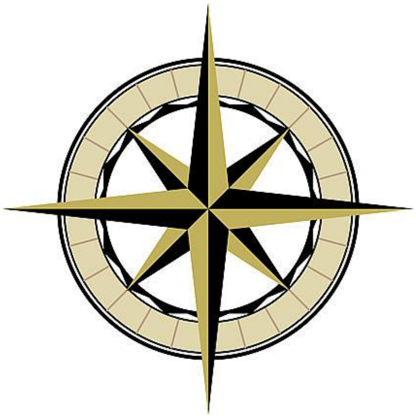
An appearance-based visual compass for mobile robots

Jürgen Sturm

Master's Thesis
for the graduation in Artificial Intelligence



Supervised by
Drs. Arnoud Visser
Prof. Dr. Ir. F.C.A. Groen
Intelligent Autonomous Systems group
Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Universiteit van Amsterdam

December 18, 2006

Abstract

Localization is one of the basic skills of a mobile robot. Much progress has been made in this field over the past years. In particular, the RoboCup competitions have focused their research within robotics on a unified challenge problem, and the yearly rule adaptations aim at increasingly approximating robotic soccer to the real world.

Until now, most approaches, however, still rely on special sensors (like laser range scanners, preferably used in the RoboCup rescue leagues) or artificial environments (like color-tagged soccer fields, as used by the RoboCup soccer leagues).

In this thesis, a novel approach is presented that can provide compass information purely based on the visual appearance of a room. A robot using such a visual compass can quickly learn the appearance of previously unknown environments. By the level of resemblance, a robot can additionally recognize qualitatively how far it is away from the former training spot. This can be used for visual homing.

The visual compass algorithm is efficient, scaleable and can therefore work in real-time on almost any contemporary robotic platform. Therefore, the approach has been implemented on the popular Sony entertainment robot Aibo.

Extensive experiments have validated that the approach works in a vast variety of environments. It has been shown that a visual compass can supply a mobile robot in natural environments with accurate heading estimates. Finally, it is shown in experiments that a robot using multiple compasses is able to estimate its translational pose.

Acknowledgments

Many thanks go to Arnoud Visser, both for during the stage of developing ideas and software as well as during the writing of this thesis. Writing one's Master's thesis on the topic of soccer robots is not possible without the support of the team behind the team: I would like to thank all the people who were members of the Dutch Aibo Team, in particular Stefan Leijnen and Silvain van Weers from Utrecht University who helped me with my first steps in the Dutch soccer code. Also I want to thank Paul van Rossum from the Technical University of Delft who helped me to conduct some experiments on the Delft Midsize field.

Additional thanks go to Hannah Whitney-Steele, Julia Frankenberger, Marit van Dijk and Alexander Branschädel for proof-reading and correcting my thesis.

I am grateful to my parents for their support and encouragement throughout the years.

Contents

1 Introduction **1**
- 1.1 Intelligent autonomous systems 1
- 1.2 RoboCup 2
- 1.3 Problem domain and related work 3
  - 1.3.1 State of the art in the 4-Legged League 4
  - 1.3.2 Other approaches 8
- 1.4 Thesis goal and research question 10
- 1.5 Outline 10

2 Probabilistic robotics **13**
- 2.1 Motion 13
- 2.2 Vision 15
- 2.3 Localization 17
  - 2.3.1 Belief distributions 18
  - 2.3.2 Bayesian filters 19
  - 2.3.3 Solution techniques 20
- 2.4 Summary 23

3 Approach **25**
- 3.1 Compass sensor 25
  - 3.1.1 Heading 25
  - 3.1.2 Single transition 26
  - 3.1.3 Transition patterns 27
  - 3.1.4 Sampling transition patterns from camera images 28
  - 3.1.5 Refining the resolution 28
  - 3.1.6 Map learning 29
- 3.2 Localization filter 30
  - 3.2.1 Motion model 30
  - 3.2.2 Sensor model 31
  - 3.2.3 Orientational belief filter 31
  - 3.2.4 Four-spot sensor model 32
  - 3.2.5 Translational belief filter 32
- 3.3 Summary 33

4 Implementation **35**
- 4.1 Compass sensor 35
  - 4.1.1 Color classes 35
  - 4.1.2 Sampling transition patterns from camera images 36

