbing the ball must let go of it (or kick it) within three seconds. Goal keepers are allowed to hold it up to five seconds if they have at least two paws inside their penalty box. Robots are allowed to leave the 5.4 m $\times$ 3.6 m metre field area, but most not leave the 4 m $\times$ 6 m carpeted area (or get a 30 second penalty).

If a robot that is not actively heading for the ball (intentionally) blocks the way to the ball for another player then this act is termed obstruction and will also invoke a 30 second penalty.

Concerning the wireless communication of the robots, the team is allowed to use in total up to 500 kbit/s. This is relatively little bandwidth as it is shared by the four robots and includes all the necessary protocol overheads.

With these few important rules in mind and the abovementioned hardware at hand let us now introduce the software that is run on the robots — transforming a standard aibo into a NUbot.

CHAPTER 2

Software used

We will give an overview of the four functional modules of the software used in the robots, and describe some preliminary modifications done to them.

2.1 Introduction

Although the level of present day robot soccer is nowhere near being able to seriously compete against human players, this second chapter should give a small insight into the complexity of the software already involved at this relatively early stage of the feat.

The hardware, firmware and operating system being provided by Sony, the actual software has to be entirely developed by the different teams competing in the RoboCup. As it is imperative for most complex problems to be broken down into smaller, more elementary tasks, the NUbot software was divided into four functional modules, namely Vision, Localisation & World Modelling, Behaviour and Locomotion. We shall describe these modules one by one, paying particular attention to the second and third module. There, a few key modifications have been made prior to working on the passing behaviour.[1] These modifications will be laid out in the last section of this chapter.

The bandwidth restrictions set by the rules limit the amount of communication that can take place between the robots. In fact, in our software, the NUbots only exchange every fifth frame (that is about every 166 ms where they believe the ball is as well as their own position (including a measure of uncertainty about both). This is particularly useful in situations where the robot cannot see the ball, or to determine the player's own ideal position on the field, based on the game situation as we shall see later on.

Although a much more in–depth description of the different software modules can be found especially in last years team report, [17], and the ones from previous years (see NUbot homepage [3]), we shall briefly present them here to provide some of the background on which this thesis' work stands. Let us start off with the Vision module.

[1] It is important to mention that some of these modifications are not directly related to deliberate passing behaviour, the topic of this thesis, and could be considered off–topic. However, considering the amount of time spent on them, their relevance and contribution to the overall game performance (especially a few weeks before the 2006 RoboCup) author and supervisors consider them worth mentioning here.

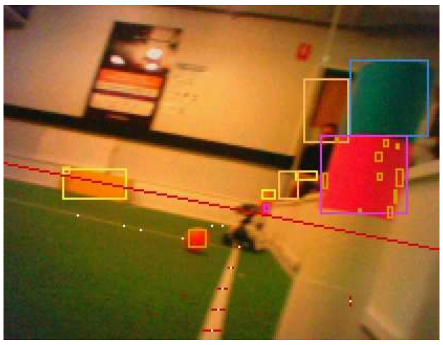
2.2 Vision

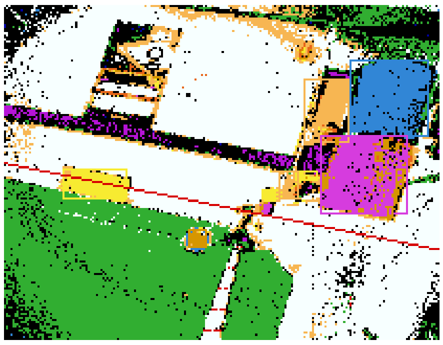
The vision module is most elementary to the robot's performance on the field. Even with the most sophisticated behavioural algorithms, the robot would be rather useless if it was left blind. The following section gives an overview of the steps involved in the image processing and the techniques used to detect different objects on the field.

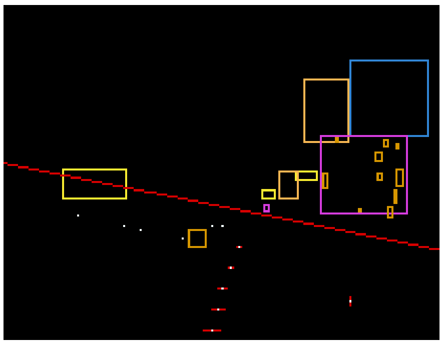
2.2.1 Algorithms used

To perceive the outside world, the robot evaluates thirty pictures per second, delivered by the built in camera. It is quite common among teams in the RoboCup to reduce the information by a sequence of different algorithms:

$$\begin{tikzcd} \boxed{\text{Image Pre–filter}} \arrow[r] & \boxed{\text{Colour classification}} \arrow[r] & \boxed{\text{Blob formation}} \arrow[r] & \boxed{\text{Object detection}} \end{tikzcd}$$


(a) Initial picture


(b) Colour classified


(c) Blobs formed

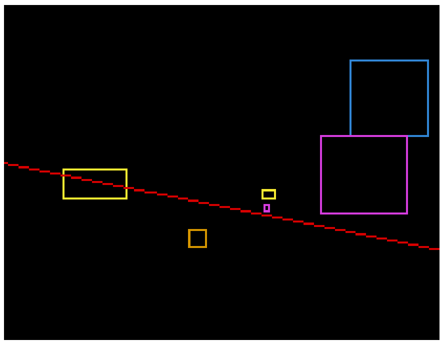

(d) Objects detected

Figure 2.1: *The original picture (a) being colour classified (b), having blobs formed (c) and objects recognised (d). Note that the blobs found and shown in (c) have been overlayed on the first two pictures. Objects detected are the yellow goal, the ball and two beacons.*

After pre–filtering (see below), the colour of each pixel is narrowed down to one out of sixteen colours[2] using a lookup table. Then "blobs" are formed by combining adjacent pixels of same colour.[3] The last step then recognises objects, such as the ball, goals, beacons, and stores the information about them, such as bearing and distance, in the global Object Array, which is used by the rest of the software. Examples for each step are shown in Figure 2.1.

2.2.2 Pre–filtering

Before being handed on to later algorithms, all the pictures pass one or two filters. The first one, always applied, tries to compensate for the lens' vignetting effect (pixels getting darker the further away they are from the optical axis of the lens).

The second filter de–skews the image if the software attempts to fit a circle to the ball (to determine bearing and distance more precisely). Skewing occurs when the camera is panned at high speeds. The way the CMOS sensor works, it reads the pixels horizontally, line by line, with the image not being held for each frame, but continuously re–exposed. This can result in the image shifting considerably by the time the scan has passed from the top left pixel to the bottom right. Knowing how fast the pixel scan occurs and how fast the camera is panning, one can compensate for these effects, as shown in Figure 2.2 below.

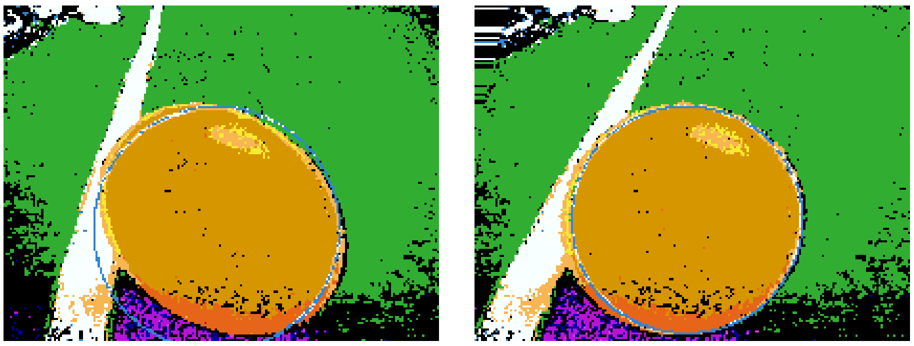

(a) *Initial picture* ***(b)*** *De–skewed picture*

Figure 2.2: *Demonstration of de–skewing an image. Observe how much the circle fitting algorithm profits from the correction.*

2.2.3 Object recognition

It is less relevant for us to present more details on the intermediate steps of the image processing. What we should remember from this important software module is, however, which objects can be recognised, that these objects are

[2] Just to name a few: ball orange, beacon blue, shadow blue, white, green, ...

[3] Before being passed on to object recognition, blobs may however be rejected, or modified by combining adjacent blobs based on extensive heuristics.

recognised on a per–frame basis and that different techniques are used to detect different objects.

In this year's code, the following objects could be recognised by the software:

- ball,
- robots,
- beacons,
- goals,
- goal posts,
- corner points

Some of these objects having considerably different shapes, it is clear that different algorithms have to be used to estimate the distance to the objects. The ball distance and bearing for instance is determined by using a least squares based circle fitting algorithm (as can be seen, for example, in Figure 2.2), where beacon distance is calculated from the amount of separation between and the size of the two colour blobs that characterise the beacon.

When the software has detected an object and was able to determine its distance and bearing, it adds that object along with that information to a globally accessible array (this array is flushed at the beginning of each frame, as, again, Vision works on a per frame basis). Other modules can then query that array and make use of the objects found in it — as our next module will do very frequently.

2.3 Localisation & World Modelling

With the different objects that the robot managed to recognise in one frame stored in the global Object Array, the robot can use that information to determine its own position, as well as that of the ball and the ball's velocities. Let us take a look at how this is done.

2.3.1 State variables

In our philosophy, the most important things for a robot to consider should be its own position as well as the ball's position and velocity. So, throughout the years, the NUbot software always focussed on a state vector containing the following seven variables:

$$\boldsymbol{x} = \begin{pmatrix} x \\ y \\ \vartheta \\ x_{\mathrm{b}} \\ y_{\mathrm{b}} \\ u_{\mathrm{b}} \\ v_{\mathrm{b}} \end{pmatrix} \tag{2.1}$$

The first three states are the robot's own x– and y–coordinates as well as heading, and the last four the ball's position and velocity components.[4] All

[4] A figure showing all these variables can be found on page 22.

lengths are measured in centimetres, the speeds are given in cm/s. Using digital hardware and getting new information at the frame rate of the camera, the states are time discrete, the sampling time being 1/30 s.

As the measurements from the camera are not overly precise and often missing (when the line of sight to the ball or a beacon is obstructed for instance), many teams use an Extended Kalman Filter to estimate the above states. The following section will describe how the currently implemented filter was designed.

2.3.2 Extended Kalman Filter

We shall now introduce the Kalman Filter used to estimate the robot's own position as well as the position and velocity of the ball. See [11] or the great introduction in [18] for more background information on the topic. Technically, our filter has to be called "extended" as it contains non–linear elements in the time and measurement update functions.

Data used

To estimate the robot's state vector (2.1) at frame k, the Extended Kalman Filter is given distance and bearing of every object Vision has reported in that frame. After extensive tests of the camera and the algorithms used, we are able to quantify the confidence about these measurements (for instance, if the ball is very far away, the limited resolution of the camera will have a negative impact on the performance of the circle fitting algorithm, thus lowering the precision of the distance measurement), which is a prerequisite for the filter.

Notation

In the following, we shall always use $\boldsymbol{x}_k$ (or sometimes $\boldsymbol{x}(k)$) to denote the state vector in frame k. The concept of *a priori* states is common in literature, and we shall use it here as well. It refers to variables that are pure estimates, not yet "corrected" by real measurement data. We mark those variables with a superposed minus, where as variables that are (corrected) estimates of others will be marked by a hat. So $\hat{\boldsymbol{x}}_k^-$ is the *a priori* estimate for the state vector $\boldsymbol{x}$ in frame k and $\boldsymbol{P}_k^-$ will be the *a priori* error covariance matrix (a diagonal matrix quantifying the filters uncertainty about its estimates of the different states).

Let $\boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1})$ denote the function that estimates the state vector of frame k based on the previous frame's estimate (together with information about the robots movement between the two frames, coming from Locomotion):

$$\boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}) = \hat{\boldsymbol{x}}_{k-1} + \begin{pmatrix} d_\mathrm{f}(k)\cos\big(\hat{\vartheta}(k)\big) - d_\mathrm{l}(k)\sin\big(\hat{\vartheta}(k)\big) \\ d_\mathrm{f}(k)\sin\big(\hat{\vartheta}(k)\big) + d_\mathrm{l}(k)\cos\big(\hat{\vartheta}(k)\big) \\ \hat{\varphi}(k) \\ \hat{u}_\mathrm{b}(k)/30 \\ \hat{v}_\mathrm{b}(k)/30 \\ 0 \\ 0 \end{pmatrix} \tag{2.2}$$

where $d_\mathrm{f}(k)$, $d_\mathrm{l}(k)$ and $\varphi(k)$ are the amount of forward– and sidewards motion as well as turn in frame k. The ball position changes by its current speed divided by the time passing between frames. We define $\boldsymbol{A}$ to be the Jacobian of this function with respect to the different states.

Furthermore we introduce the diagonal matrix $\boldsymbol{Q}$ as the model covariance matrix. It expresses our confidence in model and odometry data, as well as ensures that the estimate variances increase when no objects have seen for a certain amount of time.

To facilitate the software implementation, the filter has been set up to use two *scalar* measurements z_i, with the index i standing for either a distance $(i = 1)$ or a bearing $(i = 2)$. For that reason, the Kalman Gain $\boldsymbol{j}_{i,k}$ will be a column vector, and r_i, the respective measurement's variance, a scalar.

We also need to define two non–linear functions $h_i(\hat{\boldsymbol{x}}_k^-)$ which relates the current *a priori* state estimate to the measurement z_i, i.e. calculates the expected distance and bearing to an object o based on $\hat{\boldsymbol{x}}_k^-$. This boils down to simple geometry:

$$h_{1,k}(\hat{\boldsymbol{x}}_k^-) = \sqrt{\big(\hat{x}^-(k) - x_o(k)\big)^2 + \big(\hat{y}^-(k) - x_o(k)\big)^2} \tag{2.3a}$$

$$h_{2,k}(\hat{\boldsymbol{x}}_k^-) = \tan^{-1}\left(\frac{\hat{y}^-(k) - y_o(k)}{\hat{x}^-(k) - x_o(k)}\right) - \hat{\vartheta}^-(k) \tag{2.3b}$$

again, where $x_o(k), y_o(k)$ are either the known (and never changing) coordinates for beacons and goals, or, in case of the ball, the *a priori* estimate of its position $\hat{x}_\mathrm{b}^-(k), \hat{y}_\mathrm{b}^-(k)$.

Finally, let $\boldsymbol{c}_{i,k}^T$ be the Jacobians with respect to the different states (resulting in row vectors).[5]

Time Update

Following the classical Extended Kalman Filter approach, let us first write down the time update equations:

$$\hat{\boldsymbol{x}}_k^- = \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1}) \tag{2.4}$$

$$\boldsymbol{P}_k^- = \boldsymbol{A}_k \boldsymbol{P}_{k-1} \boldsymbol{A}_k^T + \boldsymbol{Q}_k \tag{2.5}$$

These equations give us an *a priori* estimate of the state vector for the new frame k as well the *a priori* covariance of this estimate, based on the previous frame's state estimate $\hat{\boldsymbol{x}}_{k-1}$ (and odometry data) as well as error covariance matrix $\boldsymbol{P}_{k-1}$.

Measurement Update

In this step, we include the real measurements to correct / improve the *a priori* estimate:

$$\boldsymbol{j}_{i,k} = \boldsymbol{P}_k^- \boldsymbol{c}_{i,k}\left(\boldsymbol{c}_{i,k}^T \boldsymbol{P}_k^- \boldsymbol{c}_{i,k} + r_k\right)^{-1} \tag{2.6}$$

$$\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + \boldsymbol{j}_{i,k}\big(z_{i,k} - \hat{z}_{i,k}\big) \tag{2.7}$$

$$\boldsymbol{P}_k = \left(\mathbf{id} - \boldsymbol{j}_{i,k}\boldsymbol{c}_{i,k}^T\right)\boldsymbol{P}_k^- \tag{2.8}$$

[5] They are shown calculated on page 20.

where $\hat{z}_{i,k} = h_i(\hat{\boldsymbol{x}}_k^-)$. As we are using scalar measurements, distance and bearing update the states separately (in the actual C++ implementation, a `for`–loop is used). Also, an outer loop iterates through the different objects found in the Object Array in the current frame.

The order in which the updates are done (that is distance measurement first, then bearing — or *vice versa* — or in what order the objects are used from the Object Array) does have an impact on the estimates. However, it is believed that the impact of this correlation is minimal, as no particular attention is given to this issue in the implementation, or the team reports.

Working principle

With the above equations, how does the filter actually work? As we can see, these equations are recursively defined. This requires well defined initial states. These are set to be robot and ball sitting almost at the kick–off point, with a variances covering the entire field, the robot looking straight ahead in offensive direction, with a variance of a whole turn. The ball is assumed to have zero velocity (but, again, with a variance around the maximum speed the ball could travel with under normal circumstances).

So, robot and ball are *assumed* to be in some fixed initial state, but with these massive variances, the filter will pay hardly any attention to these arbitrary initial conditions. Rather, it will attribute a lot of weight to the measurements. This can be seen in (2.6) on the preceding page, where large elements in $\boldsymbol{P}_k^-$ will cause large entries in $\boldsymbol{j}_{i,k}$, which in turn will put a lot of emphasis on the difference between expected and real measurement in (2.7). If the difference is large (an indicator, that estimated state and real states are very different), a strong correction to the state estimate will take place, based on this difference.

If in turn, the filter has become quite confident about its state estimate over a period of time, i. e. entries in $\boldsymbol{P}_k^-$ are very small, a rather noisy measurement (indicated by a large $r_{i,k}$) will only have little impact on the estimate, as the respective entries in $\boldsymbol{j}_{i,k}$ will be very small. This is the main idea and advantage of using an Kalman Filter. It can dynamically shift its “trust” in either its own estimate, or external measurements, based on the variances of both.

As mentioned in the introduction, an important modification has been done to the above set–up. For the flow of the document, these modifications are described in the last section of this chapter. Let us for now assume that the filter works well, and the robots now “know” where they are on the field, what direction they are facing, where the ball is and how fast it is moving in what direction, in order to move on to the presentation of the next software module.

2.4 Behaviour

As we mentioned in the introduction, with most of the low level skills already pushing the boundaries of what is physically possible with the provided hardware, the attention in the RoboCup shifts more and more towards team behaviour and strategy. Due to the complexity of the behaviour code we shall also just briefly lay out some key facts and ideas involved in it.

2.4.1 Individual behaviour

Before commenting on team dynamics, let us mention some of the important individual behaviours the robots can show.

Move to point

This behaviour allows for four different variations. The robot can run directly to a point (with no particular regard to its own orientation) or it can move to a point always keeping a specified heading (such as facing the ball, or the goal for instance). It can also go to a point either facing forward or backward, and then turn to a specified heading, once it has reached the destination.

Chasing

An important feature of the robots is the chase mode. In this mode, the robot will run after the ball as fast as it can, and attempt to grab it, once it has gotten close enough.

Contrary to most of the other teams at RoboCup, the NUbot chasing code will *not* slow down the robot prior to a grab. Although this results in occasional "clumsiness" when trying to grab the ball (that is, the robot might just knock the ball away with its chest, failing to execute the grab motion at the exact moment required for a successful grab), it is the great speed advantage that has given us an important advantage in many situations.

Another key advantage of our code over other teams was the dodge or avoiding feature while chasing. If a robot is chasing a ball, and the direct line of sight is obstructed by another robot, the code will add a horizontal, strafing component to the walk. This usually results in a nice arcing motion around obstacles, hardly influencing the forward speed at all.

Grabbing

Once the ball is in a specific area in front of the robot, a grab can be triggered. This depends, of course, on the exact location of the ball, which is crucial to a successful grab, as well as a number of sanity factors (such as the robots tilt for instance — if it is leaning badly to one side, the grabbing motion will usually not be successful).

An internal timer is started after a successful grab in order to respect the three second rule. This timer will make the robot let go of the ball, or kick it, depending on its heading, if the ball has not been kicked earlier.

Dribbling

When the robot has successfully caught the ball, it must be able to turn freely and move around, with good holding of the ball being very important. This is achieved by a slightly loose "hold" of the ball between the front legs and under the chin, with the mouth open to allow for the highest possible head position (in order to minimise the impact on vision distance caused by the lowered head position).

Lining up goals

Moving on, let us assume the robot has dribbled towards a position where it is sensible to take a shot at the goal. The actual lining up involves two steps: first, a rough lining up (that is turning in the right direction to face the goal at all), then a much more fine tuned aiming at the largest gap between the goal posts and the goal keeper.

The rough lining up is done by turning towards the goal, either based on vision, or just pure localisation. Determining the gap to be aimed at is then done by using scan lines (checking the colour of pixels), which are run down vertically at fixed intervals.

Here also, the robots can dodge obstacles, prior to kicking. This is very useful in avoiding defenders and has been a great advantage on numerous occasions.

Kicking

Over the years, a large variety of kicks have been created. However, only a fraction of them are actually used. They were selected based on

- whether they were reliably executable by *every* robot[6]
- whether they allowed for precise aiming and/or fast execution
- their range to be able to kick at different distances.

For our passing later on, we chose one particular kick which was considered to be the most reliable (being consistent and predictable).

Diving

If the ball is calculated to pass the player in close proximity, a dive may be triggered. The robot will spread its front legs in order to cover a wide area. This can be done by field players, but is a key movement for the goal keeper. Recently, more controlled dives have been introduced, where the robot only dives to the side where the ball is thought to pass by.

This concludes our summary of deliberate individual behaviours. We shall now see, how our AIBO's work together as a team.

2.4.2 Team behaviour

As in real soccer, it is not only the skill of the individual players, but the collective effort, the team play that makes the difference. But although there is some communication between the robots, it is limited to telling the other robots where the ball and oneself is believed to be. There is no *"I'll go for it"* or *"Watch your six"* in our game,[7] there is no captain in the team telling the other players what to do (which, however, has been implemented by other teams on the RoboCup, but has been decided against in our team). All coordination follows only from the decisions made by each robot individually.