4.1.3 Learning . . . 37
4.2 Localization filter . . . 37
4.2.1 Orientational filter . . . 38
4.2.2 Translational filter . . . 38
4.3 Behaviors . . . 38
4.4 Discussion . . . 39
4.5 Summary . . . 40

5 Results **41**
5.1 Experimental setup . . . 41
5.2 Orientational filter: qualitative results . . . 42
5.2.1 Home environment: Living room . . . 42
5.2.2 Office environment: DECIS Lab . . . 42
5.2.3 Outdoor soccer environment . . . 43
5.2.4 Indoor Midsize field . . . 44
5.2.5 Indoor 4-Legged field . . . 44
5.2.6 Demonstrations . . . 45
5.3 Orientational filter: quantitative results w.r.t. variables . . . 45
5.3.1 Variance w.r.t. distance . . . 46
5.3.2 Variance w.r.t. changing environmental conditions . . . 46
5.4 Orientational filter: quantitative results w.r.t. parameters . . . 46
5.4.1 Null hypothesis . . . 47
5.4.2 Variance w.r.t. number of colors . . . 47
5.4.3 Variance w.r.t. scanning resolution . . . 49
5.4.4 Variance w.r.t. learning time . . . 49
5.5 Orientational filter: computation time . . . 49
5.6 Translational filter . . . 51
5.6.1 Qualitative results . . . 51
5.6.2 Quantitative results . . . 52
5.7 Summary . . . 53

6 Conclusions **55**
6.1 The answer to the research question . . . 55
6.2 Applicability of the concept . . . 57
6.3 Discussion . . . 57
6.4 Future research . . . 58

A The Sony Aibo as a platform for robotics research **65**
A.1 The Sony Aibo robot . . . 65
A.2 The software framework . . . 66
A.2.1 Robot software . . . 67
A.2.2 Simulator . . . 69
A.2.3 Remote debugging . . . 69
A.2.4 Debugging by playback . . . 70
A.3 Body odometry . . . 70
A.4 Head odometry . . . 71
A.5 Camera Quality . . . 73
A.6 Processing power . . . 75
A.7 Availability of Aibos . . . 76

A.8 Experience at the UvA and in the Dutch Aibo Team . . . 77
A.9 RoboCup competitions . . . 77
A.10 Summary . . . 77

B Camera calibration and preparation **79**
B.1 Auto-Shutter . . . 79
B.2 Color clustering . . . 80
B.3 Summary . . . 82

C Direct pose triangulation **83**
C.1 Approach . . . 83
C.2 Map learning . . . 85
C.3 Sensor model . . . 85
C.4 Discussion . . . 86
C.5 Summary . . . 86

D RoboCup: UvA achievements **87**

Chapter 1

Introduction

In the course of this Master's thesis, an Aibo robot is taught the skill to orientate itself in natural environments. The robot memorizes the appearance of a place in little time by looking around and turning itself on the spot. The robot learns the appearance of its surroundings that it can use from now on as a compass. Moreover, the robot can recognize by the level of resemblance qualitatively how far it is away from the training spot.

1.1 Intelligent autonomous systems

Over the past decades, autonomous mobile robots have become an important research domain within Artificial Intelligence [24]. The reasons for this are twofold. From an economic perspective, stationary robots became an important factor [1] in Western production facilities – which put general emphasis on the field of robotics. From a research perspective, a shift in paradigm boosted progress in mobile robotics in the early 1990s enormously: the physical-grounding hypothesis [4] led to a change in focus from symbolic systems to **situated agents**. Instead of assuming the existence of a deterministic world evaluated only by a single omniscient program, adherents to situated AI [19] oppositely promoted dynamic, uncertain and imperfect real-world environments where multiple agents sense, think and (inter-)act in parallel.

Rodney Brooks formulated this insight as follows:

> "The key observation is that the world is its own best model. It is always exactly up to date. It always contains every detail there is to be known. The trick is to sense it appropriately and often enough." (Elephants Don't Play Chess, R. Brooks, 1990)

Agents constructed using this paradigm are therefore intrinsically interdependent with their environment. Such situated agents most naturally are embodied in a real robot. If the agent has full control over its body, it is said to be **autonomous**. It can then pursue its own agenda in interaction with the environment. Experiencing and acting in the real world is facilitated if the robot has the ability to move. Other actors of the real world can then also be modelled as agents, leading to the definition of **multi-agent systems**. As the real world is dynamically changing, it is commonly asserted that an agent needs the **ability to adapt** – which is in general considered as the **basis for intelligence**.

In the late 1980s, the group around Brooks constructed several robots to show that situated agents can exhibit interesting behaviors emerging from simple rules (the subsumption architecture [5]). For example, a six-legged spider robot [3] could climb over rough terrain controlled by a few simple behaviors. In another experiment, social behavior was induced: two small autonomous race cars were constructed so that they could drive around collision-free or follow moving objects when they detected something in front of them. In the end, the cars chased each other and thereby showed emergent social behavior.

In the history of autonomous robots, much more has happened in the last decade. In the early 1990s, Honda stunned the public with the first biped robot that could walk in a realistic manner (the ancestor of the later Asimo robot). In 1997, a four-legged robot called Dappie was developed by Peter van Lith (see appendix D). Two years later, Sony released the first model of the Aibo entertainment robot. The Aibos were not only liked by children and technonauts, but they were also rapidly accepted as an inexpensive and well-equipped robotic platform within the scientific robotics community.

1.2 RoboCup

Inspired by the situated-agent paradigm, several AI researchers (A. Mackworth et al [25]; M. Asada et al [12]; M. Valoso, P. Stone et al [29]) proposed, independently from each other, that robotic soccer could serve as an interesting challenge for in field of Artificial Intelligence. In 1993, the Robot World Cup initiative was founded (RoboCup for short) with the following goal:

> “By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team.” (The RoboCup Challenge)

Solving the RobCup challenge requires inter-disciplinary collaboration between numerous research domains. Issues from control theory, machine learning, computer vision, sensor data fusion, localization and multi-player collaboration are addressed in the challenge.

Various leagues have been established to pinpoint preliminary research subgoals. By the organization in leagues, participants can join a league with a similar research focus:

- In the **simulation league** software agents play in a highly abstracted soccer environment. Although all agents run on the same computer, they may only communicate a limited amount of data among each other. Therefore, the league focuses in particular on multi-player strategic collaboration and coordination problems.
- Games of the **small size league** are played by relatively small robots ($\o = 15cm$). They are controlled by a central computer that analyzes the game situation on basis of images from an overhead camera. The resulting games show pleasant swarm behavior and interesting team play.
- The **midsize league** uses relatively big but fully autonomous robots ($\o = 50cm$), typically equipped with omni-directional cameras and omni-directional drives. Current issues are locomotion, vision and team play – with the advantage of having almost no hardware restrictions.
- The **4-Legged league** obliges the use of Sony Aibo robots. The scientific issues are quadruped walking and active vision (with the turnable head camera). This league is discussed in detail in the next subsection.
- The **humanoid league** tackles in particular problems concerning the two legs. Biped walking and running are still problematic, one-legged kicking and actions like falling primarily put the current focus of research on mechanical issues.

Non-soccer leagues have later been added to deal with important problems of mobile robotics that are not covered by the traditional soccer domain:

- The **rescue leagues** focus on urban search and rescue problems for the use in catastrophe environments. These are either real (focusing on the development on robust drives, able to traverse through rough terrains) or simulated (focusing on de-centralized and robust multi-agent exploration).

Figure 1.1: Pictures taken at the RoboCup competitions. **Left:** 4-Legged league, Aibo ERS210 and ERS7, a few seconds after kickoff. **Right:** Honda's Asimo, demonstrating its abilities by shooting penalties against a human goalkeeper.

- The **at-home league** addresses in particular seemingly simple everyday problems of life. Challenges range from walking stairs, following a human in a crowded place to localizing and navigating in natural environments.

In 1997, the first RoboCup competitions were held with 37 participating teams. The success was overwhelming: the number of participating teams continuously doubled in the first years [2]. In 2003, the (national) Dutch Aibo Team was founded[1] with the aim to boost research on collaborative autonomous and intelligent systems in the Netherlands. In 2006, already 440 teams from more than 35 countries participated in the RoboCup competitions. The scientific output produced by RoboCup participants is significant: In the meantime, every 10th article[2] on mobile robots is directly related to RoboCup.