[6] And not only the "physically fittest", due to wear and tear of the joints and gears.

[7] Certainly, this will — has to — change at some stage in the future, if robots are finally to compete against humans in 2050.

It is the aim of this thesis to add some more cooperation to the team by implementing deliberate passing behaviour that *involves* communication and coordination between the robots. Before moving on to that, let us first mention the different formations the team can take, and how the robots will decide on the best position for them.

2.4.3 Formations

With only four players per team on the field, one of which is the goal keeper, there is not great deal of formation tactics involved here. Basically, the team can take two offensive and one defensive formation. These are taken depending on which half of the field the ball is in. When attacking (ball in the opponents half), two robots will go front left and front right, a sweeper staying back in the own half, in line with the ball.

When the ball enters the own half, two robots will fall back, one remaining in the opponents half (again in line with the ball). Later, an additional defensive formation has been added, where a sweeper will position itself between the ball and the own goal, just outside of the penalty box, another player chasing for the ball, the third again staying in the opponents half.

2.4.4 Positioning

The actual positioning during the entire game is done using a potential field. This is a classical approach (see [12], or a more detailed explanation in [13]), and quite common in RoboCup practice, [14]. Basically, different objects on the field form attractors and repulsors, and the superposition of their effects create a resulting shift vector, which determines where the robot is to go. Contrary to other teams, we not only use point sinks and sources, but also lines (and later segments, as described at the end of this chapter).

Potential field

A very intuitive way of looking at the potential field would be to interpret the term “potential” as “height” or “elevation”. Think of the field as a surface containing peaks, ridges, ditches and fissures. The robot could then be seen as a ball left loose on this relief — always rolling down along the steepest slope at its current location.[8] A visual representation of our field (however without any players, or the ball) is shown in Figure 2.3 on the facing page. Note the side lines and the high rising in the centre of the lower goal line. They prevent the robots from leaving the pitch and from entering their own penalty box (in order to respect the illegal defender rule).

In the implementation, we fortunately do not have to calculate the actual potential field itself as shown above. Only the local gradient is of interest and has to be determined. This is done by summing up the influences of the different points and lines involved, as we shall see later on.

[8] To be overly precise, it would have to be a ball without inertia.

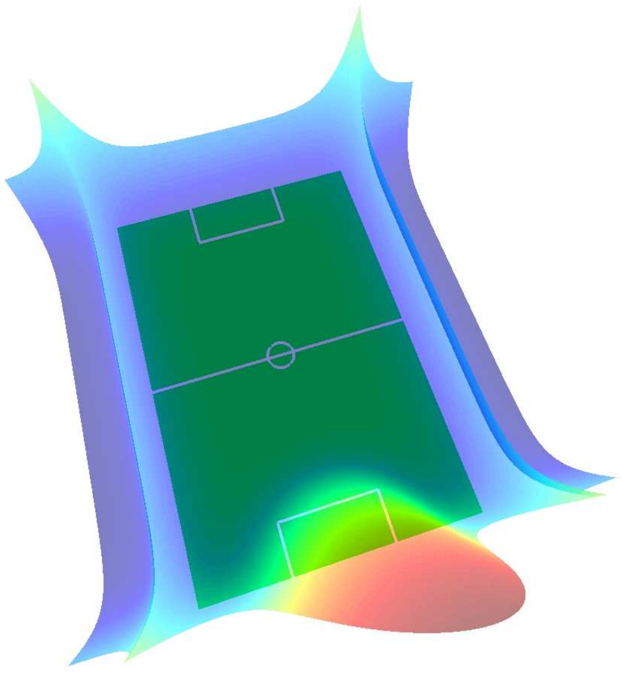


Figure 2.3: *Visual representation of the entire potential field, using a point repulsor in the own goal centre to prevent an "illegal defender".*

Objects

So which are objects influence the robot's positioning? For all players, the side lines are repulsive. Also, points of influence are the ball, the player's home position, its team mates, and the goals. Obviously, all of them will have different parameters (the ball for instance is a very strong attractor, with a very wide spread area of influence, where team mates are small repulsors, to prevent crowding) depending on the role of the player in the current formation, as well as its current mode (whether it is chasing the ball, it is to assume initial formation for kick–off, etc.).

Potentials

To determine the influence of points and lines, we should define them in a more mathematical way. We shall characterise them by their position as well as two parameters: a "height" and a "spread" factor:

Definition 2.1 (Potential created by a point)

We define the potential p_{P} created at some point (x, y) by the potential point $P = (x_{\mathrm{P}}, y_{\mathrm{P}})$ as follows:

$$p_{\mathrm{P}}(x, y) = \alpha_{\mathrm{P}} \cdot \exp \frac{-d_{\mathrm{P}}^2}{\sigma_{\mathrm{P}}^2} \tag{2.9a}$$

with

$$d_\mathrm{P} = \sqrt{(x - x_\mathrm{P})^2 + (y - y_\mathrm{P})^2} \tag{2.9b}$$

where α_P is the maximum potential and σ_P the spread.

In a similar way, let's define the influence of a line on a particular point's potential.

Definition 2.2 (Potential created by a line)
A potential line $L = (A_\mathrm{L}, B_\mathrm{L}, C_\mathrm{L})$, defined by $A_\mathrm{L}x + B_\mathrm{L}y + C_\mathrm{L} = 0$ creates at some point (x, y) the potential

$$p_\mathrm{L}(x, y) = \alpha_\mathrm{L} \cdot \exp \frac{-d_\mathrm{L}^2}{\sigma_\mathrm{L}^2} \tag{2.10a}$$

with

$$d_\mathrm{L} = \frac{A_\mathrm{L}x + B_\mathrm{L}y + C_\mathrm{L}}{\sqrt{A_\mathrm{L}^2 + B_\mathrm{L}^2}} \tag{2.10b}$$

and α_L as the maximum potential, σ_L the spread.

As mentioned above, it is the steepest *gradient* that determines where the robot will move. The gradient being a linear operator, we can, in order to calculate the local gradient, just sum up the individual gradients resulting from the points and lines.

Gradients

For the above equations, determining their contribution towards the gradient is relatively straightforward. We find the different components of the gradient at some point (x_0, y_0) by calculating the partial derivatives along the two axes. So, for (2.9), we find

$$\left.\frac{\partial p_\mathrm{P}(x, y)}{\partial x}\right|_{(x_0, y_0)} = -\frac{2\alpha_\mathrm{P}}{\sigma_\mathrm{P}^2}(x_0 - x_\mathrm{P}) \exp \frac{-d_\mathrm{P}^2}{\sigma_\mathrm{P}^2} \tag{2.11a}$$

and

$$\left.\frac{\partial p_\mathrm{P}(x, y)}{\partial y}\right|_{(x_0, y_0)} = -\frac{2\alpha_\mathrm{P}}{\sigma_\mathrm{P}^2}(y_0 - y_\mathrm{P}) \exp \frac{-d_\mathrm{P}^2}{\sigma_\mathrm{P}^2} \tag{2.11b}$$

Similarly, the components of the gradient created by (2.10) are

$$\left.\frac{\partial p_\mathrm{L}(x, y)}{\partial x}\right|_{(x_0, y_0)} = -\frac{2\alpha_\mathrm{L}}{\sigma_\mathrm{L}^2} \frac{A_\mathrm{L}}{A_\mathrm{L}^2 + B_\mathrm{L}^2}(A_\mathrm{L}x_0 + B_\mathrm{L}y_0 + C_\mathrm{L}) \exp \frac{-d_\mathrm{L}^2}{\sigma_\mathrm{L}^2} \tag{2.12a}$$

and

$$\left.\frac{\partial p_\mathrm{L}(x, y)}{\partial x}\right|_{(x_0, y_0)} = -\frac{2\alpha_\mathrm{L}}{\sigma_\mathrm{L}^2} \frac{B_\mathrm{L}}{A_\mathrm{L}^2 + B_\mathrm{L}^2}(A_\mathrm{L}x_0 + B_\mathrm{L}y_0 + C_\mathrm{L}) \exp \frac{-d_\mathrm{L}^2}{\sigma_\mathrm{L}^2} \tag{2.12b}$$

From here, the overall gradient is calculated by summing up the respective components resulting from the different objects. These components then determine the robots movement (its heading is usually governed by the balls location).

This concludes our remarks on the robots behaviour, and we shall now look at the last software module.

2.5 Locomotion

This modules takes care of all the physical actions of the robot, that is controlling all the motors involved in order to move the robot around the field, carry out the grabbing, kicking or diving for the ball motions, or move its head when searching for it.

It is important to mention that most of the low level control is done by the robot's firmware. So all the movements at the end are commands to the drivers, in the form of a desired angle together with a desired time, in which this angle is to be reached.

2.5.1 Walk

Particular attention has been given to the robot's walk. Again, machine learning has been employed to maximise the speed, which is around 42 cm/s now, depending on the robot. See M. Quinlan's great PhD thesis [16] for more details.

An important feature of the walk is not only its speed, but its omnidirectionality. The Locomotion Engine can be given not only a forward component for setting the speed of the walk, but also a sidewards component, as well as a desired turn rate, giving us all the flexibility needed for efficient game play.

This parametrised walk is made possible by individual leg control (for instance, smaller steps on one side of the robot allow for a turning component in a normal forward walk) together with a very fast yet robust re–synchronisation algorithms, needed in particular when drastically different individual leg movements were made.

2.5.2 Head control

This boils down to two elementary functions — general movements, and object tracking. In the first mode, a certain head motion is performed, in order to search for the ball, pear at a beacon or the like, without any feedback or reaction to what is actually seen.

Tracking however, is a proper control task, a classical visual servoing problem. Usually, once the ball has been spotted, the head will do its best to track it, keep it in the field of vision, independent of what the body "undercarriage" is doing.

In both modes, it is crucial that the pitch of the head, i. e. the vertical elevation of the field of view, is carefully kept within certain limits, so that neither vision distance is unnecessarily reduced or beacons (which are the highest objects on the field) are overseen by having the head to low, nor that too much noise or disturbances are caught (for instance, an orange shirt in a crowd around the field could be misinterpreted as a ball, or the many additional colour blobs spotted could slow down the vision processing) when looking up too heigh.

The NUbot panning motion of the head, when searching for the ball, is slightly slower than in other teams. The slight disadvantage in terms of speed is outweighed by the reliability to actually recognise the ball when it comes within the field of view (other teams often seem to "overlook" the ball with the head panning at high speeds).

These remarks close our introduction to the existing hard– and software set–up of the robot. The last section of the chapter is dedicated to the early modifications we did.

2.6 Modifications

The following changes to the current software, done before working on the actual passing algorithm, greatly improved the success rate for grabbing (which makes passing a lot more effective), fulfilled last minute needs for the 2006 RoboCup competition and allowed the author to get a deeper insight into the current software of the robot.

These modification took place in two areas — the Extended Kalman Filter, and the potential field. We shall start with the former.

2.6.1 Extended Kalman Filter

As mentioned earlier, the measurements used in the Extended Kalman Filter are distance and bearing, coming straight from the Vision module. Using these measurements directly usually proses no problems for most of the objects on the field — but for the ball. Obviously, the distance measurement will frequently tend towards zero (especially when chasing the ball). We have shown in Subsection 2.3.2 on page 11 that the Extended Kalman Filter uses the Jacobians $\boldsymbol{c}_i^T$ of the measurement model functions (2.3). Having the estimated ball distance tend towards zero will result in numerical issues in these Jacobians, which becomes apparent, when we calculate them.

Jacobians

Let $\hat{x}^-$ and $\hat{y}^-$ denote the current *a priori* estimates of the position of the robot, $\hat{x}_\mathrm{b}^-$ and $\hat{y}_\mathrm{b}^-$ those of the ball. To improve readability, we shall omit the superposed minus as well as the index k in the following. We find for the Jacobians of the measurement model functions

$$\frac{\partial h_1(\hat{\boldsymbol{x}})}{\partial \hat{\boldsymbol{x}}} = \boldsymbol{c}_1^T = \begin{pmatrix} (\hat{x} - \hat{x}_\mathrm{b})/\hat{d} \\ (\hat{y} - \hat{y}_\mathrm{b})/\hat{d} \\ 0 \\ (\hat{x}_\mathrm{b} - \hat{x})/\hat{d} \\ (\hat{y}_\mathrm{b} - \hat{y})/\hat{d} \\ 0 \\ 0 \end{pmatrix}^T$$

and

$$\frac{\partial h_2(\hat{\boldsymbol{x}})}{\partial \hat{\boldsymbol{x}}} = \boldsymbol{c}_2^T = \begin{pmatrix} (\hat{y}_\mathrm{b} - \hat{y})/\hat{d}^2 \\ (\hat{x} - \hat{x}_\mathrm{b})/\hat{d}^2 \\ -1 \\ (\hat{y} - \hat{y}_\mathrm{b})/\hat{d}^2 \\ (\hat{x}_\mathrm{b} - \hat{x})/\hat{d}^2 \\ 0 \\ 0 \end{pmatrix}^T$$

with $\hat{d} = \sqrt{(\hat{x} - \hat{x}_\mathrm{b})^2 + (\hat{y} - \hat{y}_\mathrm{b})^2}$.

Problems

Clearly, when the filter's estimate of the ball's position tends towards the robots own position (as it should, when the ball is approaching the player's chest), numerical problems can occur (potential division by zero). These are believed to be one of the reasons for the occasional "explosion" of the robot's position (that is, a sudden delocalisation of the robot, i. e. the robot position jumps significantly inside a couple of frames) as well a limiting factor for the number of successful grabs. One method for preventing these numerical issues is described in the following.

Relative ball position

Switching to a different coordinate system when the ball comes closer (we have chosen an arbitrary 50 cm as threshold) will result in a different Jacobian, potentially preventing our numerical problem.

We suggest the use of a so–called "relative ball" coordinate system, which is not a local polar coordinate system (corresponding to the ball's distance and bearing), but a local *Cartesian* coordinate system. We consider the ball's x– and y–coordinates $(x_\mathrm{b}^\mathrm{r}, y_\mathrm{b}^\mathrm{r})$ relative to the current robot's current position and heading.

Due to the classical set–up (relative coordinates are already used internally elsewhere in the software), they are oriented as follows: if the robot looks straight ahead, it looks along the positive direction of its y–axis, the x–axis extending towards its right, as shown in Figure 2.4 on the following page.

Modification of the Extended Kalman Filter

The measurements coming from Vision can be transformed easily into this new coordinate system. For a measurement couple distance and bearing of the ball $(d_\mathrm{b}, \vartheta_\mathrm{b})$, we have simply

$$x_\mathrm{b}^\mathrm{r} = -d_\mathrm{b} \sin \vartheta_\mathrm{b} \tag{2.13a}$$

$$y_\mathrm{b}^\mathrm{r} = d_\mathrm{b} \cos \vartheta_\mathrm{b} \tag{2.13b}$$

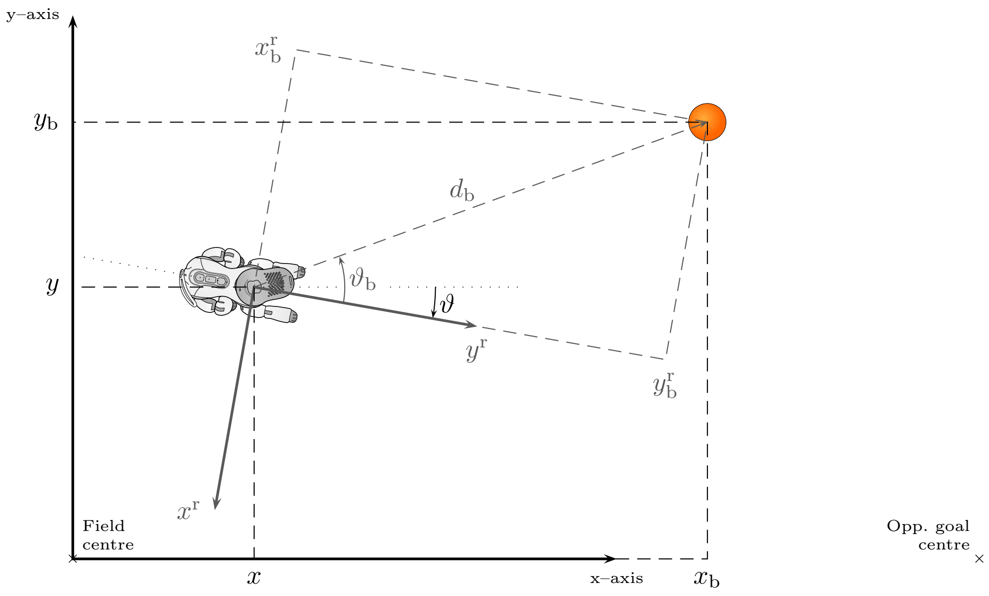


Figure 2.4: *Overview of most of the important coordinates introduced so far.*

These new measurement variables result in new measurement model function $h^{\mathrm{r}}_i(\hat{\boldsymbol{x}})$, replacing (2.3) on page 12:

$$h^{\mathrm{r}}_1(\hat{\boldsymbol{x}}) = \left(\hat{x} - \hat{x}_{\mathrm{b}}\right) \sin\left(\hat{\vartheta}\right) - \left(\hat{y} - \hat{y}_{\mathrm{b}}\right) \cos\left(\hat{\vartheta}\right) \tag{2.14a}$$

$$h^{\mathrm{r}}_2(\hat{\boldsymbol{x}}) = \left(\hat{x} - \hat{x}_{\mathrm{b}}\right) \cos\left(\hat{\vartheta}\right) + \left(\hat{y} - \hat{y}_{\mathrm{b}}\right) \sin\left(\hat{\vartheta}\right) \tag{2.14b}$$

They also change the Jacobians which is exactly what we wanted to achieve. With $i = 1$ now standing for the "measurement" of the ball's relative x–position, $i = 2$ that of it's relative y–position, we have

$$\frac{\partial h^{\mathrm{r}}_1(\hat{\boldsymbol{x}})}{\partial \hat{\boldsymbol{x}}} = {\boldsymbol{c}^{\mathrm{r}}_1}^T = \begin{pmatrix} -\sin(\hat{\vartheta}) \\ \cos(\hat{\vartheta}) \\ \left(\hat{x} - \hat{x}_{\mathrm{b}}\right) \cos\left(\hat{\vartheta}\right) + \left(\hat{y} - \hat{y}_{\mathrm{b}}\right) \sin\left(\hat{\vartheta}\right) \\ \sin(\hat{\vartheta}) \\ -\cos(\hat{\vartheta}) \\ 0 \\ 0 \end{pmatrix}^T$$

and

$$\frac{\partial h^{\mathrm{r}}_2(\hat{\boldsymbol{x}})}{\partial \hat{\boldsymbol{x}}} = {\boldsymbol{c}^{\mathrm{r}}_2}^T = \begin{pmatrix} -\cos(\hat{\vartheta}) \\ -\sin(\hat{\vartheta}) \\ \left(\hat{x} - \hat{x}_{\mathrm{b}}\right) \cos\left(\hat{\vartheta}\right) + \left(\hat{y} - \hat{y}_{\mathrm{b}}\right) \sin\left(\hat{\vartheta}\right) \\ \cos(\hat{\vartheta}) \\ \sin(\hat{\vartheta}) \\ 0 \\ 0 \end{pmatrix}^T$$

The clear advantage over (2.3) on page 12 is that, here, no term is divided by the ball distance, thus preventing the abovementioned numerical problems.

Covariances

Finally, before being able to implement the modified filter, we need to determine the new measurement variances r^{r}_i. As argued in Appendix A, we are safe to use the measurement covariance from the original distance measurement (that is coming from Vision) for *both* x^{r} and y^{r}, i. e. $r^{\mathrm{r}}_i(k) = r_1(k)$ for $i = 1, 2$.

Results and demonstration

It is fairly hard to demonstrate the improvement of the robot's ball handling capabilities, especially the grabbing,[9] but some first statistics indicate an increase of at least 20% in the success rate for grabs.

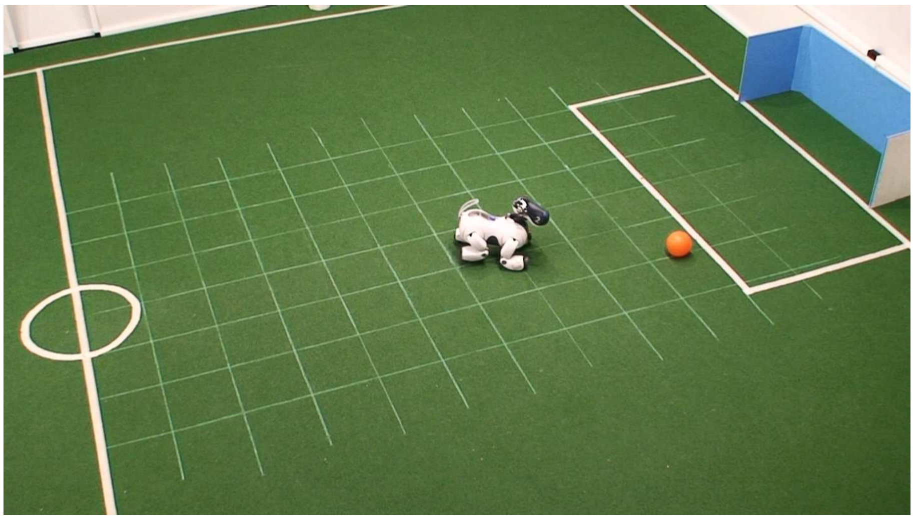


Figure 2.5: *Testing the effect of our modification of the robot's Extended Kalman Filter on its localisation.*

However, the beneficial effect on the robot's localisation could be demonstrated by running the two different Extended Kalman Filter versions side by side. See Figure 2.6 on the next page for plots of the player's estimated position with and without using the "relative ball" coordinate system, compared to his real position (the truth data has been gathered by analysing a video of the experiment, as shown in Figure 2.5).

In the experiment, the ball has just been removed from the robot's chest near field coordinates (60,0). It was then placed back on the field closer to the goal, near (190,-60). At first, the player turns on the spot to search for the ball. Once he has seen it again, he starts chasing it.

[9] Mainly for two reasons. First, too many parameters involved in grabbing depend on the position estimates and thus had to be completely retuned with the new, improved estimates. Second, it is almost impossible to reproduce the exact situation (and Kalman filter history) twice to run it with and without the modification.

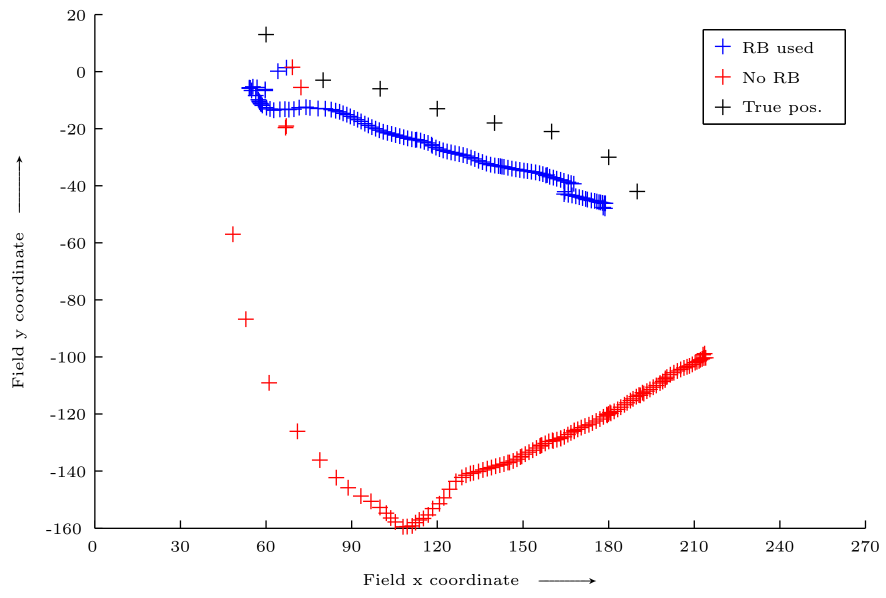


Figure 2.6: *Comparison of the robot's localisation with and without using the "relative ball" coordinate system with some truth data, which has been obtained using video footage of the experiment.*

During the time when he could not see it, the ball stayed — from the Extended Kalman Filter's point of view — near (60,0), with continuously increasing variance. Once Vision detected the ball again, this time near (190,-60), the filter had to "catch up" as quickly as possible on that new location. This is where the two implementations start to differ.

Looking at Figure 2.6, which displays the estimated robot position frame by frame, we can see the "explosion" of the model not using the "relative ball". The very large corrections quickly shift the perceived robot position away from its true value. This can be explained by looking at (2.6) and (2.7) on page 12. Here, a large *a priori* covariance matrix is multiplied by the large Jacobians of the measurement model functions and thus create the strong corrections.

The model using the "relative ball" coordinates, however, shows smoother updates of the robot's position which stays significantly closer to its real values. One may also note the near constant offset between estimated and real positions here. This can be explained by the fact that on the way to the ball, the robot could only see two objects — the ball and the right goal post. This was not enough (and precise) information for the Extended Kalman Filter to be able to correct the initially existing offset.[10]

[10] Nevertheless, this is not a problem when it comes to chasing and grabbing the ball, as in such a situation it is almost entirely the ball measurements that update the robot's (and ball's) estimated position — resulting in the *relative* position of both being estimated correctly.

These remarks finish our section on the modification of the Extended Kalman Filter. Let us close this chapter by discussing our second preliminary modification to the original game code.

2.6.2 Potential field

The modification done here is the addition of a feature. At the moment, the potential field as introduced in Subsection 2.4.4 on page 16 only allows for points and (infinite) lines.

Problem

In order to prevent an "illegal defender", until now a strong repulsor for all players other than the goal keeper was placed in the middle of the own penalty box, as we have seen earlier in Figure 2.3 on page 17.

Although simple and computationally cheaper, there is a major drawback in using this technique to prevent a robot from entering its own penalty box. It is obviously hard to evenly cover a rectangular area with a circular object. Either you "over–" or "undercover" the area.

Solution

To solve this problem and give even more flexibility to the potential field, we introduced another "object". In addition to points and lines, we can now also add segments to the potential field. These are then used to cover the penalty box much better, and could also be used for other purposes.

Implementation

Mathematically, our segments are a combination of two points and a line. They are defined by their two end points, and a height and spread factor. The calculation of the resulting gradient involves two steps: first, using some vector maths and a few simple logic test, it is determined whether the current point lays between, or outside of the two points that characterise the segment. Then, either calculate the local gradient using the formula for a point or a line potential.

Let us assume a situation as shown in Figure 2.7 on the next page, and we want to calculate the segment's effect on the local gradient in P, Q and R.

1. Location Using two dot products and two logical checks, we can determine whether the point is under the influence of the segment (that is when it lays in region $\mathcal{B}$), or the points S_1 or S_2 (regions $\mathcal{A}$ and $\mathcal{C}$ respectively).

Let us call the property of the point being in either $\mathcal{A}$, $\mathcal{B}$ or $\mathcal{C}$ respectively $\mathfrak{A}$, $\mathfrak{B}$ and $\mathfrak{C}$. We then have

$$\begin{aligned} \mathfrak{A}\mathfrak{B} \quad &\Leftrightarrow \quad \boldsymbol{s}\cdot\boldsymbol{p}_2 < 0 \\ \mathfrak{B}\mathfrak{C} \quad &\Leftrightarrow \quad \boldsymbol{s}\cdot\boldsymbol{p}_1 < 0 \end{aligned}$$

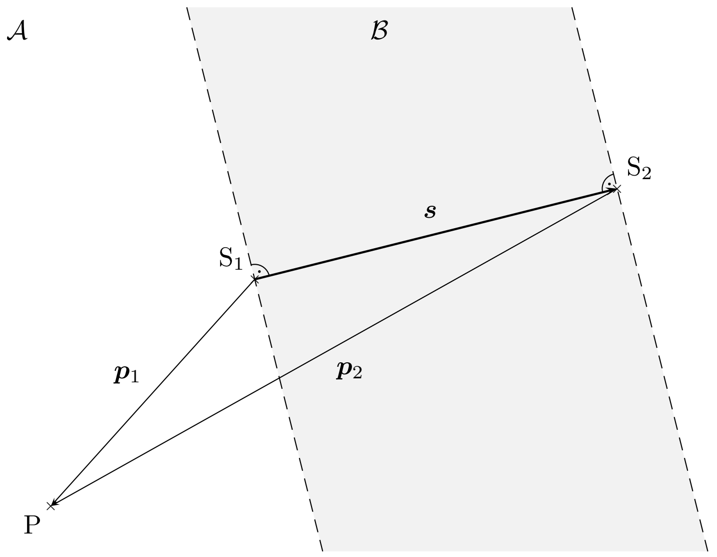


Figure 2.7: *Illustration for determining where a point* P_1 *is relative to a segment given by the points* S_1 *and* S_2*.*

In the next step, we perform the following two logical tests:

$$\begin{array}{ccccc} \mathfrak{A} & \Leftrightarrow & \neg(\mathfrak{B}\mathfrak{C}) \wedge (\mathfrak{A}\mathfrak{B}) & \vee & \boldsymbol{s}\cdot\boldsymbol{p}_1 = 0 \\ \mathfrak{C} & \Leftrightarrow & \neg(\mathfrak{A}\mathfrak{B}) \wedge (\mathfrak{B}\mathfrak{C}) & \vee & \boldsymbol{s}\cdot\boldsymbol{p}_2 = 0 \end{array}$$

If neither of them is true, then $\mathfrak{B}$ is the case, i. e. P lays between the end points of the segment, thus is under the influence of the line.

2. Gradient Now that we know the location of the point, we can go ahead and calculate the gradient using either (2.11) or (2.12) on page 18. For the latter however, we need to determine A_L, B_L and C_L, which is straightforward. If the components of $\boldsymbol{s}$ are x_s and y_s, we simply have

- if $x_\mathrm{s} = 0$, then $A_\mathrm{L} = 1$ and $B_\mathrm{L} = 0$ (vertical segment),
- if $y_\mathrm{s} = 0$, then $A_\mathrm{L} = 0$ and $B_\mathrm{L} = 1$ (horizontal segment),
- else, $A_\mathrm{L} = -y_\mathbf{s}/x_\mathbf{s}$ and $B_\mathrm{L} = 1$

The third parameter of the line is finally calculated using one of the points, say $S_1 = (x_{S_1}, y_{S_1})$. This would lead to $C_\mathrm{L} = -A_\mathrm{L}x_{S_1} - B_\mathrm{L}y_{S_1}$.

Note that the normalisation in (2.10b) on page 18 is important to ensure perfect blending between the point– and line influence at the borders between the different "zones". The result from the technique described is demonstrated in Figure 2.8 on the facing page.

It is important to mention that for the penalty box two small tweaks are used. First, the segments only influence points *outside* of the box. Inside of it, a point repulsor (that, in turn, only influences the *inside*) has been placed, to

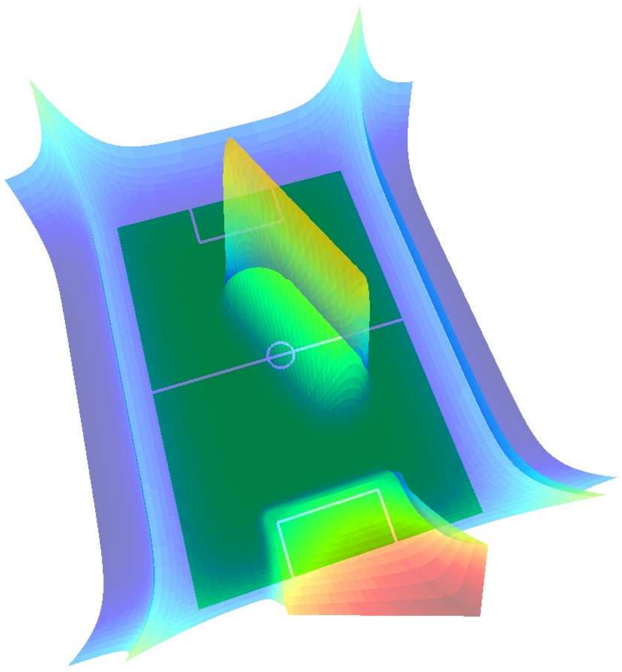


Figure 2.8: *Visual representation of the entire potential field, now using segments and a point repulsor to prevent an "illegal defender". We have also added another arbitrary segment for demonstration purposes.*

push the robot out in case it finds itself inside the box. If the segments here were double–sided, it would not be able to leave the box.[11]

Second, the end points of the top segment of the box are purposely not having any influence. This is to prevent overly "high" corner points which would result from the superposition of these end points with those of the side segments of the box. Such overly high points would create a more wide spread corners that would, in turn, hinder a robot from "sliding" around the corners of the box when chasing the ball for instance.[12]

This concludes our preliminary modifications, and, in the broader sense, our introduction to the NUbot players.

[11] Of course, if the robot really is inside the box, it is removed as set by the rules. However, our modification is meant to cover the case where the player is badly localised (i. e. he thinks he is inside the box, but in fact is not), to prevent it from stepping any closer and potentially *really* entering it.

[12] The result of such an unnecessary superposition of potentials can be seen in the four corners of the pitch, where the side lines add their respective influence (which, however, is deemed acceptable).

CHAPTER 3

Algorithm design

We will describe our approach to the decision making for passing, in what direction to actually kick the ball, and how sender and receiver cooperate in that case.

3.1 Introduction

Having presented the hardware and software of the NUbots, along with some modifications to the existing Extended Kalman Filter to improve ball position and velocity estimation, we will now focus on the main topic of this thesis. This chapter lays out our approach to the decision making and execution of deliberate passes.

Before presenting our algorithm, let us quickly consider an introductory example to highlight some of the more general problems involved. In the situation shown in Figure 3.1 on the next page, we can tell immediately: *"Yes, this would be a good pass."* This statement is easy to make for us, as on the one hand we have a good and very precise overview of the entire game situation from our "bird's eye view", and on the other hand we have of our experience, intelligence and intuition.

Our robotic soccer players however — with their camera about 15 cm above the ground and only about 50° of horizontal field of view, not to mention the low resolution and poor quality pictures delivered by their camera — do not have this overview, thus have to rely on and trust their own localisation and that of their team mates. Also, they do not possess any real "experience" in soccer playing and cannot predict the behaviour of the other players as a human player can. So, how can we establish rules for judging the quality of a pass? How can we you determine the ideal bearing for kicking the ball?

In the preparation of this thesis, two approaches have been considered. They were to either use a one– or two–dimensional potential field to respectively determine ideal bearings for passing, or ideal locations for the ball to go. Concerning the 2D approach, we had the following problem. Contrary to the potential field we encountered in the last chapter where we only needed to calculate the *local* gradient, the situation here would require finding a *global* maximum. As such a task would require much more involved and extensive computations, it was soon decided that we should rather take the one–dimensional approach.

Once the decision for passing has been made, it needs to be executed. Surely, this should involve some form of communication and coordination be-

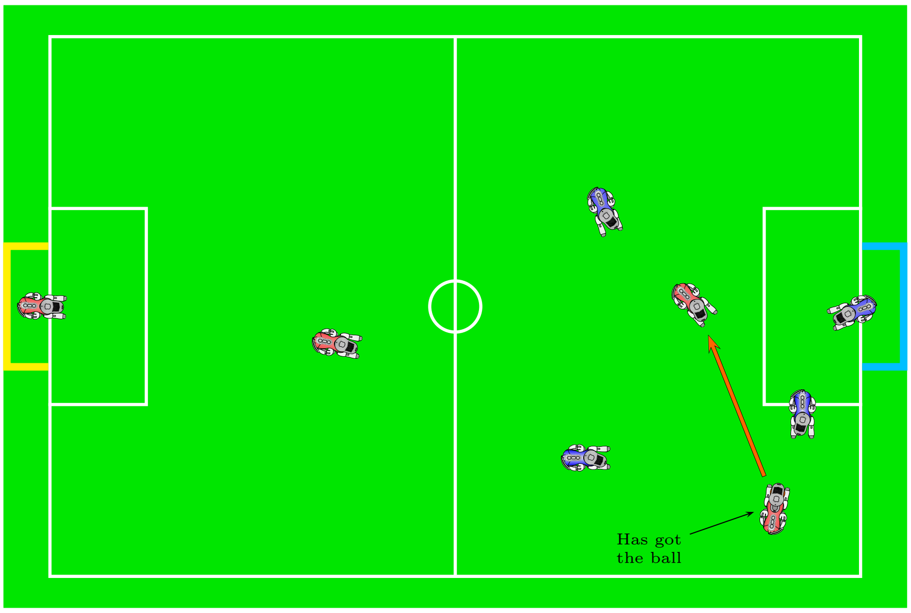


Figure 3.1: *Red Team is attacking blue goal. Passing the ball as indicated by the arrow would certainly be advantageous in this situation.*

tween sender and receiver.[1] The sender needs to line up the shot, and the receiver should get himself in a position to receive the pass. Ideally, the knowledge of the exact moment of the kick should also be incorporated in the receivers Extended Kalman Filter to improve its ball position and velocity estimation.

So let us start off with describing the decision making process, first discussing the main ideas, then its mathematical transcription and finally the resulting overall algorithm (its implementation in python being shown in the Appendix). The last section then focusses on the communication and further modification to the Extended Kalman Filter.

3.2 Main concept

In order to come up with an effective and robust algorithm for the decision making, our questions from the introduction need answering. We should come up with a set of rules which will favour passing in one particular direction, but also prevent useless or too dangerous passes. For that, we are considering the following one dimensional potential field.

3.2.1 Horizon scanning

Our general approach is based on considering the full horizon around the player. The aibo's are physically not strong enough to kick the ball off the ground in a

[1] From now on well shall call sender the player currently in possession of the ball (and about to potentially play a pass), receiver his team mate that is intended to receive the (potential) pass.

controlled manner (in order to intentionally kick over other players). So, when it comes to passing, our NUbots have only one option — that is to kick the ball in one particular direction besides maybe first dodging around a defender.

With the quality of the localisation as well as the performance and precision of the kick[2] we use, it is sufficient to consider a resolution of one degree, resulting in 360 possible kicking directions.

3.2.2 What makes a good pass

Passing — from the sender point of view — ultimately means kicking the ball into one particular direction. This action however only results in a "pass" when the receiver picks the ball up and carries on with it from that point. It is a "good" pass, when a strategical advantage has been gained, such as gaining space, clearing defenders or getting a better angle for shooting at the goal.

As the robot about to make the pass cannot predict the behaviour of its team mate (as a human soccer player of course does), he will have to base his decision uniquely on a "snapshot" of the positions of himself, his team mates and opponents.

Three fundamental criteria for a good pass are obvious:

– the pass should go precisely to or in front of a team player,

– the receiver must not be hidden behind an opponent player and

– the pass should never go backwards, or cross the field in front of your own goal.[3]

More elaborate criteria would be, that the receiver should

– not be too far away from the sender, but also not too close,

– be close to the goal,

– be clear from opponent players.

So how can these "linguistic rules" be translated into an algorithm that can be implemented in a normal programming language to run as fast and as efficient as possible?

3.2.3 Fuzzy logic

The term linguistic rules usually invites fuzzy logic to the scene. The concepts of this emerging area in modern information processing technologies were first formally introduced by Latfi A. Zadeh in his seminal paper [19], and later extended in [20]. Since then, fuzzy logic has become one of the mainstays in modern knowledge based systems, with many real–world applications ranging from home appliances to industrial installations. The interested reader may be

[2] Note that we only use one "standard" kick, which kicks the ball straight ahead. The particular kick we are using has been chosen from the large repertoire of kicks because is the most reliable kick and has an adequate speed of usually around $40 - 60$ cm/s.

[3] At least in our RoboCup philosophy. The low levels skills are still to unreliable and the probability of an opponent intercepting such a "pass" and obtaining a good goal scoring opportunity is too high.

referred to the recent text [15] for some broad and in–depth background on the topic.

Fuzzy sets, which are the entities at the heart of fuzzy logic, try to capture or represent objects or properties with uncertain, "fluid" boundaries — such as linguistic notions. A classical example would be the notion of "size", for instance. When can a person be considered "tall"? If it is taller than two metres, or 1.90 m or 1.80 m? It is obviously hard to make a (meaningful) binary decision between being tall and not being tall.

The approach we shall take below — whilst not directly following fuzzy logic paradigms — does embrace related concepts in trying to mathematically describe some of the heuristics involved in passing.

3.3 Mathematical transcription

Having discussed important characteristics of a good pass, we shall now see how these criteria are actually written down or "translated" into mathematical expressions so that we can deal with them on an algorithmic level. Let us start off with a few remarks concerning the potential field itself, to then discuss the different objects that influence it.

3.3.1 Potential field

The field encountered here is quite different from the one we saw in the last chapter. On the one hand it is only a one–dimensional potential field — corresponding to the full 360 degrees of horizon line around the sender. On the other, the potential in one point tries to relate to the quality of a possible pass in that direction, thus some form of desirability (in the other case we were rather focussing on the gradient, not the potentials themselves).

Potentials

Let us define a positive potential[4] at some bearing to be a "good" direction to play the ball in (i. e. a direction, that would quite likely result in a "good" pass). A negative potential, conversely, would correspond to a disadvantage to the team if the ball was kicked that way.

The overall potential field is the cumulative effect of the different elements that influence the field, mathematically speaking the superposition of the effects of each attractor. So what objects influence the potential or attraction of one particular direction?

Sources

In our set–up, we distinguish between three classes of objects that can produce some form of attraction. These classes are

– team mates,

– opponents and

[4] From now on we shall also speak of "attraction" or "desirability".

– “other”.

The objects in these classes differ mainly in two ways: the general sign of the attraction (positive or negative) and their clipping behaviour as describe below.

Team mates Obviously, the major sources of attraction should be team mates. If they are hidden behind an opponent player, their attraction should be clipped (hence ignored). However, two team mates that are in line with the sender should be allowed to superpose or double up their amounts of attraction.

Opponents A disastrous action would be to play the ball in the hands (paws) of an opponent player, so these are our big sources of repulsion. Two opponents behind each other should add up their negative influence, but an opponent hidden behind a team mate should not have any negative impact in the concerned bearings.

Other This class of objects only contains one object at the moment, but this can, of course, be extended in feature work if necessary. We call this object the goal attractor, with field coordinates $\mathrm{GA} = (210, 0)$. This point exerts a positive attraction, that cannot be clipped by any other object. It serves the purpose of attracting or favouring passes closer to the goal by giving a small additional attraction to bearings close to the goal. That point is not located directly on the goal line but rather 60 cm in front of it in order to produce good cross passes from the sides.

Clipping

As clipping effects are very important in our application here, let us repeat the clipping rules using Table 3.1 below. The objects shown there are supposed to be in line with the sender, the second object farther away from the sender than the first object, thus being hidden behind the first object.