After each world cup, the competition rules are adjusted to reflect the progress made in the field and to keep the problems challenging: For example, since 2004 the Four-legged league plays on a bigger field without a boundary, while the games of the Midsize league took place in 2006 in open daylight.

Various teams from the Universiteit van Amsterdam have since 2001 participated in different leagues of the RoboCup competitions (see Appendix, table D.1). The fact that several prices were won at the world championships validate the scientific competitiveness of the Intelligent Autonomous Systems group at a world-wide scale.

1.3 Problem domain and related work

From the previous section, it can be concluded that robotic soccer became a widely accepted challenge problem in the field of Artificial Intelligence. Therefore it is prudent to contribute to these efforts in terms of a graduation project.

The author has chosen the 4-Legged league as the problem domain for two reasons:

- the **focus on vision** (i.e. real-time image processing), and
- its **application in actual robots** (the Sony Aibos)

In the following, the state-of-the-art of the 4-Legged league will be summarized based on the latest technical reports of successful teams, and from their current algorithms the resulting strong and weak points will be pointed out. Then, alternative approaches from outside the RoboCup domain will be carefully examined to collect algorithmic inspirations for possible improvements.

---

[1] Universiteit van Amsterdam, Universiteit Utrecht, Technische Universiteit Delft, Rijksuniversiteit Groningen, Technische Universiteit Eindhoven, Universiteit Twente, Saxion Universities at Enschede, DECIS Lab

[2] For the year 2005, Google Scholar counted 522 scientific articles containing the word "RoboCup" compared to 5190 articles in the same period of time containing the words "mobile robot".

1.3.1 State of the art in the 4-Legged League

The rules of the 4-Legged league[3] specify that only Sony Aibo Entertainment robots are to be used. Outsourcing hardware development and maintenance helps keeping costs and participation efforts minimal. Unfortunately, Sony decided in spring 2006 to discontinue their robotics branch – forcing the 4-legged league to seek a replacement.

A Sony Aibo stands on four legs, and has a low-resolution camera[4] built in the tip of its nose. Thanks to more than 20 degrees of freedom, the robot is able to perform relatively agile movements; in particular, it can turn its head in three dimensions by which it can perceive more than half of its surroundings when looking around. As all teams have to use robots of the official Aibo series, the solutions for this only differ in algorithms.

The league handles an open-source and documentation policy with the aim to share improvements between the teams and to facilitate that new teams can participate with low entry costs. For example the soccer code of the Dutch Aibo Team [49] and others[5] has been branched from the excellent software basis supplied by the German Team[41].

In the following, selected approaches from successful teams are decomposed into four fundamental modules [28]. Most emphasis is herein put on the vision and localization, as this forms the context for the approach described in this thesis.

Motion

Body motion of Aibos playing soccer basically refers to walking (with 4 legs) and kicking (with the whole body). As the Aibo is able to sense its body pose by sensors in its joints, it is possible to record a series of body configurations that the Aibo can store and replay later on. Using this technique, many kicks (and some of the walks) of the German Team [41] have been created.

However, walking this way is neither very fast nor flexible. Therefore, most teams apply a parametrized walk [11] that allows the Aibo to walk omnidirectional. The idea is to treat each of the Aibo's legs like a independent, steerable wheel. Often too many parameters involved to optimize the gait manually, and therefore several teams have successfully applied genetic algorithms to continuously increase the maximum speed of their Aibo robots; last year, forward speeds greater than 40cm/s have been reached [32].

Next to speed, precision is also an issue. Mechanical differences between robots complicate parameter optimization, and even with perfect parameters, walking on four legs stays prone to long-term drifts (i.e. due to slippage).

Vision

Aibos perceive the world mainly through their internal camera; therefore perception in the 4-Legged league primarily comes down to image analysis. Relevant visible objects (while playing soccer) in the images are: the two goals (in blue and yellow), an orange ball, white field lines on a green carpet, and four flags. Some teams also recognize other robots (dressed in blue and red). Perception typically outputs the world coordinates of the detected objects in a coordinate system relative to the robot. Most teams use similar processing steps between the raw image and the percepts:

Image segmentation The color space of the raw images is usually discretized into a small number of color classes that are present on the field (like white, green, yellow and blue). Some teams additionally define intermediate color classes [39] (like yellow-orange) to deal in a more sensible way

---

[3] `http://www.tzi.de/4legged`

[4] 208x160 pixels, 24bit YCrCb color space, 30fps

[5] S.P.Q.R. [38] from Italy, Cerberus [35] from Turkey, Tec Rams from Mexico [40], Wright Eagle [43], sharPKUngfu and TsinghuaHephaestus from China, and two German spin-offs: the Hamburg Dog Bots [36] and the Microsoft Hellhounds

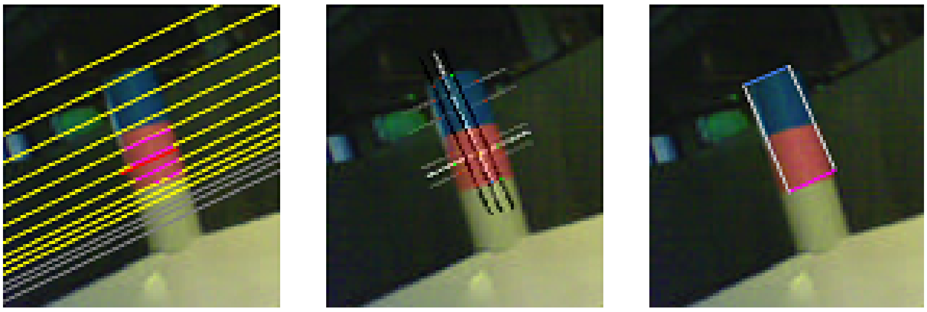


Figure 1.2: Image analysis as performed by the German Team (winner 2005). **Left:** Object seeding; raw images are scanned along sparse grid lines. Pink pixels trigger the flag specialist. **Middle:** Object specialists; flag specialist determines second color and exact dimensions. **Right:** Final flag percept

with overlapping color class regions in color space. The color-to-colorclass mapping is usually performed using a look-up table, that has to be provided to the robot a-priori.

Image scanning The color class image is then converted into an intermediate representation. Many teams scan through the image along (sparse) grid lines [41, 42] to generate a list of object candidates (seeds). Other teams convert the whole image into a blob representation[35, 39]: connected pixels of the same color class are grouped together, and are subsequently represented by only the corresponding bounding box and a color class tag.

Object recognition This intermediate representation is then scanned for objects. In the case of candidate lists, pixel-based object specialists prove or disprove the object seeds. For example, the grid line scanner of the German Team generates a candidate whenever a long run of pink pixels is detected. Then the flag specialist determines its second color and the exact dimensions (see figure 1.2). In the case of blob representation, the detected bounding boxes are translated into real objects using large object libraries. The object library of the nuBots [39] contains for example a rule about orange balls: a small yellow-orange blob above a big orange blob above a small red blob (see figure 1.3).

Considering this, two striking limitations become obvious:

Manual calibration of color classes Image segmentation in most teams relies on the existence of a color-to-colorclass mapping, which has to be provided to the robot beforehand. Usually this calibration process involves hours of manual color tagging, and requires constant illumination over an extended period of time. This renders demonstrations at arbitrary places practically impossible.

Some teams work on automatic color calibration techniques, like UT Austin Villa [27] from the USA or the Dutch Aibo Team [44].

Other teams have tried to replace the static color mapping with dynamic approaches. Cerberus[35] from Turkey and SPQR [38] applied dynamic clustering techniques.

As processing time is crucial, consequently on-line approaches for dynamic color (re-)clustering consume expensive processing time. The ability to adapt to illumination changes therefore requires other modules to work faster – which in turn can reduce the overall performance. Additionally, the current implementations of such approaches are prone to diverge – for example when too too many children with colorful cloth are sitting around the field.