		1st object		**2nd object**		**Clipping?**
Sender	$\longrightarrow$	Team mate	$\longrightarrow$	Team mate	$\longrightarrow$	$\times$
Sender	$\longrightarrow$	Team mate	$\longrightarrow$	Opponent	$\longrightarrow$	$\surd$
Sender	$\longrightarrow$	Team mate	$\longrightarrow$	Goal attr.	$\longrightarrow$	$\times$
Sender	$\longrightarrow$	Opponent	$\longrightarrow$	Team mate	$\longrightarrow$	$\surd$
Sender	$\longrightarrow$	Opponent	$\longrightarrow$	Opponent	$\longrightarrow$	$\times$
Sender	$\longrightarrow$	Opponent	$\longrightarrow$	Goal attr.	$\longrightarrow$	$\times$
Sender	$\longrightarrow$	Goal attr.	$\longrightarrow$	Team mate	$\longrightarrow$	$\times$
Sender	$\longrightarrow$	Goal attr.	$\longrightarrow$	Opponent	$\longrightarrow$	$\times$

Table 3.1: *Clipping rules: The goal attractor and objects of same type are never clipped.*

If an object is being clipped, it’s core potential and sloped parts (as described in Definition 3.2 on page 36) will have no influence on the overall potential in the range of bearings of the core of the first object.

Region of interest

Even though we calculate the potential along the entire horizon around the player, it is clear that passes should only be considered within in a limited portion or region of the horizon, usually near the direction of the goal. We call this the region of interest. It can also be seen as a fail safe, that, for instance, no pass would ever go backwards (possibly in the direction of the own goal).

For that, we came up with a simple set of rules that determine our region of interest. Generally, we distinguish between two situations.

(i) The first one is a defensive situation, that is, when the sender is in the first quarter of the field ($x < -135$). Although passes can be useful here for clearing the ball, we do not want any passes crossing the field in front of the goal (again, as a safety precaution). If the ball is close to the x–axis of the field, passes should only go outwards (i. e. towards the sidelines). Only if the player is further out (towards the sidelines) he should be allowed to play passes going up the field.

   We decided to have the region cover a constant ±36° to both sides of its centre, which moves from ±90° (i. e. pointing outwards) to 0° (pointing up the field) as y goes from 0 to ±180.

(ii) If we are in the other three quarters of the field ($x \geq -135$), passes should generally go in the rough direction of the opponent goal. For that reason, we centre the region of interest on the bearing of the goal attractor GA. Also, as we approach the goal, we should "open up" the region of interest, i. e. consider a larger area of bearings for passing — we do this by changing the extent of the region of interest depending on the players x–position on the field.

   More precisely, if the robot is in his own half (i. e. $x < 0$), the region of interest extends ±36° around the goal attractor bearing. If the robot is in the offensive half, the range increases continuously from ±36° to ±72° as x goes from 0 to 270.

Let us put this down again more formally:

Definition 3.1 (region of interest)

The region of interest $\mathcal{R}$ is the set of angles in the direction of which (and only which) passes are allowed to be played. When $x \geq 135$, it is symmetric around the bearing of the goal attractor $\mathrm{GA} = (210, 0)$, otherwise changes depending only on y. These two cases are precisely:

(i) $x < 135$:

$$\mathcal{R}(x,y) = \begin{cases} \left\{\vartheta \in [0,360[\;\middle|\; 54 - \frac{y}{2} \leq \vartheta \leq 126 - \frac{y}{2}\right\}, & \textit{if } y \geq 0 \\ \left\{\vartheta \in [0,360[\;\middle|\; -126 - \frac{y}{2} \leq \vartheta \leq -54 - \frac{y}{2}\right\}, & \textit{otherwise} \end{cases}$$

(ii) $x \geq 135$:

$$\mathcal{R}(x,y) = \left\{\vartheta \in [0,360[\;\middle|\; \vartheta_{\mathrm{GA}} - \Delta\vartheta_{\mathcal{R}} \leq \vartheta \leq \vartheta_{\mathrm{GA}} + \Delta\vartheta_{\mathcal{R}}\right\}$$

where

$$\vartheta_{\mathrm{GA}} = \begin{cases} \tan^{-1} \dfrac{-y}{210 - x}, & \textit{if } x \neq 210 \\ -\,90 \cdot \operatorname{sign} y, & \textit{otherwise} \end{cases}$$

and

$$\Delta\vartheta_{\mathcal{R}} = \begin{cases} 36, & \textit{if } x < 0 \\ 36 \cdot (1 + x/270), & \textit{otherwise} \end{cases}$$

All angles and operations involved are to be taken modulo 360.

The different cases and the region of interest's behaviour is demonstrated and commented on in the next chapter, that is in Subsection 4.2.1 on page 45. As mentioned above, it plays an important role in the final passing decision, as we shall see next.

Pass trigger and direction

There are two conditions that have to be fulfilled for a pass to be triggered.

(i) The sender must not be inside the opponent's penalty box,

(ii) the potential has to exceed a certain threshold value *inside* the region of interest.

If both conditions are satisfied, a pass will be played in the direction of the maximum potential (inside the region of interest). The difference between that maximum and the current threshold value is taken to be a measure of quality of the pass. The larger the value, the better. With a modification to the current goal shooting algorithm[5] we could use this quality measure to balance between playing the pass or rather taking a direct shot at the goal and potentially scoring right away without the danger of giving away the ball in the course of passing.

Clearly, the threshold value is an important parameter in our algorithm, as it has a very strong and direct influence on the amount of passes that will be attempted in the game. Its value, called τ, depends on the sender position on the field. If the sender is further away than 120 cm for the goal attractor point GA, it has a constant value of τ_0. However, as d_{GA} is smaller than 120 and decreases further, the threshold is increased continuously until it is 50% higher than its nominal value. So we have

$$\tau(d_{\mathrm{GA}}) = \begin{cases} \tau_0, & \text{if } d_{\mathrm{GA}} > 120 \\ \tau_0 \frac{120 - d_{\mathrm{GA}}}{120 \cdot 2} + \tau_0 & \text{otherwise} \end{cases}$$

This said, let us now take a closer look at how exactly the objects influence our potential field.

[5] That is, calculate some measure of confidence for scoring a goal, using for instance the current distance from the goal or the width of the largest gap between keeper and goal post.

3.3.2 Attractors

An attractor[6] generally is an object that will contribute some positive (or negative) potential to the field, thus favouring (or disfavouring) a particular region of the horizon.

For reasons of flexibility, but also computational efficiency, we chose the following, simple shape for our attractors. In our example we use a positive height (resulting in positive attractor), but repulsors have the exact same properties.

Characteristic values

Let us now formally define an attractor together with all its characteristic values.

Definition 3.2 (Attractor)

As shown in Figure 3.2 below, the potential $p_{\mathrm{A}}(\vartheta)$, for $\vartheta \in [0, 360[$, created by an attractor $\mathrm{A}(c, h, r, s, \gamma)$ is of trapezoidal shape.

The potential it creates is symmetric (with the angle being modulo 360) about its centre bearing c. The core of the object's potential is a flat top at height h with a certain radius r, the width of the flat part to both sides of the centre.

The sides then slope down linearly with a finite slope s in the so–called sloped part until they reach the cut–off value γ, where the potential "bottoms out" flat again, called the cut–off part.

There are two kinds of attractors. Depending on the sign of h they are called positive attractors (or just attractors) or negative attractors (or repulsor).

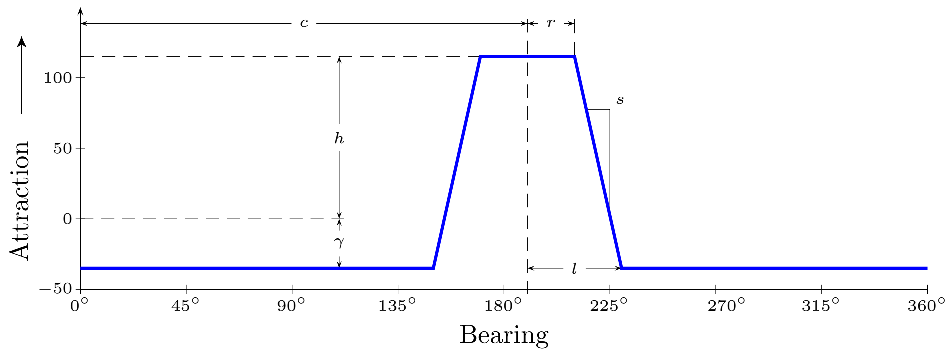


Figure 3.2: *Illustration for a positive attractor, showing the characteristic values: centre c, height h, radius r, slope s and cut–off γ. The auxiliary variable l is defined below (3.2).*

Again, we can get a negative attractor (i. e. a repulsor) by changing the sign of h, s and γ. For that reason, the term "height" should be understood more as a "maximum (minimum) potential", or "maximum attraction (repulsion)".

[6] Note that there is always a certain ambiguity between "attractor" in the general sense, where it could have a positive *or* negative sign, and the "attractor" in the narrow sense, being a strictly attractive object. It should, however, be clear from the context which one is meant.

Choice of parameters

As we have seen, an attractor is completely described by its five parameters. A good choice of these parameters is to the performance of the overall algorithm and they should depend on the relative position (distance and bearing) of the object creating it.

The power of our potential field approach relies not only on the strong attractor or repulsor cores. The small contributions created by the cut–off parts outside the main bearing of an object (such as the goal attractor, or the choice of cut–off values) also play a key role in ultimately favouring the "best" direction for passing possible in a particular situation.

The actual numerical values for the parameters of the following functions can be found in Appendix B.

Centre The centre of an attractor should obviously lay in the direction of the object it is related to. However, concerning team mates for example, one could also try to shift it a bit relative to the player's real bearing based on his own heading as the robots have certainly more influence on the area in front of them than behind their back.

Height We have got three different formulas for the potential in the core of the object — one formula per object class, that is team mates, opponents and the goal attractor respectively:

$$h_{\mathrm{T}}(d) = \begin{cases} 0, & \text{if } d \leq d_{\mathrm{A}} \text{ or } d \geq d_{\mathrm{B}} \\ h_{\mathrm{T},0} \cdot \frac{d - d_{\mathrm{A}}}{d_{\mathrm{M}} - d_{\mathrm{A}}}, & \text{if } d > d_{\mathrm{A}} \text{ and } d < d_{\mathrm{M}} \\ h_{\mathrm{T},0}, & \text{if } d \geq d_{\mathrm{M}} \text{ and } d \leq d_{\mathrm{N}} \\ h_{\mathrm{T},0} \cdot \frac{d_{\mathrm{B}} - d}{d_{\mathrm{B}} - d_{\mathrm{N}}}, & \text{if } d > d_{\mathrm{N}} \text{ and } d < d_{\mathrm{B}} \end{cases}$$

$$h_{\mathrm{O}}(d) = h_{\mathrm{O},0} \cdot \frac{d}{650} + h_{\mathrm{O},0}$$

$$h_{\mathrm{GA}}(d) = h_{\mathrm{GA},0}$$

These functions are shown in Figure 3.3 on the following page. The number 650 corresponds to the length of the diagonal of the entire field (≈ 649.0 cm), the longest distance that should be encountered in the game.

Concerning the choice of the shape of the functions, the idea was that passes should not be attempted when the receiver is too close, or too far away. With two "transitional" regions between d_{A} and d_{M} as well as d_{N} and d_{B}, the main distance for a receiver to attract a pass lays between d_{M} and d_{N}.

Opponents however have a slightly decaying repulsion as distance grows — the closer they are, the more dangerous they are. The goal attractor has a constant core height, as it should attract passes independent of how far away it is.

Radius The core radius for opponents and team mates is specified in centimetres locally to both side of the robot (for instance $r_{\mathrm{T},0} = 20$ cm for team mates). Depending on the distance to the object, this radius is then "translated" into

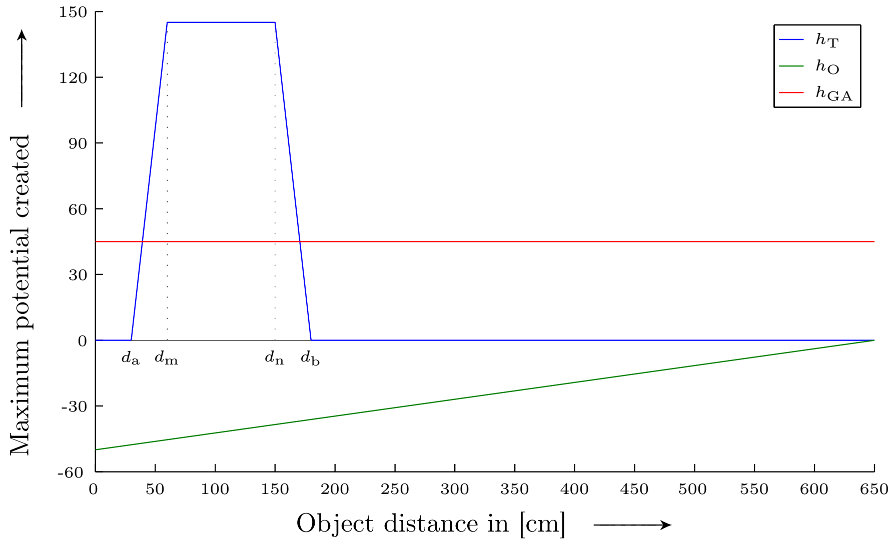


Figure 3.3: *Maximum potential created by an object as a function of its distance. Objects are team mates, opponents and the goal attractor.*

a corresponding angle by using

$$
\begin{aligned}
r_T(d) &= \tan^{-1}\left(\frac{r_{T,0}}{d}\right)\\
r_O(d) &= \tan^{-1}\left(\frac{r_{O,0}}{d}\right)\\
r_{GA}(d) &= r_{GA,0}
\end{aligned}
$$

A non–zero radius allows on the one hand to "cover" some of the uncertainty in localisation of the other objects (robots),[7] and on the other hand to simulate the "domain of influence" of the robot, i. e. how far it can reach if it suddenly dives, for instance.

The goal attractor however has presently got a zero radius to make it a very precise attracting bearing.

Slope The slopes also play an important role on how local or broad an influence the object has. These are our simple equations for the sloping parts of the trapezoidal potential funciton:

$$
\begin{aligned}
s_T(d) &= s_{T,0}\cdot\frac{1}{d}\\
s_O(d) &= s_{O,0}\cdot d\\
s_{GA}(d) &= s_{GA,0}
\end{aligned}
$$

For team mates, the slopes should become steeper the closer the robot is in order to limit its influence. This is particularly important in the following

[7] In the future, one could make the radius change according to the confidence of the respective position estimations to better capture the effects of good — or bad — localisation.

situation. Let us assume we have team mate A standing relatively close and at a good passing distance. Another one, B, is also close to it in terms of bearing, but behind it in distance. With B having a broader slope, he will add some potential to A, who should then attract the pass — this is what we want. However, if A had a broader slope he might "leak" enough potential to B, who is further back, triggering a pass in that direction, which would be less desirable in this situation.

Opponents in turn are considered more dangerous when they are close (as their behaviour is completely unpredictable) and the effects of a poor position estimation of opponents are also more pronounced, so their (negative) influence should broaden the closer they are.

The goal attractor has a constant slope, again, as the distance to it is considered irrelevant.

Cut–off These values are constant, and of opposite signs with the respective maximum heights: for team mates, the cut–off value is slightly negative in order to disfavour directions that are well outside of any team mate. Conversely, bearings that are clear from any opponent player should be slightly attractive. This will have an important fine tuning effect as we shall see later.

Calculation of the potential

Once all the parameters are set for an object, we can calculate the potential it creates. For some attractor $\mathrm{A}(c, h, r, s, \gamma)$ and some bearing ϑ (with all operations again being modulo 360), we simply have

$$p_{\mathrm{A}}(\vartheta) = \begin{cases} h, & \text{if } |\vartheta - c| \leq r \\ \gamma, & \text{if } |\vartheta - c| \geq l \\ h + s \cdot \big(|\vartheta - c| - r\big), & \text{otherwise} \end{cases} \tag{3.2}$$

where $l = (\gamma - h)/s + r$ would be the distance between centre and the point where the slope is cut off.

As mentioned earlier, choosing this simple, piecewise linear form allows for a relatively inexpensive computation but still giving us all the flexibility and versatility we need.

3.3.3 Implementation

With these definitions and comments made we can now look at the implementation of our concept. We have moved the source code of the python function `check_for_pass()` into Appendix B on page 73. The listing is accompanied by comments on how different steps of the algorithm are implemented, the main focus laying on efficiency and computational inexpensiveness.

This very function is run on the robot every time it has successfully grabbed the ball. If the decision is made to pass the ball, the robot has about two seconds to execute the following actions.

3.4 Pass execution

We shall close this chapter by looking at the execution of the pass and the involved cooperation and communication between sender and receiver.

3.4.1 Lining up

When a pass is to be played it is crucial that the ball is kicked as precisely as possible along the suggested bearing. In terms of execution, the robot will have to turn to that bearing initially based purely on its localisation.

Poor localisation

Let us give an example situation. If, for instance, a pass should be played at 35° and the sender thinks his own heading is $-1°$, he will have to turn $+36°$ before kicking the ball. However, if the robot is badly localised (i. e. his *real* heading is, say, $-17°$) such an "open loop" or "feedbackless" turning would result in an offset of, in this case, 16° — which could lead to kicking the ball directly in the hands of an opponent player!

Obviously, it would be much better if the robot could "see" the receiver and line up the shot based on this visual feedback. Luckily, there is already a robot recognition algorithm in the NUbot code base, [9, 17]. It was developed by K. Hong, but its use in the game code was, at the time, only limited to dodging opponent players. Being relatively expensive in processing time, error prone in reporting distances and as dodging could be done more easily (and reliably) using the infra—red sensors, it was turned off quite soon after the 2005 RoboCup.

However, this useful piece of code is given a new application now — we use it to visually scan for a team mate in the vicinity of the passing bearing. If a team mate is spotted close to the passing bearing, we assume it to be the receiver of the pass, and then use it to visually line up the shot. That way, we are much more independent of potentially poor localisation.

Robot detection

There are, however, several issues involved in "seeing robots" — most of them are, again, due to the poor quality of the camera. We do not want to go into the details of the code (again, see [9] for more details), but let us make the following comments.

Robot detection is based on the colour blobs from the robot uniforms (as opposed to the black and white pattern or the shapes we humans probably use most when "seeing" an aibo), so it is important that the camera captures as much of the uniform as possible.[8] Due to shadowing under the head, it is very hard to see a robot that is facing us straight. The more it is turned sideways, the more we see of the side patch which is less shadowed, and the more likely the algorithm will "catch" it. This is shown in Figure 3.4 on the facing page, a picture taken with another NUbot.

[8] As the blue uniforms for the robots have a very dark colour, which is much harder to classify than the relatively bright red, we use for all our experiments red uniforms, that is "we" are Red Team, and the opposition is Blue Team.

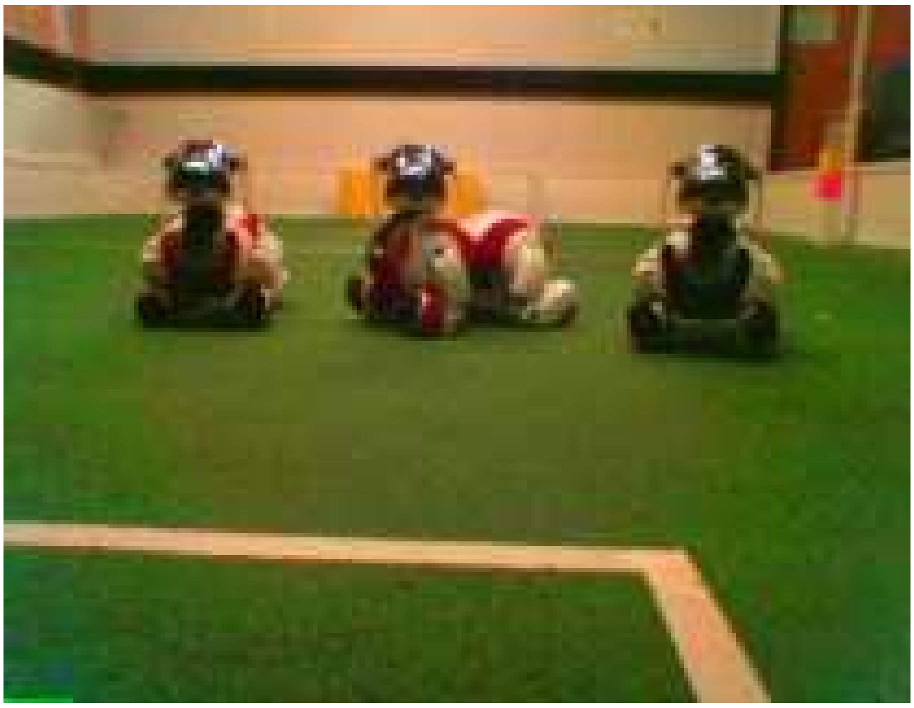


Figure 3.4: *Observe the shadowing on the front patch of the left robot compared to the red on the side patch of the second. The third robot wears a blue uniform. Picture taken directly from another robot.*

Another issue is related to the tuning of the colour look–up table which is currently only loaded when the robot is boots up. In the currently implemented colour reduction technique, a separation has to take place between orange, “red–orange” and red, confronting us with a certain trade–off. If the table is configured more aggressively towards red (i. e. the soft–colour “red–orange” capture more shades of red), robot detection will work much better and the chance of actually “seeing” the receiver is much higher. The drawback is that red robots will also be frequently mistaken for the ball which confuses the Kalman Filter and ultimately deteriorates the ball handling capabilities in general.

However, if we include more orange, the ball is seen more reliably, but red robots will be picked up only in rare cases. Ideally, it should be made possible to change between two tables “on the fly”, i. e. use one table in normal game mode, and another one (with more reds) when lining up a pass.

In any case, we need the receiving robot to turn sideways while the sender is lining up the shot. The required communication for this will be explained in the next section.