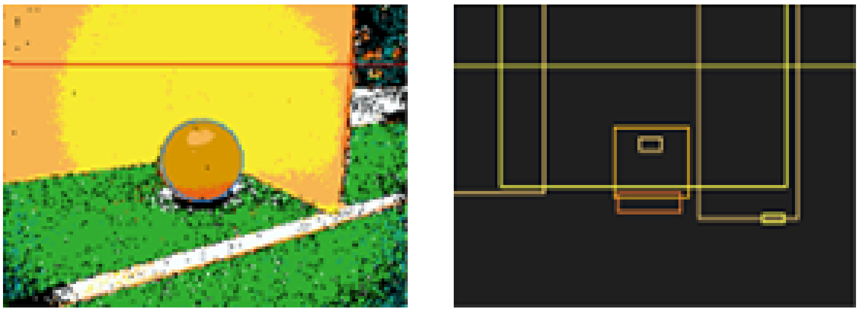


Figure 1.3: Image analysis as performed by the nuBots (winner 2006). **Left:** Image segmentation into color classes **Right:** Resulting bounding boxes of color class blobs

Dependency on artificial landmarks The second limitation is the full dependency of all approaches on the fixed set of artificial landmarks. All approaches inherently include this a-priori-knowledge in the form of predefined color classes and hard-coded object detectors. Therefore, most teams can only play on official soccer fields (with currently 6 landmarks).

Two teams[6] demonstrated (but not yet published) at this year's open challenge that their perception works even without the flags – solely by localizing on field lines and the (colored) goals.

These aspects certainly point out two serious shortcomings of current approaches. Autonomous robots of the future should be able to adapt intrinsically to new environments on their own – or at least keep human intervention at a minimum. Moreover, approaches requiring special environments (with artificial landmarks installed) are not desirable as this confines the robots to a small subset of possible applications. This does not mean that an agent should not take advantage of existing landmarks; but it should not rely on them a-priori.

Localization

Typically, the perceptions provided by vision are noisy and incomplete. For example, the Dutch Aibo Team reported [49] that only 25% of the visible flags had been detected and around 70% of the visible goals. False positives were not common in the lab (1%), but yielded serious problems at the competition site.

Localization integrates the odometry data from motion and the detected objects from vision in order to update the robot's beliefs about its current pose. Typically next to the robot's pose on the field also the ball coordinates are tracked by a (second) localization filter.

Mainly two localization techniques are used in practice:

(Extended) Kalman filters (EKF) model the robot's belief by a single Gaussian. Effective in computation and easy in implementation, this is a common choice for many teams. The Dutch Aibo Team uses a Kalman filter for ball modelling.

Monte-Carlo localization (MCL) can more easily cope with ambiguities as they often occur in robot soccer. Therefore MCL methods are applied by most teams [16] for position tracking. However they are computationally much more expensive; the Dutch Aibo Team uses them for the robot pose with a relatively small number of samples.

---

[6] Tsinghua Hephaestus and Wright Eagle from China

The reported accuracies of the teams differ significantly. The German Team claimed to be able to localize at approximately 6cm accuracy (on the old field), while the Dutch Aibo Team could only reproduce an accuracy of approximately 13cm. A few teams [40] even reported localization errors bigger than 30cm, which is – in consideration of the 80cm wide goal – significantly imprecise.

Behaviors

Localization supplies the robot with both knowledge about its own pose and the position of the ball. Action planning and decision making then is performed by the behavioral layer. This name already indicates that typically no deep inference takes place: many teams (like the Dutch Aibo Team[49]) use state machines as a behavioral language, others ([41, 39, 37] even use interpreted scripting languages (like python) as the typical processing time needed for behavioral evaluation is relatively insignificant. The implemented behaviors can be thought of as little programs that enable the robot to perform specific tasks. Therefore, they are also referred to as (behavioral) skills:

Player skills refer to single players only, like the skill to position the robot at its kickoff position, approaching and grabbing the ball, or kicking the ball. Several teams use automatic learning for the ball handling skills.

Team skills refer to team play, where multiple team mates interact with each other (in a way that is typically beforehand negotiated by the robots using wireless communication). Role allocation for example prohibits that a robot that is busy with the ball is interrupted by an approaching team mate. Many interesting things can be done here, like passing the ball between two players (Passing Challenge in 2006[7]) or even guiding a blind-folded robot over the field (as demonstrated in the Open Challenge by the German Team in 2006).

Summary In this short review about the state-of-the-art in the 4-Legged league it can be seen that playing soccer with Sony Aibos is substantially far developed: The soccer games look interesting and are also appealing for the audience. Great improvements have been made with quadruped walking, and the robots of most teams can localize reliably. This year's challenges have proven that team play (i.e. ball passing) is in principle possible; however, it will take a while until these skills will really become visible in actual games.