3.4.2 Communication

In order to have more reliable passes (or, more precisely, to have the ball kicked along the exact desired bearing) we want to use visual tracking for lining up.

This requires the receiving robot to turn sideways so that a good and non–shadowed portion of his uniform can be seen. Also, once the ball has been kicked, we have got some additional information about it (its rough bearing and velocity), which can be used in a "manual" update to the Extended Kalman Filter in order to improve the ball position and velocity estimation.

Broadcasting

For the required communication, we extended the communication that is taking place between the robots. Currently, each robot only broadcasts its own estimates of ball position, own position, variances on both and a behaviour message (such as "ball grabbed", "ball being kicked", etc.).

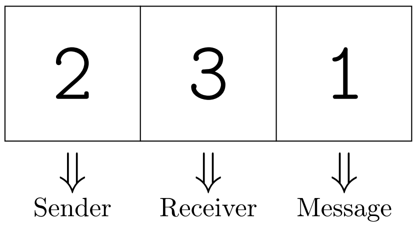
To facilitate and speed up implementation, we simply added another integer value to an existing TCP packet, which is sent out every five frames (i. e. about every 166 ms). It contains the actual message along with sender and receiver id, by using the following encoding:

– The hundreds correspond to the sending robot id,

– the tens to the intended receiver (addressee),

– the ones to the actual message.

Note that the goal keeper (he always has id 0) is never considered for passing, as trying to pass to him would be too dangerous and have no real advantage. For instance, a typical message would be:



We include both sender and receiver id in the message so that the message can be broadcast without targeting anyone in specific (at the TCP packet level), which facilitated the implementation. Each robot that receives the message then decides whether it was meant for it or not and reacts appropriately.

Messages

Currently, we have three defined messages. The first two messages (id `1` respectively `2`) mean, loosely speaking: *"Turn left [resp. right] so that I can see you"*, relating to the visual lining up of the kick.

The third message (id `3`) stands for: *"I've* just *kicked the ball"*, which will allow for an improved ball position and velocity estimation.

Lining up related messages Let us return to our example above. There, player 2 is sending message `1` to player 3, and thus tells him to turn toward a certain angle left of the imaginary line connecting the two robots, as depicted

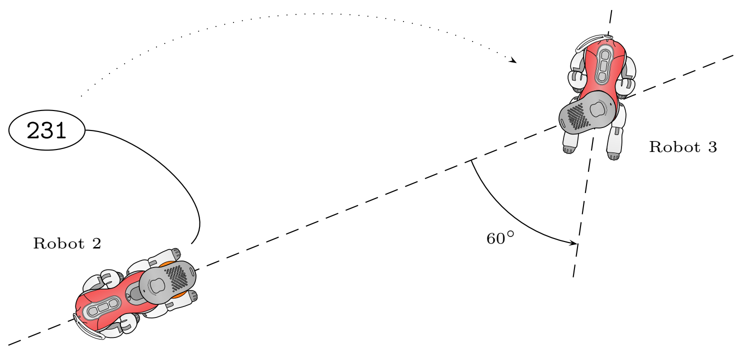


Figure 3.5: *Robot 2 sending message* 1 *to robot 3, which turns accordingly.*

in Figure 3.5. This angle is currently set to 60°, a compromise between fully facing the sender / ball (for more reliable grabbing especially if the sender is rather close, but making it hard to be tracked), and exposing more of the side patch of the uniform for higher "visibility" (which allows more precise kicking, but the receiver may not be able to turn back in fast enough to catch the ball once it has been kicked).

The decision, whether the receiving robot should turn left or right is made by the sender depending on the pass bearing relative to the receiver in case of a through pass (i. e. when the ball is not kicked directly at the team mate, but slightly off to one side).[9] If the pass is to go straight to the receiver, the receiver shall turn either left or right, whichever is the shortest turn from his current heading.

When a player receives the message to turn left (or right) relative to some sending robot, he calculates the bearing of that robot, and turns accordingly on the spot. The sender starts broadcasting the turning message as soon as the passing algorithm has finished calculating bearing and receiver, and continues to broadcast it until he has kicked the ball. He then sends out message 3 with the next TCP packet available (but only once).

Using this technique, the receiver is "locked" at his current position (based on which the passing decision has been made), turning to — and holding — the angle relative to the sender. When the ball is finally kicked, (more precisely when he does not receive message 1 or 2 any more), he is "released" from this forced sideways angle and (back in the standard ball chasing mode) turns back to face the ball. However, it updates it's Extended Kalman Filter as described in the following.

Ball kicked message By using a specific kick, we have a rough idea of the velocity of the ball (as mentioned earlier, the ball usually reaches $40 - 60$ cm/s after the kick), and as it is meant to travel along a certain bearing, we can

[9] It is easy to see that through passes will only occur if the core radius for team mates $r_{\mathrm{T},0}$ is non–zero.

include that information in the Extended Kalman Filter, similar to the odometry update described on page 12. This is done the instant when the player receives message 3.

Normally, we assume that the change of the ball's velocity is negligible, so the sixth and seventh component of the vector added to $\hat{\boldsymbol{x}}_{k-1}$ in (2.2) are zero. When the receiver gets the message that the ball has just been kicked (hopefully very close towards him), these two components are set to $50 \cdot \cos(\pi + \vartheta_{\text{sender}})$ respectively $50 \cdot \sin(\pi + \vartheta_{\text{sender}})$ for that particular frame, where sender is, of course, the relative bearing of the sender. Also, we drastically increase the model covariances for the ball velocity in that frame, to put a strong emphasis on the measurements, away from the model (which does not include kicks).

This concludes this chapter on algorithm design. We shall now see, how the passing algorithm performs in simulation, and how it works integrated in the current NUbot code along with the modifications to the communications, behaviour and Extended Kalman Filter.

CHAPTER 4

Demonstration

We demonstrate our algorithm and present results from simulations in MATLAB*, as well as from practical tests with the robots.*

4.1 Introduction

Now that we have described the concepts and approaches for its implementation, it is time to demonstrate the features of our passing algorithm and show some results from actual practical tests with the NUbots.

The development and testing of the passing decision algorithm was done initially using MATLAB. The powerful debugging tools as well as the simple yet flexible graphic capabilities of MATLAB made it an ideal tool for testing and fine tuning the many parameters involved. It was only in a second step, that the algorithm was ported to python™ and tied into the existing NUbot software.

Obviously, it will be hard to show some real game moves on paper, as the dynamic development of the game can only be captured well using video. However, we shall attempt to display some laboratory results using a series of pictures taken from a video.

So let us first demonstrate the key functionality of the decision making algorithm and later review some real game moves from the actual soccer pitch.

4.2 Matlab Simulation

We shall start by showing how the region of interest changes along the field.

4.2.1 Region of interest

As we have laid out in Definition 3.1 on page 34, this is the only section of the horizon within which passes are allowed to be played. It changes depending on the sender's position on the field.

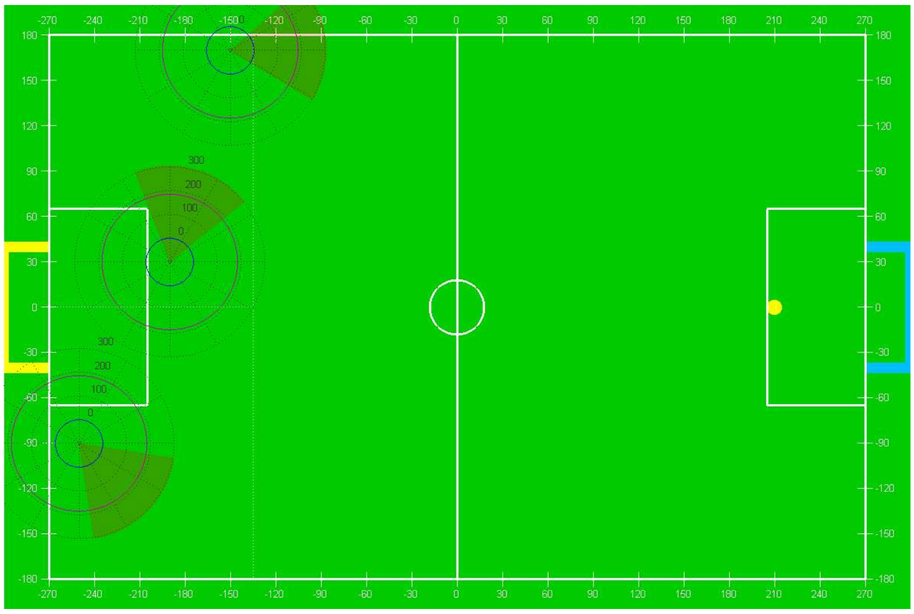


Figure 4.1: *Region of interest in the last quarter of the field.*

Defence

We wanted the region of interest to always face away from the x–axis, as a pass crossing the field in front of your own goal would bear too much risk of an opponent catching the ball and having an ideal position to take a direct shot at the goal.

The dotted white lines indicate the different areas on the field involved, that is one at $x = 135$ showing where the defensive mode is activated and one in the middle that separates left and right half of the field (where the region of interest is to face either towards the left or right side line).

To visualise the region of interest in Figure 4.1 above, we overlaid a semi transparent "wedge" on the little radar like gauges (polar plots) which also show the potential around the horizon (blue line) together with the threshold value (magenta line).[1]

As desired, near the sides of the field passes are allowed to go up–field, but near the centre only sideways going passes are possible.

[1] Please note that the goal attractor's potential is not taken into account in the plots here.

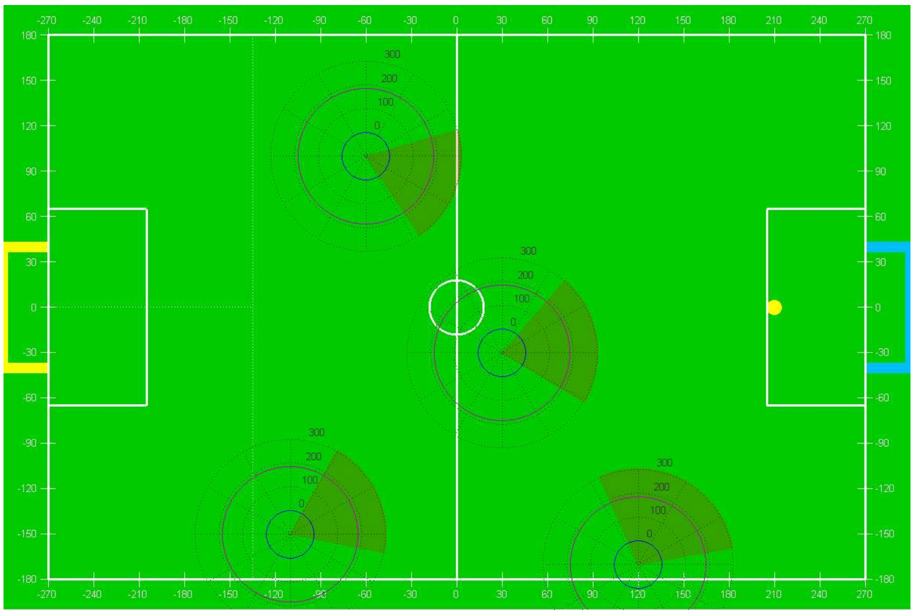


Figure 4.2: Region of interest around midfield.

Midfield

When the game moves towards the midfield, the region of interest is to be centred on the bearing of the goal attractor (indicated by the yellow dot).

In the defensive half, the region of interest covers a total of 72°. However, once the player moves beyond the half way line, it widens up, as can be seen when comparing the bottom left and bottom right gauge in Figure 4.2 above

Being still too far away from the goal attractor, the threshold remains at its constant default value for all four sample points.

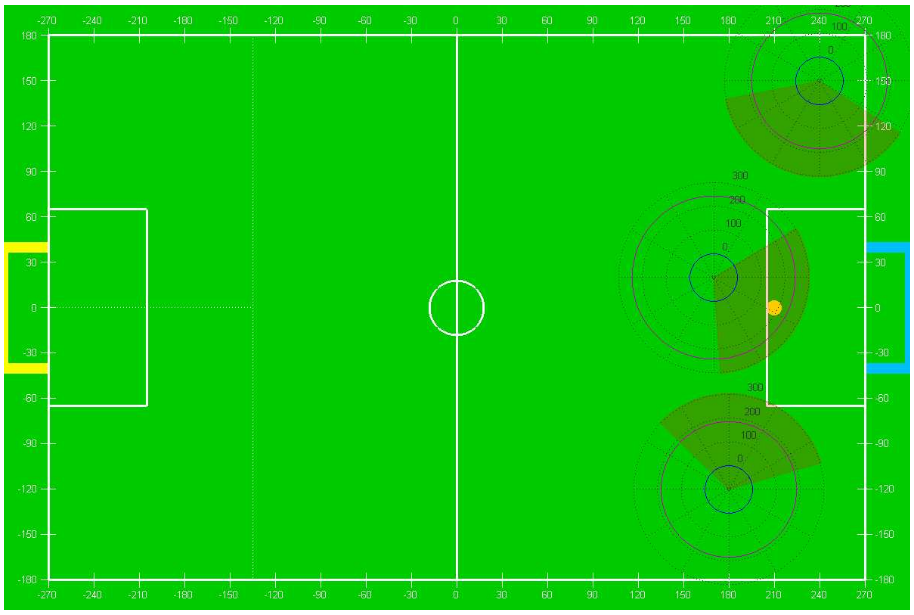


Figure 4.3: *Region of interest in the offensive third of the field.*

Offence

Especially in the offensive third of the field passes can have a tremendous impact on the team's performance. In real soccer, cross passes[2] play a key role in attacks — this is similar here (but with the important difference that we cannot kick the ball off the ground nor follow up with a header).

The desired cross pass feature is the main reason why the goal attractor is located slightly in front of the goal, not directly on the goal line. That way, the region of interest is opened far enough to allow for cross passes as well as “escape” passes, as can be seen in the top right gauge of Figure 4.3.

Being close to the goal attractor, we can see how the threshold value at the sample point in the middle is increased.

With the region of interest behaving as desired, we shall now demonstrate clipping effects.

[2] That is passes played from the corners inwards in front of the opponent goal. They are an important tactical move as they allow to bring the ball back into play when an attacker was boxed into a corner, can effectively counter defensive patterns that are specific to just one side of the field or force defenders to turn rapidly by large angles.

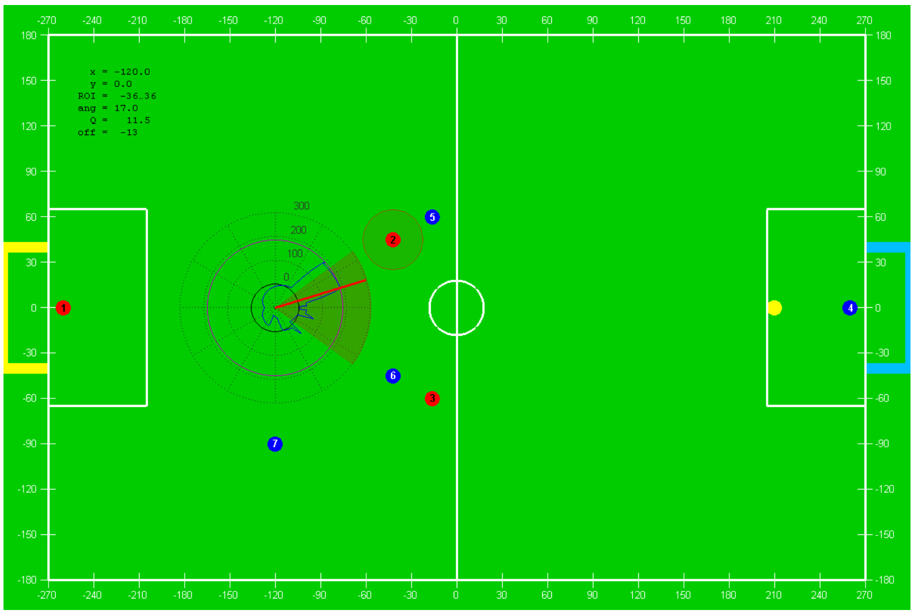


Figure 4.4: *Example situation to show clipping effects.*

4.2.2 Clipping effects

As we have mentioned in Subsection 3.3.1 on page 33, objects hidden behind others should, in certain circumstances, have no effect on the object in front of them. For instance, an opponent player in line, but half a meter behind a team mate, should have no serious impact on the passing decision whether to pass to that particular team mate.

To demonstrate how our algorithm takes care of these clipping phenomena (more details in about the implementation of that feature can be found on page 76) we have set up a testing situation in Figure 4.4 above. Again, the blue line corresponds to the potential along the horizon, scaled in the indicated polar grid.

To give a better look "behind the scenes", we also created Figure 4.5. There, the upper plot shows each object's *individual* contributions to the potential field along the 360 degrees of horizon (for the same situation as above). The lower plot displays the resulting overall potential.

Looking at the first subplot we can see the goal attractor's wide spread influence, as well as a number of "high rising" positive peaks. These correspond, of course, to the team mates. When comparing this to the lower plot, we can see that opponent player 5 has only little influence on the potential created by player 2 — as player 5 is hidden behind him. Conversely, player 6 removes most of the positive influence of player 3, as the opponent player, this time, occludes the team mate's influence.

Also, observe how only the cores of the potentials are clipped, not the sloped or cut–off parts of the potential. This explains the little spikes in the overall potential near player 2 and player 6.

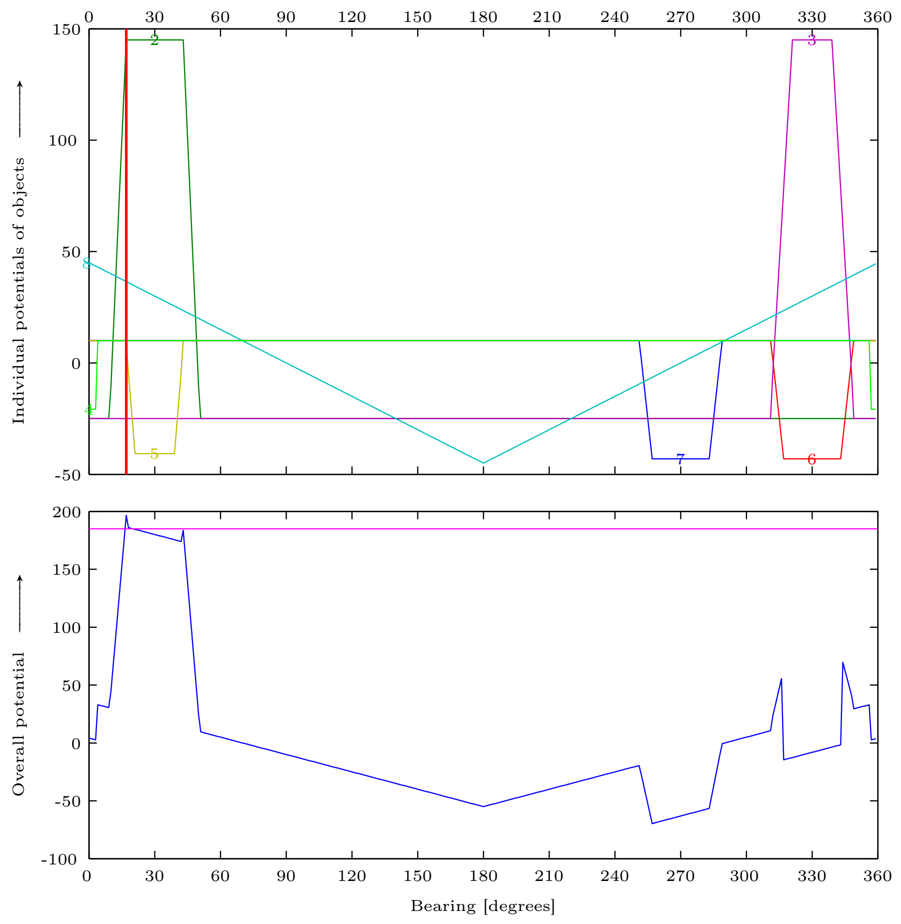


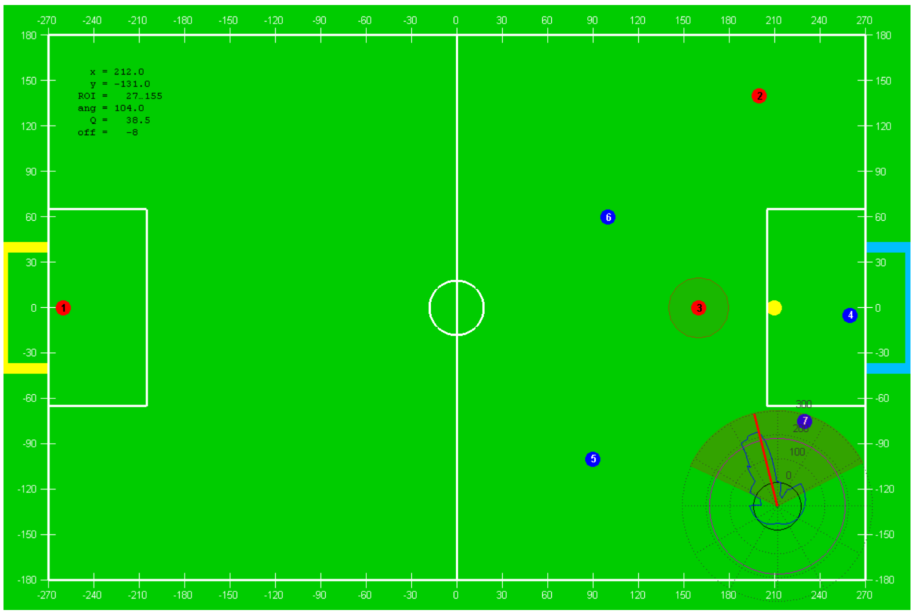
Figure 4.5: *Graphs for the individual and overall potentials for our example situation.*

As the potential exceeds the threshold at 17°, which is inside the current region of interest, a pass is triggered as indicated on the plots. Our MATLAB routine also highlights the player that the algorithm intends to be the receiver.

With these details mentioned, let us take a look at some realistic game situations and discuss the potential field's response to them.

4.2.3 Game situations

We shall now present four typical game situations to evaluate the suggestions of our passing algorithm. The accompanying detailed plots for the potentials have been moved to Section A.2 in Appendix A.



```
  x = 212.0
  y = -131.0
ROI =   27…155
ang = 104.0
  Q =   38.5
off =   -8
```

Figure 4.6: *The same situation as in Figure 3.1.*

Situation 1 — Offence, cross pass

Let us begin with the situation introduced in Figure 3.1 on page 30. The player currently in possession of the ball cannot take a direct shot at the goal as he is blocked by opposition player 7. Even if he dodged around the player, he would still have a bad angle for a good shot. The other players are rushing back to defend their goal, but team mate 3 is free in the middle of the field. As discussed earlier, we would like the player corned on the bottom right to play a cross to his team mate in front of the goal.

As desired, the algorithm suggests the pass towards player 3. It is the goal attractors slope that creates the unique maximum, which also lays closest to the goal (among the bearings of the team mates core potential).

That way, the resulting pass will be a nice cross with a certain through pass character, as the ball is not kicked directly at, but in front of the receiver.

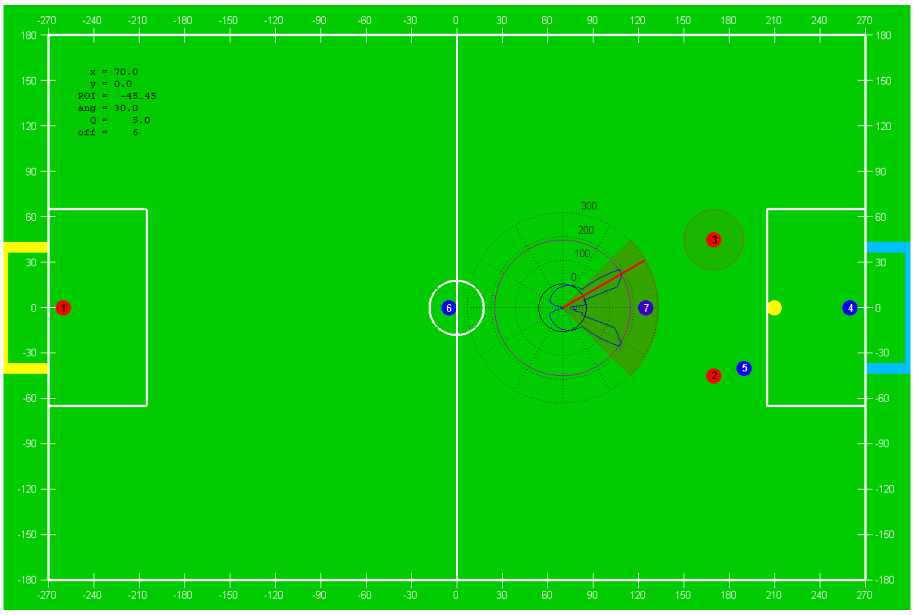


Figure 4.7: *A simulated offensive situation.*

Situation 2 — Offence, forward pass

Another offensive situation is shown in Figure 4.7 above. However, this time we made it artificially symmetric: sender and players 4, 6 and 7 are exactly in the middle of the field; players 2 and 3 in turn are at the same height and equal distance from the middle line. We chose this setting to highlight the fine tuning effects of the cut–off parts of the potentials, as both players 2 and 3 attract a pass here (the potential exceeds the threshold near both players).

Observe player 5. He is mostly hidden behind player 2, thus has not much effect on the latter. However, it is his cut–off part that favours player 3 for passing — whom we would prefer to be the receiver here as he is "more clear" of opponents than player 2.

Hence, even in mostly (and unrealistically) symmetric situations, our algorithm prefers one player over another based on how far he is away from opponent players.

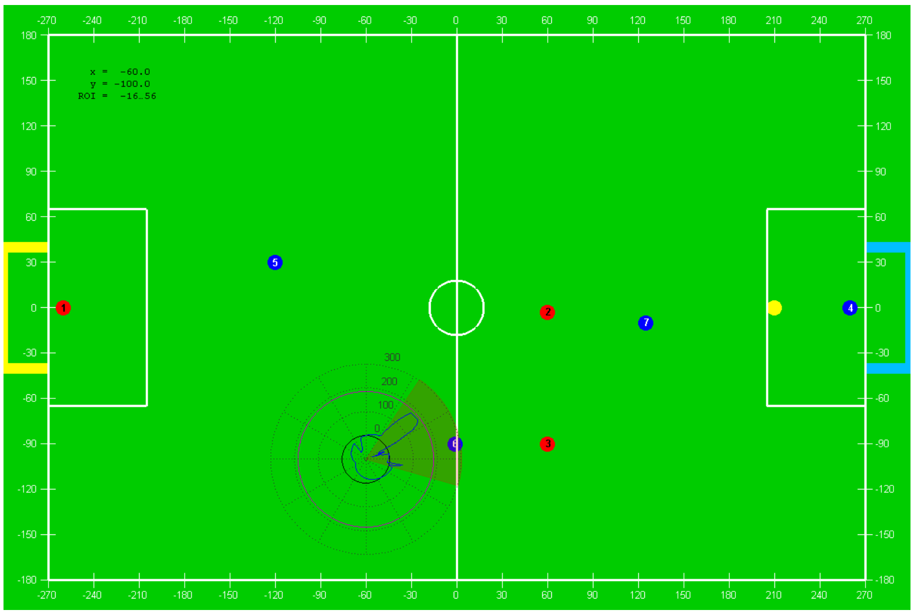
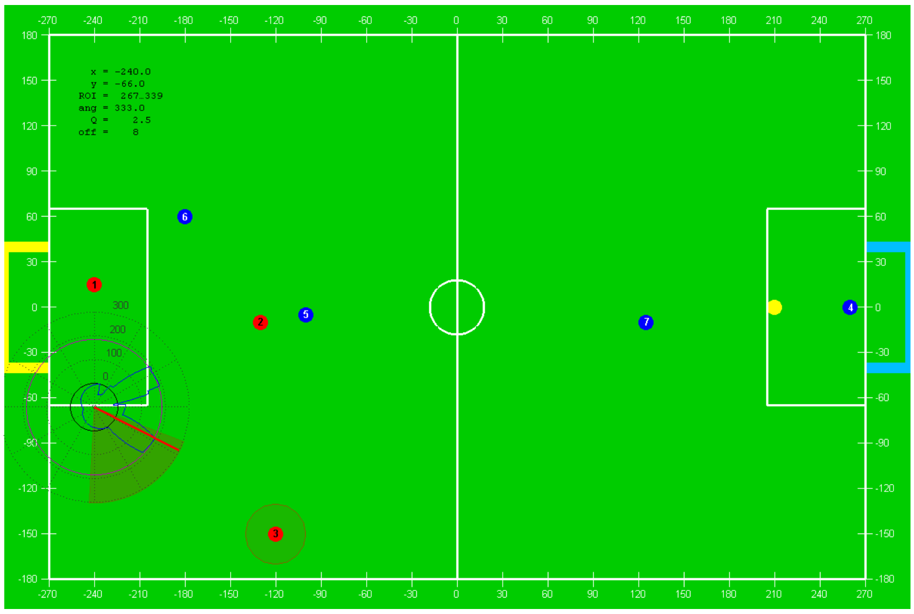


Figure 4.8: *A simulated midfield situation.*

Situation 3 — Midfield, forward pass

Let us move further towards midfield, as shown in Figure 4.8 above. For a change, there is no pass suggested in this situation here.

Player 3 is hidden behind an opponent player and player 2 is too far away to be considered for passing.



```
   x = -240.0
   y = -66.0
ROI =  267…339
ang = 333.0
   Q =     2.5
off =     8
```

Figure 4.9: *A simulated defensive situation.*

Situation 4 — Defence, clearing pass

This is another typical situation in RoboCup games. It could result from a successful dive by the keeper, who just managed to hold a shot at the goal by player 6 for instance.

The player currently in possession of the ball chased after it and should now play a clearing pass. Looking at the potentials alone, player 2 would attract a pass in this situation. As we are in the last quarter of the field, however, the region of interest does not allow by design any passing in that direction. This restriction is particularly useful here, as otherwise a pass to player 2 may cause the ball to go dangerously close to opponents.

4.3 Practice

Let us now discuss a few results from real life test, i. e. with the robots on the field.

4.3.1 Practical problems

There are a number of issues to be mentioned first. In the simulations, for instance, we know exactly where all the players are, from the own as well as from the opposition. On the real pitch however, we have error prone localisation of the sender as well as all the other players on the team. This imprecise or bad localisation will, of course, have an impact on the passing decision as well as the execution of the pass. To a certain degree, our implementation can cope with these uncertainties by using non–zero core radii on the one hand (which can be interpreted as an *area* where the receiver could be as compared to a precise spot), and, on the other, using our flexible visual tracking (flexible as it allows to track the receiver even if he is not seen exactly where is expected to be seen).

Another major problem is, so far, that in real game situations we have no information whatsoever about the opponents. In the future, robot detection will have to be improved to the point where it can pick up robots from both teams in a much more reliable way than currently available. Once this is available, our passing algorithm can be used in real game situations. For our testing, we worked around the issue by placing switched off opponent robots ("bricks") in know positions and "hard coding" their positions into our algorithm.

Also, as mentioned earlier, we have to find a compromise in the colour look–up table tuning between more reliable ball handling or better lining up of passing kicks.

4.3.2 Offence

Our first set–up corresponds to a typical offensive situation. Figure 4.10 on the following page displays eight frames of a video sequence taken from a typical passing situation. Let us go through them one by one.

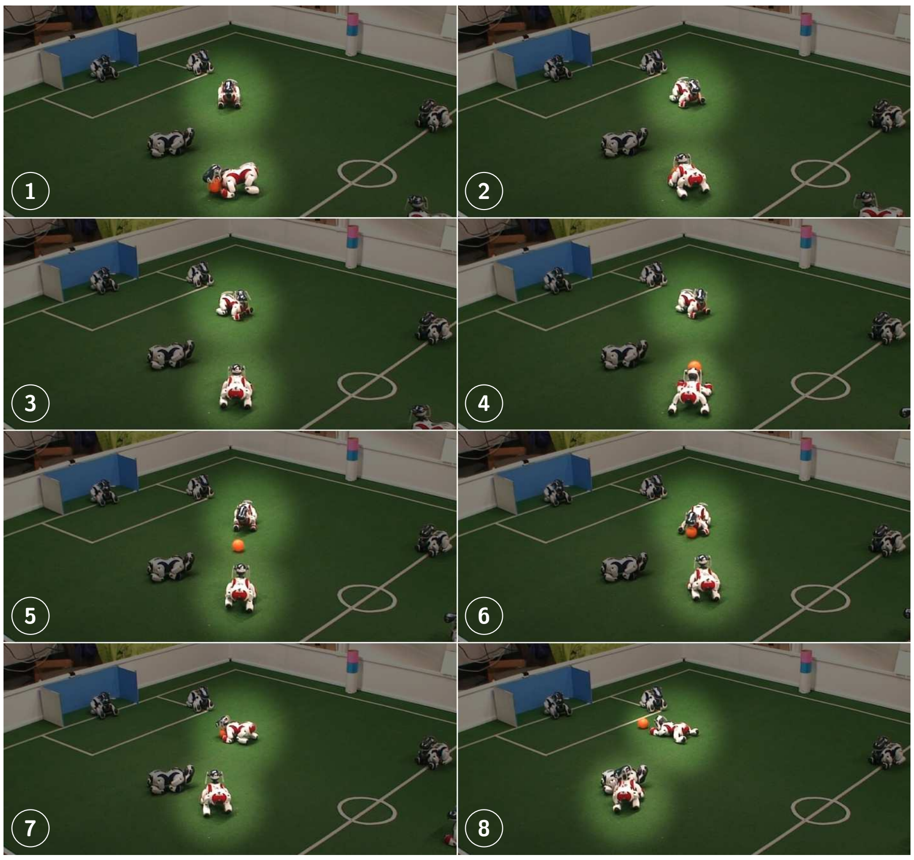


Figure 4.10: *A real offensive situation.*

1. The robot on the bottom has just grabbed the ball and is executing `check_for_pass()` to evaluate passing options.

2. The decision is made to pass the ball to the player in front of the goal. The sender starts broadcasting his intention, and the receiver starts turning sideways so that the sender can see him.

3. The sender has finished lining up the pass and executes the kick. At the same time he broadcasts the kicking message.

4. As soon as the receiver stopped receiving the sender's *"turn so I can see you"* message, the receiver started turning back so that his body would face the ball again. He also manually updated his Extended Kalman Filter the instant he received the "ball kicked" message.

5. The ball is on the way and the receiver gets ready to grab the ball

6. He has caught the ball and, having evaluated passing options, decides to take a shot at the goal.

(7) Having lined up the kick . . .

(8) . . . he takes a shot.

This would be a typical situation occurring frequently in Four Legged League games. If he did not attempt a pass, the player on the bottom of the screen would either — with the current software and strategies — first try to dodge the opponent player in front of him and then take a direct shot at the goal, or he would let go of the ball potentially just handing it over to the opponent. In case of shooting at the goal, the chances of scoring would be rather slim as he is relatively far away from the goal (and with the three second rule, he would not have enough time to close in much further).

However, playing a pass as shown here would not only have the advantage of keeping the ball away from opponent players, but also hand over the ball to a player that is in a much better position for scoring.

4.3.3 Defence

A different set–up would be the following rather defensive situation, similar to the one discussed on page 54. As displayed in Figure 4.11 on the following page, the goal keeper managed to hold a shot at the goal, knocking the ball off to the side. This is what happened then step by step:

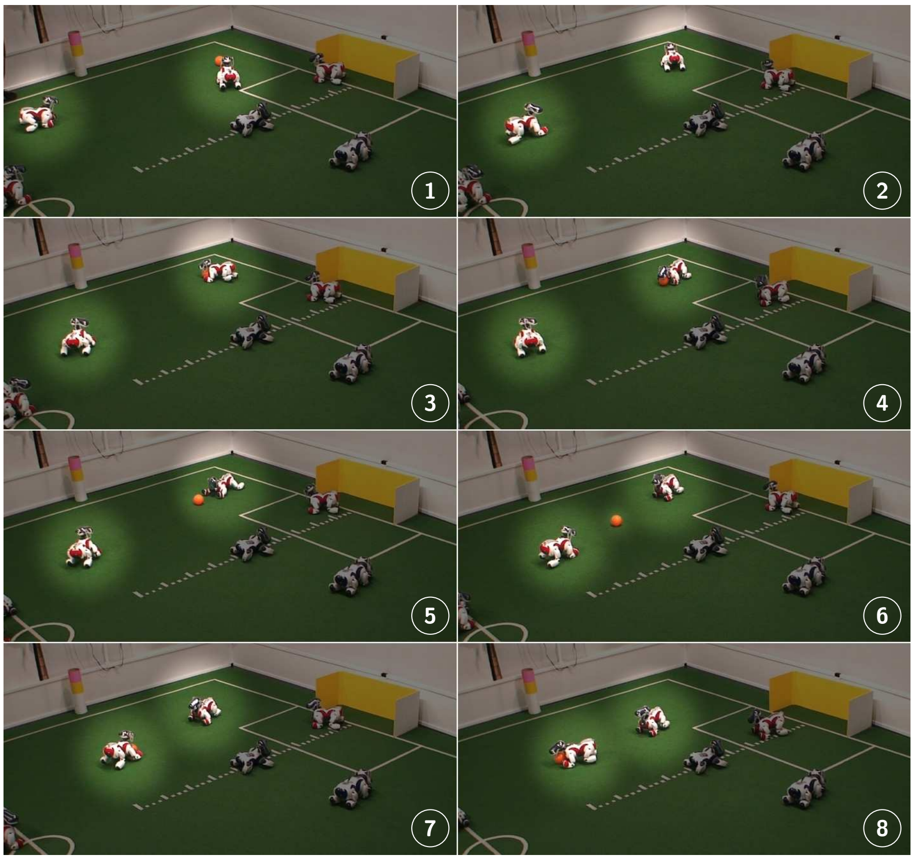


Figure 4.11: *A real defensive situation.*

1 One of the defenders chases after the ball.

2 Having successfully grabbed the ball, he evaluates his passing options, and decides to play a pass to the other player near the side line.

3 While turning towards the calculated passing bearing, the sender broadcasts his intent, making the receiver stay put, but turn sideways for better detectability.

4 The sender has seen his team mate and fine tunes his orientation.

5 Having received the message that the ball is kicked, the receiver updates his Extended Kalman Filter and turns back, getting ready to take the ball.

6 The ball is on the way and the receiver continuously corrects his position to catch the ball.

7 Having grabbed the ball, the receiver evaluates his options . . .

8 . . . and decides to run forward with the ball.

The current software would have “blindly” kicked the ball up the field, roughly in the direction of the goal. As this would be done purely based on localisation, chances are fairly high that he would either kick the ball out of bounds, or that he would kick it too far infield, running the risk of just handing it back to the opposition.

With these results we close this last chapter and move on to the Conclusion of the thesis.

Conclusion

In the first Chapter, we introduced the robots used in the Four Legged League, commenting on their information gathering and processing hardware as well as their means to interact with the outside world. Selected rules from the official rule set provide background information on the game process and the particularities of this league.

The following chapter gave an overview of the existing software run on the robots. We discussed the Vision, Localisation & World Modelling, Behaviour and Locomotion modules which "transform" the ordinary aibo entertainment robots into little soccer players. A modification was suggested to the potential field, allowing for the creation of "potential segments" in addition to the already implemented point and (infinite) line sources, which helps for a better and more precise prevention of illegal defender penalties and gives more flexibility when using the potential field for other purposes.

We also proposed a modification to the Extended Kalman Filter, improving the ball position and velocity estimation which ultimately enhanced the ball handling capabilities of the robots especially when attempting to grab the ball.

Chapter 3 was then concerned with the design of the actual passing algorithm. We motivated our approach by contemplating key features and characteristics of passes in general, and how this could be translated into an algorithm. Inspired by concepts from fuzzy logic, we set up a one–dimensional potential field along the line of horizon.

The team and opponent players as well as the goal attractor were sources (or sinks) of potential, and the superposition of their potentials was assumed to indicate good (or bad) directions to play the pass in. A threshold potential had to be exceeded to trigger a pass, provided also that this trigger occured within a certain region of interest.

Once the decision was made, the execution of the pass involved cooperation between sender and receiver. To improve the precision of the kick, we used visual feedback to track the receiver of the pass. For that, he was required to turn sideways to increase the chance of being picked up by the robot recognition code. Knowing when and in what direction the ball has then been kicked, the receiver also used that information to provide a better update to his Extended Kalman Filter.

The last chapter served mainly to demonstrate and test our algorithm. Using a Matlab implementation, we showed the evolution of the region of interest along the field, tested the clipping of potential when some players were hidden behind others and finally commented on the decisions made in different typical game situations.

Using a number of frames taken from a video recording, we then presented our overall work in two classical game situations, describing step by step the actions taken by sender and receiver, also highlighting the important benefits that resulted from the new passing behaviour.

As mentioned in the introduction, passes are unquestionably an indispensable “tool” in real team sports. It is clear that an efficient, reliable and also tunable passing behaviour would constitute a major contribution to the overall game performance and a key advantage over other teams that do not play passes.

We have shown that our approach gives satisfactory and consistent results in simulations; practical tests have also shown good results, even if better or more reliable ball handling capabilities would further improve the efficiency of the passing, and more consistent robot detection by the Vision module would allow for better lining up of the shots.

In that regard, this thesis is at least a proof of concept that it is possible to introduce such high level behaviour in the Four Legged League. Previously, there was little motivation to invest in further research and development of more robust and precise robot recognition, resulting in rather unreliable robot detection (which additionally only works for red robots). This limits not only the performance of lining up the shot, but also creates one important shortcoming: in real game situations, we do not know the positions of the opponent players.

If robot detection was at a point where it would deliver adequate information about the opponent positions, our algorithm could be used right away without any modification. All we needed to do is to quickly sweep the region of interest with the camera, and once all information is gathered, run our decision making algorithm. The scanning would require some time from the three seconds until the robot would have to kick the ball, but, in our experience, this should not be a problem as the turning and lining up generally seems to require very little time.

We also came to suggest some of the following points which should be addressed in future work: Make robot detection more reliable, to deliver more precise information about both red and blue robots in sight, independent of their orientation (as we have no influence on the heading of opponent robots). Once this is done, modify the behaviour that a player (who considers passing in the first place) scans the region of interest plus some safety margin on both sides to also take robots into account which are just outside of it. Having gathered all the relevant information, run the passing algorithm and possibly carry out the pass execution.

If the Vision module also returned some measure of confidence about the different robots seen, one should include this information in the core radius for the potentials later on, as that way the amount of uncertainty about its position can be included in the evaluation. As mentioned in the third chapter, one should also come up with some measure of confidence about scoring a goal from the current position (or after potentially dodging an opponent in the way) in order to balance between taking a direct shot or playing a pass.

Furthermore, one could try to include the heading information of the players to create non–symmetric potentials. Clearly, there is no point in playing a through pass towards the goal if the receiver is facing backwards (which is generally unlikely with the NUbot code, but still possible).

Finally, one should also modify the receivers behaviour while the ball is travelling in such a way, that he only *strafes* left or right to catch the ball, instead of chasing after it always facing it straight on. This would help to make much better use of through passes, which are an important feature of our algorithm.

APPENDIX A

Proof & additional plots

The following proof has been moved here not to clutter up the main body of the thesis. We also display some complimentary plots to the game situations presented in Chapter 4.

A.1 Variances after coordinate transformation

We shall now argue, that it is safe to use $r_i^{\mathrm{r}}(k) = r_1(k)$ for $i = 1, 2$ as claimed on page 23 at the end of Subsection 2.6.1.

Let us assume a standard situation as depicted in Figure 2.4 on page 22. We have two measurements for the ball, that is its distance and bearing d_{b} and ϑ_{b} (for reasons of simplicity we shall drop the index b for the rest of this section). The measurement errors or variances of both are assumed to be independent from each other, i. e. not correlated. Furthermore let us denote

$$
\begin{aligned}
E\{d\} &= \bar{d} & E\{\vartheta\} &= \bar{\vartheta} \\
E\{(d-\bar{d})^2\} &= \sigma_d^2 & E\{(\vartheta-\bar{\vartheta})^2\} &= \sigma_\vartheta^2
\end{aligned}
$$

and $\boldsymbol{x}^{\mathrm{r}} = \boldsymbol{f}(d, \vartheta)$ the function (2.13) on page 21 that relates the original measurements to the new relative coordinates x^{r} and y^{r} (i. e. performs the transformation from local polar coordinates to local Cartesian coordinates). For small variances and ignoring higher order terms, we can assume

$$
\begin{aligned}
\begin{pmatrix} x^{\mathrm{r}} \\ y^{\mathrm{r}} \end{pmatrix} &= \boldsymbol{f}(d, \vartheta) \\
&= \boldsymbol{f}(\bar{d}, \bar{\vartheta}) + \underbrace{\left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}^{\mathrm{r}}}\right|_{(\bar{d},\bar{\vartheta})}}_{:=\boldsymbol{J}} \cdot \begin{pmatrix} d-\bar{d} \\ \vartheta-\bar{\vartheta} \end{pmatrix} \qquad \text{(A.1)}
\end{aligned}
$$

If we now take the expectation value on both sides of (A.1), we find

$$
\begin{pmatrix} \bar{x}^{\mathrm{r}} \\ \bar{y}^{\mathrm{r}} \end{pmatrix} = \boldsymbol{f}(\bar{d}, \bar{\vartheta}) \qquad \text{(A.2)}
$$

as

$$
E\left\{\begin{pmatrix} d-\bar{d} \\ \vartheta-\bar{\vartheta} \end{pmatrix}\right\} = E\left\{\begin{pmatrix} d \\ \vartheta \end{pmatrix}\right\} - E\left\{\begin{pmatrix} \bar{d} \\ \bar{\vartheta} \end{pmatrix}\right\} = \boldsymbol{0}
$$

Now, looking at the variances, we find by using (A.2) in (A.1) and with $\sigma_{\mathrm{t}}^2 := d^2\sigma_\vartheta^2$

$$\operatorname{var}\begin{pmatrix} x^{\mathrm{r}} \\ y^{\mathrm{r}} \end{pmatrix} = \boldsymbol{J}\begin{bmatrix} \sigma_d^2 & 0 \\ 0 & \sigma_\vartheta^2 \end{bmatrix}\boldsymbol{J}^T$$
$$= \begin{bmatrix} \sigma_d^2\sin^2\vartheta + \sigma_{\mathrm{t}}^2\cos^2\vartheta & -\sigma_d^2\sin\vartheta\cos\vartheta + \sigma_{\mathrm{t}}^2\sin\vartheta\cos\vartheta \\ -\sigma_d^2\sin\vartheta\cos\vartheta + \sigma_{\mathrm{t}}^2\sin\vartheta\cos\vartheta & \sigma_d^2\cos^2\vartheta + \sigma_{\mathrm{t}}^2\sin^2\vartheta \end{bmatrix} \tag{A.3}$$

At this point, let us state and prove the following lemma.

Lemma A.1 (Approximation of variances)

If $\sigma_d^2 \geq \sigma_{\mathrm{t}}^2$, then from (A.3) follows

$$\operatorname{var}\begin{pmatrix} x^{\mathrm{r}} \\ y^{\mathrm{r}} \end{pmatrix} \leq \sigma_d^2\,\mathbf{id} \tag{A.4}$$

Proof: Rearranging (A.4), we find

$$\mathbf{0} \leq \begin{bmatrix} \sigma_d^2 & 0 \\ 0 & \sigma_d^2 \end{bmatrix} - \begin{bmatrix} \sigma_d^2\sin^2\vartheta + \sigma_{\mathrm{t}}^2\cos^2\vartheta & -\sigma_d^2\sin\vartheta\cos\vartheta + \sigma_{\mathrm{t}}^2\sin\vartheta\cos\vartheta \\ -\sigma_d^2\sin\vartheta\cos\vartheta + \sigma_{\mathrm{t}}^2\sin\vartheta\cos\vartheta & \sigma_d^2\cos^2\vartheta + \sigma_{\mathrm{t}}^2\sin^2\vartheta \end{bmatrix}$$

and then

$$\mathbf{0} \leq \begin{bmatrix} (\sigma_d^2 - \sigma_{\mathrm{t}}^2)\cos^2\vartheta & (\sigma_d^2 - \sigma_{\mathrm{t}}^2)\cos\vartheta\sin\vartheta \\ (\sigma_d^2 - \sigma_{\mathrm{t}}^2)\cos\vartheta\sin\vartheta & (\sigma_d^2 - \sigma_{\mathrm{t}}^2)\sin^2\vartheta \end{bmatrix}$$

i. e. the above matrix must be symmetric positive semi–definite. This is the case if and only if all the leading principal minors have non–negative determinants (Sylvester criterion, [10]), resulting in our case in two conditions:

(i) The matrix must have non–negative determinant,

(ii) the first element must be non–negative.

The first condition is always fulfilled since the determinant is identically zero; the second is satisfied with the Lemma's initial assumption $\sigma_d^2 \geq \sigma_{\mathrm{t}}^2$. $\square$

Hence, in our application, we also need to focus only on the second condition. Naturally, we cannot give exact formulas for the variances of the distance and angle measurements as these depend on too many and a number of unknown factors involved in the visual capturing and processing. However, experiments have shown that the following functions seem to give a good estimate of both variances, depending on the distance of the ball:

$$\sigma_d^2(d) = \left(\underset{[3,200]}{\operatorname{sat}}\left[\frac{d}{10}\log_{10}(d)\right]\right)^2 \tag{A.5}$$

and

$$\sigma_\vartheta^2(d) = \begin{cases} 0.02, & \text{if circle fit successful and } d < 30 \\ 0.004, & \text{if circle fit successful and } d > 30 \\ 0.02, & \text{if circle fit unsuccessful} \end{cases} \tag{A.6}$$

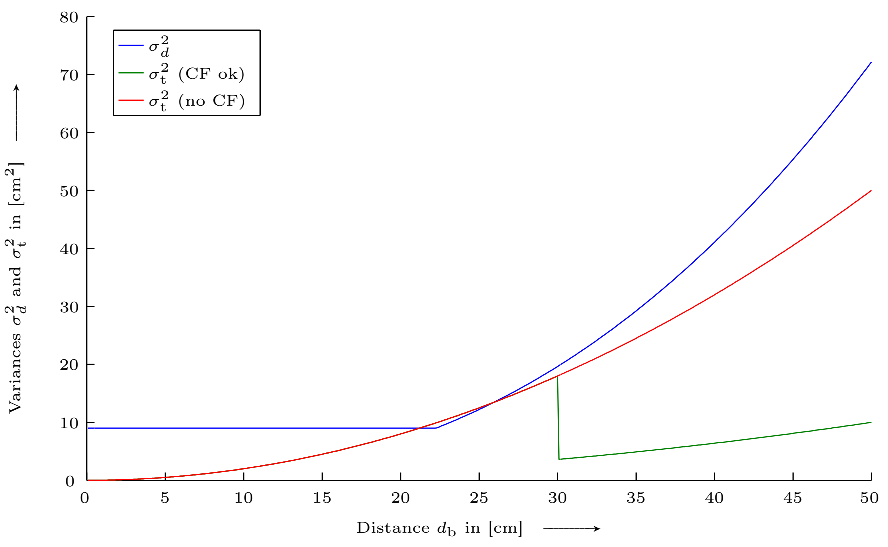


Figure A.1: *Comparison of the different variances of ball distance and bearing measurements.*

where the sat function here crops any value below 3 and above 200 to these limits.[1] The functions (A.5) and (A.6) are shown in Figure A.1. As desired, we can see that, the curve for σ_d^2 is (almost always) above the curves for σ_ϑ^2, hence condition (ii) is also (almost always) fulfilled.

We can also take a slightly different approach. With σ_ϑ^2 being close to zero, we could use the near linearity of the sine function for small angles to interpret the product σ_t^2 as the *tangential* variance of the ball, drawn in grey in Figure A.2 on the following page (it may be hard to spot it behind the actual arc that would represent the real tangential variance drawn on top of it to show how well they match).

Concerning condition (ii), we can now see that it should always be fulfilled. The variance of the distance measurement should always be larger than σ_t^2 (on the sketch, the grey arrow should always stay inside the circle): Tests have shown — and it is quite intuitive — that the angle measurements are more precise than the distance measurements, as pixel errors (or the limited resolution of the camera) for instance have a much smaller impact on the centre of the circle fitted to the ball (to determine its bearing) than on the radius (which gives us its distance). In other words, for a certain distance d, the tangential variance σ_t^2 is almost always smaller than σ_d^2.

So, in conclusion, although we may over–estimate the covariances slightly (though the over–estimation is small at short ranges), we have a conservative

[1] As mentioned in Chapter 2 on page 10, a least squares based circle fitting is used to determine the balls bearing and distance. If the algorithm cannot properly fit a circle to what is believed to be the ball in the captured picture, the bearing has to be determined using the raw vision blob, resulting in significantly less precise estimation of the angle. Also, if the ball is very close, the circle fitting is less precise as usually parts of the ball leave the narrow field of view of the camera which obviously complicates the circle fitting.

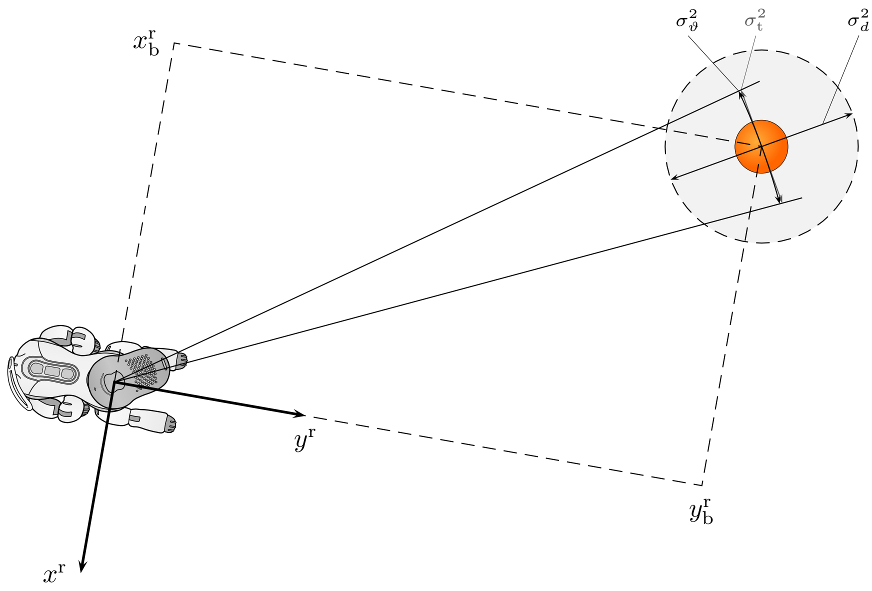


Figure A.2: *Close–up showing the different variances and how they relate to each other.*

estimate making it reasonably safe to use σ_d^2 for both the x^r and y^r "measurements" in the Extended Kalman Filter.

A.2 Additional plots

The following series of plots complements the various situations introduced in Subsection 4.2.3 on page 50.

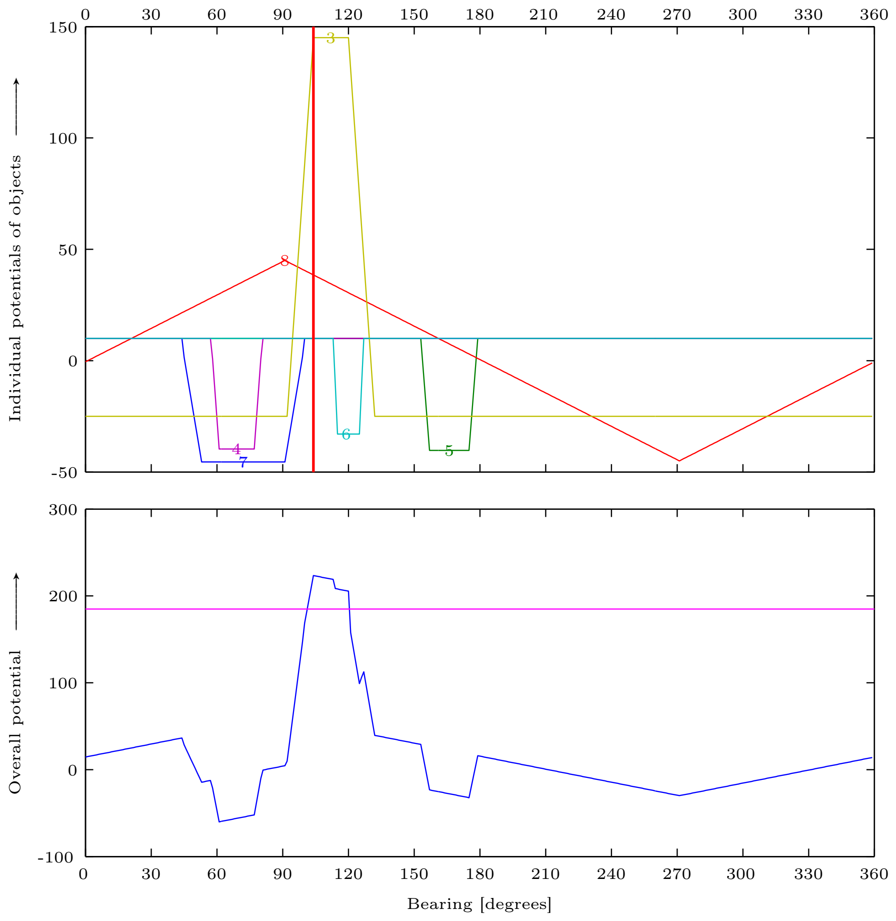


Figure A.3: *Potentials in Situation 1.*

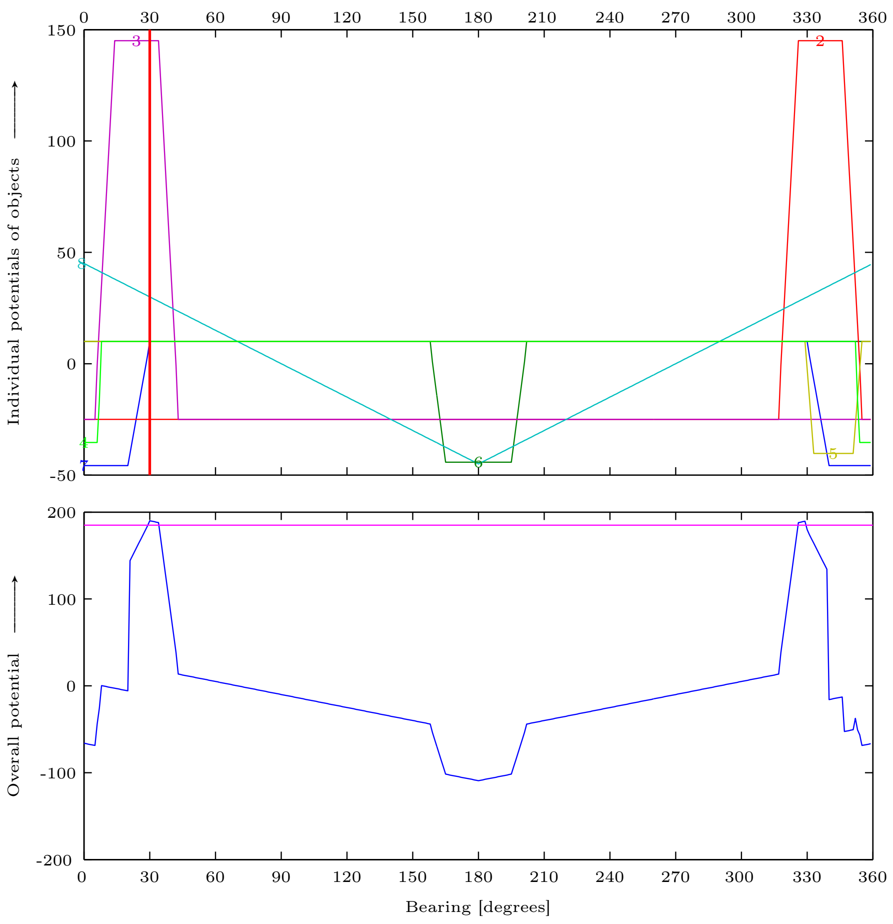


Figure A.4: *Potentials in Situation 2.*

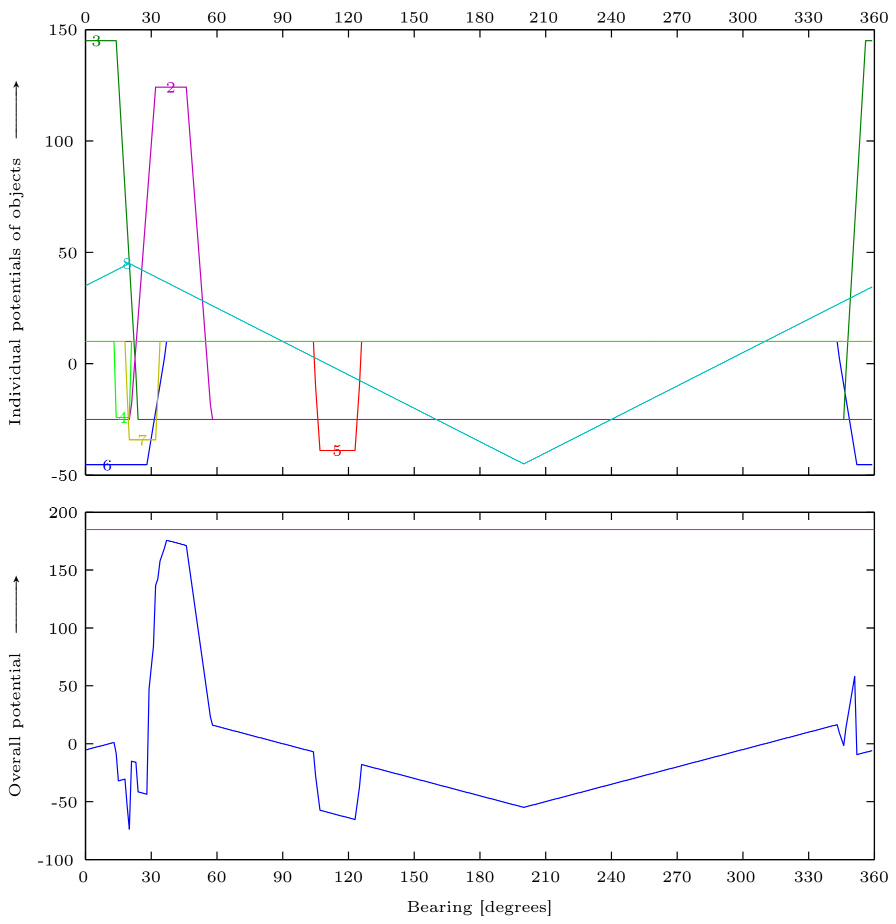


Figure A.5: *Potentials in Situation 3.*

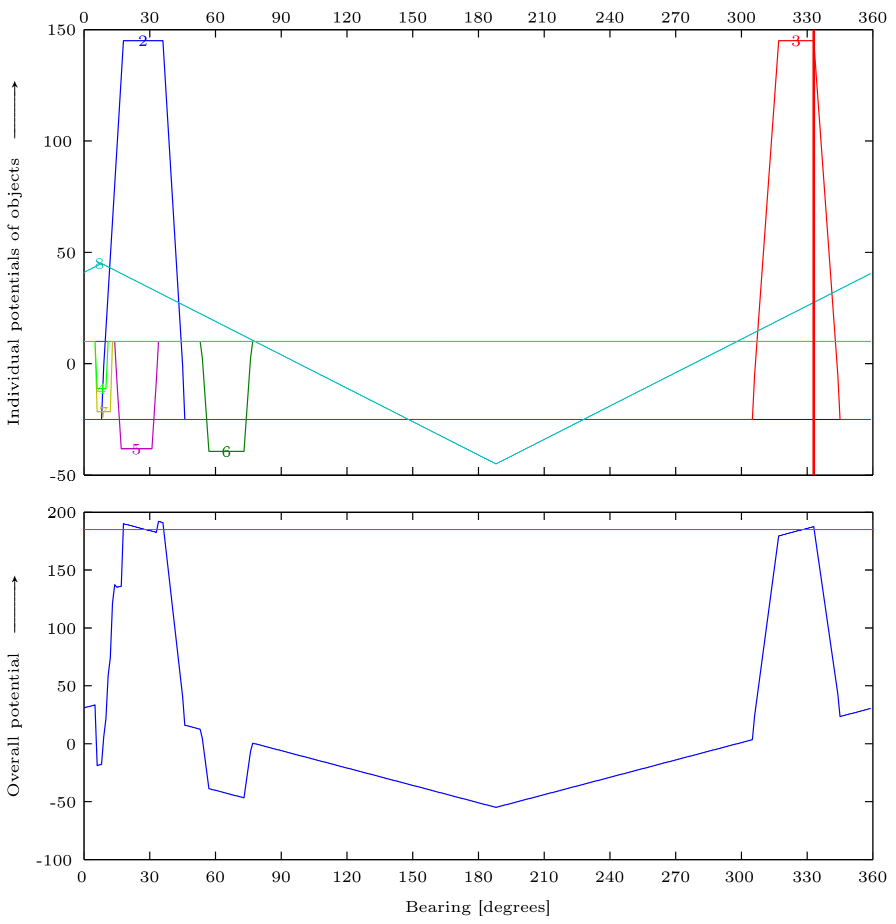


Figure A.6: *Potentials in Situation 4.*

APPENDIX B

Source code & parameters

After presenting our actual parameter set–up, we will comment on the source code of the main passing.

B.1 Parameters

The following parameters determine the passing behaviour by influencing the potential field on which the passing decision is based. See Chapter 3 for details.

Symbol	Value	Name / short description
τ_0	185	Nominal threshold for passing trigger
d_A	30	Team mate core potential shape parameter
d_M	60	Team mate core potential shape parameter
d_N	150	Team mate core potential shape parameter
d_B	180	Team mate core potential shape parameter
$h_{\mathrm{T},0}$	145	Maximum core potential from team mate
$h_{\mathrm{O},0}$	-50	Maximum core potential from opponent
$h_{\mathrm{GA},0}$	45	Maximum core potential from goal attractor
$r_{\mathrm{T},0}$	20	Core radius (in [cm]) of team mate
$r_{\mathrm{O},0}$	20	Core radius (in [cm]) of opponent
$r_{\mathrm{GA},0}$	0	Core radius (in [cm]) of goal attractor
$s_{\mathrm{T},0}$	-2000	Slope coefficient of team mate potential function
$s_{\mathrm{O},0}$	0.1	Slope coefficient of opponent potential function
$s_{\mathrm{GA},0}$	-0.5	Slope coefficient of goal attractor potential function
$\gamma_{\mathrm{T},0}$	-25	Cut–off for team mate potential function
$\gamma_{\mathrm{O},0}$	10	Cut–off for opponent potential function
$\gamma_{\mathrm{GA},0}$	-200	Cut–off for goal attractor potential function

Table B.1: *Parameters from the third chapter.*

B.2 Source code

Going through the source code step by step, we shall now see how we implemented the concepts and equations from the third chapter in one large algorithm. Currently this is done in python™ as with the large number of tunable parameters it is important to have a short development / update cycle, which would not be the case with C++ (as described in the first chapter).[1] However, it is quite likely that the code will be ported to C++ at some later stage to save processing power (for an improved and refined robot detection algorithm for instance).

We recommend [2] as not only an excellent reference but also as a very condensed tutorial for the python™ programming language. A more extensive introduction would be [1].

B.2.1 Initialisation

As with most algorithms, a few things need to be set–up and initialised:

```
# -*- coding: latin-1 -*-
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
          ___       _ _   _       _ _           _   _
         |_ _|_ __ (_) |_(_) __ _| (_)___  __ _| |_(_) ___  _ __
          | || '_ \| | __| |/ _` | | / __|/ _` | __| |/ _ \| '_ \
          | || | | | | |_| | (_| | | \__ \ (_| | |_| | (_) | | | |
         |___|_| |_|_|\__|_|\__,_|_|_|___/\__,_|\__|_|\___/|_| |_|


"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
from math import pi,sqrt,atan,atan2,fabs
from operator import mod

offline = False # for debugging

# passing threshold
thresh = 185

# team mates
h_T0 =   145
r_T0 =    20
s_T0 = -2000
g_T0 =   -25
d_A , d_M , d_N , d_B = 30, 60, 200, 230

# opponents
h_O0 =   -50
r_O0 =    20
s_O0 =     0.1
g_O0 =    10

# goal attractor
h_GA0 =   45
r_GA0 =    0
s_GA0 =   -0.5
g_GA  = -200
ga_coord = [210 , 0]

# misc.
r2d = 180/pi; d2r = pi/180;

def l_mod(the_list,the_mod): # modulus function for lists
  out = []
  for i in the_list:
    out = out + [i % the_mod]
  return out

def modpm180( ang ): # mod angles to -180...180
  out = mod(ang+360,360)
  if out > 180:
    out = out - 360
  return out
```

[1] As mentioned earlier, we implemented an (in the results) identical version of the algorithm in Matlab for its development but also the generation of the graphics.

For our convenience, we define a function `l_mod()` which is simply an extension of the normal modulus function to lists. Also, we create `modpm180()`, a function which normalises angles into the interval $]-180, 180]$.

B.2.2 Object set–up

This is where `check_for_pass()` itself starts, which will eventually return pass quality, direction, receiver id and the angle how much the pass will go off the receiver's bearing (for through passes).

```python
def check_for_pass(the_team,the_opp):
  # RETURNS: QUALITY , BEARING (in rad!) , RECEIVER , DELTA_BEARING (in rad!)

  myx , myy = the_team[0][0] , the_team[0][1]

  if (myx > 205 and fabs(myy)-65 < 0): # in opp's pen. box -> shoot! not pass
      print "Nooooooooooooooooooooooooooooooooooooooooo pass (bc in pbox)!"
      return [0.0,0.0,-1,0.0]



  """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
         ____                _          _     _           _
        / ___|_ __ ___  __ _| |_ ___   ___ | |__ (_) ___  ___| |_ ___
       | |   | '__/ _ \/ _` | __/ _ \  / _ \| '_ \| |/ _ \/ __| __/ __|
       | |___| | |  __/ (_| | ||  __/ | (_) | |_) | |  __/ (__| |_\__ \
        \____|_|  \___|\__,_|\__\___|  \___/|_.__// |\___|\___|\__|___/
                                                 |_/

  """"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
  # Each object contains the following information
  #
  #    0        1       2        3         4        5       6       7
  # center | height | radius | slope   | cut-off | dist |  kind   |  id
  #  deg       ua      deg     ua/deg     ua       cm     1,2,3    int
  #
  #  where kind: 1 = team, 2 = opp, 3 = other | id = 1,...,7
  n = len(the_team)-1
  team_d = [0.0]*n;   team_th = [0.0]*n;   objects = [];   i = 0;

  # 1. team mates
  for team_mate in the_team[1:n+1]:

    d = sqrt((myx-team_mate[0])**2+(myy-team_mate[1])**2)
    if d == 0:
      print "Team mate reported with d=0! Something's wrong!"
      return [0.0,0.0,-2,0.0]

    th = int(mod(atan2(team_mate[1]-myy,team_mate[0]-myx)*r2d+360,360)+.5)

    if   d ≤ d_A:   h_t = 0
    elif d ≤ d_M:   h_t = (d-d_A)/(d_M-d_A)*h_T0
    elif d ≤ d_N:   h_t = h_T0
    elif d ≤ d_B:   h_t = (d_B-d)/(d_B-d_N)*h_T0
    else:           h_t = 0

    objects =                   objects   +                    \
      [[th, h_t, int(atan(r_T0/d)*r2d+.5), s_T0/d, g_T0, d, 1, i]]

    team_d[i] = d; team_th[i] = th; # store d and th for later

    i += 1

  # 2. opponent players
  for opponent in the_opp:

    d = sqrt((myx-opponent[0])**2+(myy-opponent[1])**2)
    if d == 0:
      print "Opponent reported with d=0! Something's wrong!"
      return [0.0,0.0,-2,0.0]

    th = int(mod(atan2(opponent[1]-myy,opponent[0]-myx)*r2d+360,360)+.5)

    h_o = -h_O0*d/650+h_O0

    objects =                    objects   +                     \
      [[th, h_o, int(atan(r_O0/d)*r2d+.5), d*s_O0, g_O0, d, 2, i]]

    i += 1

  # 3. goal attractor
  d = sqrt((myx-ga_coord[0])**2+(myy-ga_coord[1])**2)
  th = int(mod(atan2(ga_coord[1]-myy,ga_coord[0]-myx)*r2d+360,360)+.5)

  if d == 0:
    print "Goal attractor reported with d=0! VERY unlikely, that!!"
    return [0.0,0.0,-2,0.0]

  objects =                    objects   +    \
    [[th , h_GA0 , r_GA0 , s_GA0 , g_GA , d , 3 , i]]

  if d < 120: # increase threshold as player comes closer to goal
    the_thresh = (120-d)/120 * thresh/2  +  thresh
  else:
    the_thresh = thresh

  ga_bear = th # store goal attractor bearing for later


  # 4. sort objects by distance
  objects.sort(lambda x,y: cmp(float(x[5]), float(y[5])) )
```

At the beginning, we check if the player is in the offensive penalty box. If he is in there, he should never consider passing (too risky), but rather take a shot at the goal himself.

However, if he is not in that area, the actual algorithm is launched. First we calculate the relative bearing and distance for each object class, that is team mates, opponents and goal attractor (as their positions are given in absolute coordinates through the global Object Array and passed over in that form).

Then, the an entry for that object is stored in the local `objects`–array. Finally, the objects are sorted according to their distance, from closest to farthest (in preparation of our clipping solution later on).

We also set the actual threshold value $\tau(d_{\mathrm{GA}})$ for triggering a pass, now that we have calculated the distance to the goal attractor.

B.2.3 Calculation of the potential field

The next big step is to iterate through the objects to determine and add their influence, degree by degree, to the potential field.

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

       ____      _                       _         __ _      _     _
      / ___|__ _| | ___      _ __   ___ | |_      / _(_) ___| | __| |
     | |   / _` | |/ __|    | '_ \ / _ \| __|    | |_| |/ _ \ |/ _` |
     | |__| (_| | | (__ _   | |_) | (_) | |_ _   |  _| |  __/ | (_| |
      \____\__,_|_|\___(_)  | .__/ \___/ \__(_)  |_| |_|\___|_|\__,_|
                            |_|

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
posblock = []; negblock = []; att = [0 for i in xrange(360)];

for obj in objects:
  co_c = obj[0];  co_h = obj[1]; co_r = obj[2];
  co_s = obj[3]; co_co = obj[4]; co_k = obj[6];

  # ignore zero height objects
  if co_h == 0: continue
```

```

    # create potential, respecting clipping effects
    lim = round((co_co-co_h)/co_s+co_r)

    if co_k == 1: # team mates, so we want to clip negative attractors

      for deg in xrange(181):

        if deg ≤ co_r: # within flat part
          half = co_h
        elif deg ≥ lim: # cut-off part
          half = co_co
        else: # sloped part
          half = co_h+co_s*(deg-co_r)

        # shift value by object_center degrees
        deg_shifted = int(mod(360+deg+co_c,360))
        if deg_shifted not in negblock or deg ≥ lim:
          att[deg_shifted] += half

        # mirror value from 360 back
        if deg ≥ 1 and deg ≤ 179:
          deg_shifted = int(mod(360-(deg-co_c),360))
          if (deg_shifted not in negblock) or (deg ≥ lim):
            att[deg_shifted] += half

      posblock = posblock + l_mod(xrange(co_c-co_r+360,co_c+co_r+360+1),360)

    if co_k == 2: # opponents, so we want to clip positive attractors
      for deg in xrange(181):

        if deg ≤ co_r: # core
          half = co_h
        elif deg ≥ lim: # cut-off part
          half = co_co
        else: # sloped part
          half = co_h+co_s*(deg-co_r)

        # shift value by object_center degrees
        deg_shifted = int(mod(360+deg+co_c,360))
        if deg_shifted not in posblock or deg ≥ lim:
          att[deg_shifted] += half

        # mirror value from 360 back
        if deg ≥ 1 and deg ≤ 179:
          deg_shifted = int(mod(360-(deg-co_c),360))
          if (deg_shifted not in posblock) or (deg ≥ lim):
            att[deg_shifted] += half

      negblock = negblock + l_mod(xrange(co_c-co_r+360,co_c+co_r+360+1),360)

    if co_k == 3: # other objects, no clipping
      for deg in xrange(181):

        if deg ≤ co_r: # core
          half = co_h
        elif deg ≥ lim: # cut-off part
          half = co_co
        else: # sloped part
          half = co_h+co_s*(deg-co_r)

        # shift value by object_center degrees
        deg_shifted = int(mod(360+deg+co_c,360))
        att[deg_shifted] += half

        # mirror value from 360 back
        if deg ≥ 1 and deg ≤ 179:
          deg_shifted = int(mod(360-(deg-co_c),360))
          att[deg_shifted] += half
  # end for obj in objects


  if offline: # store results to csv file for debugging
    write = ''; i = 0;
  for t_att in att:
    write = write+`i`+','+`t_att`+'\n'
    i += 1
  filename = "E:\Studium\d_diplomarbeit\python\data.csv";
  infile = open(filename,"w");    infile.write(write);    infile.close();
```

The main idea here is to use a loop going from 0 to only 180 degrees (which is sufficient due to the symmetry of our potential function) to generate the potential in each point according to (3.2) on page 39. At the same time we shift the potential by c degrees and "mirror" the value to gain the full 360 degrees influence.

At the same time we take care of the clipping with the help of two lists — a positive and a negative block list. After an object has added its 360 degree wide influence to the potential field, the angles that make up its core (that is all the angles in $[c-r, c+r]$) are appended to the respective block list.

The positive block list is then used to block future negative influences at each of its member bearings (as future objects are necessarily farther away due to our sorting by distance at the end of the previous code snippet); the negative block list in turn takes care of clipped positive influences. However, only the sloped and core parts of some object's potential function can be blocked. We do not want to clip the influence from the cut–off part of the potential functions as we want to retain the "fine tuning" effect they create around the field.

B.2.4 Final passing decision

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
       __  __       _                _           _     _
      |  \/  | __ _| | _____    __| | ___  ___(_)___(_) ___  _ __
      | |\/| |/ _` | |/ / _ \  / _` |/ _ \/ __| / __| |/ _ \| '_ \
      | |  | | (_| |   <  __/ | (_| |  __/ (__| \__ \ | (_) | | | |
      |_|  |_|\__,_|_|\_\___|  \__,_|\___|\___|_|___/_|\___/|_| |_|


"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
# calculate ROI
if myx < -135: # last quarter of the field -> ROI = f(myy)
  if myy ≥ 0:
    throi = 90-.5*myy
  else:
    throi = -90-.5*myy
  throi = mod(int(throi+.5),360)

else: # ROI = f(GA_bearing)
  throi = ga_bear

if myx > 0: # sender in offensive half - scale ROI with with myx
  dthroi = int( (1+myx/270)*36+.5 )
else: # defensive half - ROI constant width
  dthroi = 36

roi = l_mod(  xrange( (throi-dthroi)+360 , (throi+dthroi)+361 )  ,  360  )

# find maximum inside ROI
max_att = 0.0; i = 0;
for i in roi:
  if (att[i] ≥ the_thresh) and (att[i] ≥ max_att):
    max_att = att[i]
    pass_ang = i

# calculate pass receiver
if max_att > 0: # we found a winner

  min_d = 1000.0; receiver = -1;
  pass_range = l_mod(range(pass_ang-35,pass_ang+36),360)
```

```

    for i in xrange(n):
      if ( team_th[i] in pass_range ) and ( team_d[i] < min_d):
        receiver = i

    # calculate how far the ball is to go "ahead" of the receiver, if not bang on
    delta_ang = modpm180(pass_ang-team_th[receiver])
    d20 = fabs(delta_ang)-20 # pass more than 20 deg off the receiver -> crop it down to 20
    if d20 > 0: # delta > +-20 deg
      if delta_ang > 0:
        delta_ang = 20
        pass_ang -= d20
      else:
        delta_ang = -20
        pass_ang += d20

  # output and return
    print "PASS: %i deg | Q: %3.3f | ID: %i | D: %i deg" \
                          % (pass_ang,max_att-the_thresh,receiver,delta_ang);

    return [max_att-the_thresh,pass_ang*d2r,receiver+1,delta_ang*d2r]

  else:

    print "Noooooooooooooooooooooooooooooooooooooooooooooooooooooooo pass !"
    return [0.0,0.0,-1,0.0]



if offline: # for debugging
  team =  [ [209.3,-91.0] ,[162.2,43.2] ]
  opp  = [[270,0],[119,102],[-1,-90],[175,-10]]

  check_for_pass(team,opp)
```

In this last stage of our algorithm, we first calculate the region of interest as described on page 34. We then look for a “winning” bearing inside it by searching the maximum potential but ignoring angles that do not exceed the threshold τ set above.

If we have found a “winner”, we need to determine what robot is actually meant to be the receiver.[2] This is done taking two things into consideration: On the one hand, with a very high probability the receiver will be at most 35° away from the passing bearing.[3] On the other hand, if there are more than one team mate in this region, it is usually the closer one that is supposed to get the ball. So, for our algorithm, we assume that the receiver is a) the closest team mate that b) is within ±35° of the passing bearing. Extensive simulations have shown that this method gives great and very reliable results.

Finally, we either return the pass bearing together with our measure of quality (again, the difference between the bearings potential and the threshold), the receiver id and the amount of degrees the passing bearing lays off the actual receiver bearing, or we return a negative quality as message that no pass is being suggested.

[2] Interestingly enough, this information does not directly follow from the maximum in the potential field when the core radius of team mates $r_{T,0}$ is non–zero.

[3] This corresponds roughly to a team mate core radius of 20 cm at a distance of d_M = 60 cm, the closest receiver distance that could trigger a pass.

List of Symbols

General notations

n	Scalars; lowercase letters
$\boldsymbol{x}$	Vectors; lowercase bold letters (always supposed to be column vectors, if not transposed), or elementwise in parentheses: $\begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix}^T$
$\boldsymbol{M}$	Matrices; uppercase bold letters, or elementwise in brackets: $\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$
$\mathcal{S}$	Sets; uppercase "calligraphic" letters
P	Points; uppercase "upright" letters, with coordinates $\mathrm{P} = (x_\mathrm{P}, y_\mathrm{P})$
$\mathfrak{P}$	Properties, assumptions; uppercase "Fraktur" letters
$\bar{x}$	The average value of x
$\hat{x}$	An estimated value of x
x^-	An *a priori* value of x
x_k	The value of variable x at frame k, sometimes also denoted $x(k)$
$\mathbf{id}$	Identity matrix of suitable dimensions
$\mathbf{0}$	Zero matrix of suitable dimensions

Bibliography

We tried to provide as many publicly accessible web links as possible.

[1] David Ascher and Mark Lutz. *Learning Python*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2nd edition, December 2003.
http://books.google.com.au/books?vid=ISBN0596002815

[2] David M. Beazley. *Python Essential Reference*. New Riders Publishing, Indianapolis, IN, USA, 2nd edition, June 2001.
http://books.google.com/books?vid=ISBN0735710910

[3] Stephan Chalup. Newcastle robotics laboratory homepage. Website, February 2006. http://robots.newcastle.edu.au/

[4] RoboCup Four Legged League Technical Committee. Teams 2006. Webpage, January 2006. http://www.tzi.de/4legged/bin/view/Website/Teams2006

[5] RoboCup Technical Committee. RoboCup four–legged league rule book, April 2006. http://www.tzi.de/4legged/pub/Website/Downloads/Rules2006.pdf

[6] Sony Corporation. ERS–7M3 user's guide. User manual, 2005.
http://esupport.sony.com/US/perl/model-documents.pl?mdl=ERS7M3

[7] Sony Corporation. Sony AIBO Europe — Official Website. Website, 2006.
http://support.sony-europe.com/aibo/

[8] Python Software Foundation. Python programming language — official website, 2006. http://www.python.org

[9] Kenny Hong. Enhancements to vision processing, and debugging software for robocup soccer. Bachelor Thesis, November 2005.
http://www.cs.newcastle.edu.au/~c2103674/FinalReportWebVersion.pdf

[10] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, UK, 1985.
http://books.google.com.au/books?&id=9wTacOjHE6IC

[11] Rudolph E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82:34–45, March 1960.
http://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf

[12] Oussama Khatib. Real–time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5:90–98, April 1986. http://www.cs.unc.edu/~welch/kalman/media/pdf/Kalman1960.pdf

[13] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, USA, December 1990.
http://books.google.com.au/books?id=Mbo_p4-46-cC

[14] Tim Laue and Thomas Röfer. A behavior architecture for autonomous mobile robots based on potential fields. 8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences), July 2005. http://www.informatik.uni-bremen.de/kogrob/papers/rc05-potentialfields.pdf

[15] Witold Pedrycz and Fernando Gomide. *An introduction to fuzzy sets: analysis and design*. The MIT Press, Cambridge, MA, USA, May 1998.
http://books.google.com.au/books?vid=ISBN0262161710

[16] Michael J. Quinlan. *Machine Learning on AIBO Robots*. PhD thesis, The University of Newcastle, Callaghan, NSW, Australia, March 2006.
http://robots.newcastle.edu.au/quinlan.html

[17] Michael J. Quinlan, Steven P. Nicklin, Kenny Hong, Naomi Henderson, Stephen R. Young, Timothy G. Moore, Robin Fisher, Phavanna Douangboupha, Stephan K. Chalup, Richard H. Middleton, and Robert King. The 2005 NUbots team report, February 2006.
http://robots.newcastle.edu.au/publications/NUbotFinalReport2005.pdf

[18] Greg Welch and Gary Bishop. An introduction to the kalman filter, April 2004. http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf

[19] Lotfi A. Zadeh. Fuzzy sets. *Journal of Information and Control*, 8:338–353, June 1965. http://dx.doi.org/10.1016/S0019-9958(65)90241-X

[20] Lotfi A. Zadeh. Fuzzy logic and approximate reasoning. *Synthese*, 30:407–428, September 1975. http://www.springerlink.com/link.asp?id=j747055567237033