The reason for this is that the higher modules (like the team skills, but also the player skills and even localization) can only perform as well as the underlying modules do as they dependent on them. As attractive as research on higher level behaviors may be, some things might just render impossible with the limitations of the current solutions. This brings back the main focus on the three basic layers of the robots: motion, vision and localization.

Rule changes in the 4-Legged league (and also in other leagues) indicate that within a few years, the amount of artificial landmarks will be greatly reduced[8], and the precondition of constant illumination[9] will be discontinued.

Therefore, it seems to be mandatory to find ways how mobile robots (i.e. Sony Aibos) can localize in more natural environments without a-priori knowledge about the site. In the next subsection, a literature review of approaches applied outside the RoboCup world is given with the aim to be inspired on how to remedy the limitations of the current approaches in the 4-Legged league.

---

[7] The Dutch Aibo Team demonstrated the work done by a DOAS group of the UvA at this challenge.

[8] See the almost-SLAM challenge in 2005, the New Goal challenge in 2006, and the reduction to 2 (instead of 4) flags in 2007.

[9] As already implemented in the midsize league in 2006

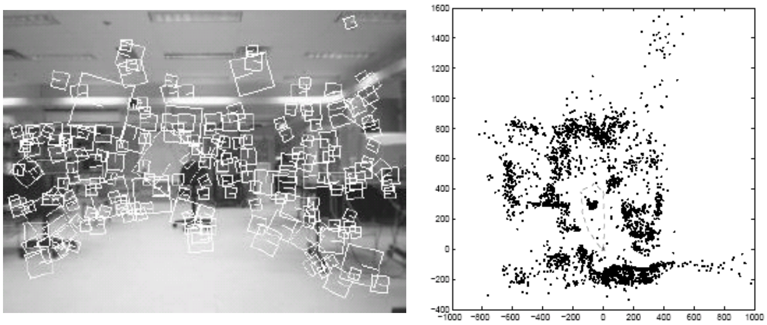


Figure 1.4: SLAM approach using SIFT features [26]. **Left:** Image with SIFT features marked **Right:** Bird's eye view of the SIFT landmark map after training

1.3.2 Other approaches

Approaches from the visual SLAM domain

Promising solutions that remove the dependency on specific landmarks can be found in the visual SLAM domain. Some approaches create dynamic landmarks on-the-fly – for example by extracting **SIFT features** [18] from the images.

A well-working implementation on a stereo-vision robot is described in [26]: for each frame, the robot matches the detected SIFT features in the left and right image (typically 100 features per frame, see fig. 1.4(left)). By triangulation, the robot estimates the three-dimensional coordinates of the features and subsequently adds them to its 3D SIFT database map as a new dynamic landmark. If the robot then moves along, it matches newly perceived features with the map (fig. 1.4(right)) – by finding the correspondence using a least-squares-minimization. Except for the fact that SIFT features are computationally expensive to extract (their approach works at 2Hz on contemporary hardware), approaches like this prove that accurate localization of mobile robots is possible in natural environments.

Approaches on visual homing

Several approaches inspired by biology support the landmark paradigm – but reduce its complexity significantly. The source of inspiration is that small animals (such as insects) navigate through natural environments seemingly with little effort. Honey bees and desert ants – despite their relatively simple nervous system (and implicitly small data storage capacity) – are able to find their way back to their hive or anthill. An approach for **visual homing** is described in [21], based on the **snapshot model**:

The robot is equipped with an omni-directional camera allowing it to take panoramic pictures. The robot divides its surroundings into 360 sectors, each of which can either be occupied by a landmark or be empty. At the target pose, the robot takes a snapshot of the scene. When the robot is kidnapped, it seeks a correspondence between the new image and the recorded snapshot – simply by correlating each sector with the closest matching neighbor. Each sector correspondence then produces a motion vector – either to the left or to the right, with its length corresponding to the disparity of the match. Finally, all sector motion vectors are summed up and guide the robot iteratively towards its target pose. Of course, the initial assumption of a black and white world is relatively strict (and can be given up [33]). Still, this approach shows that successfull navigation in natural environments not necessarily requires complex algorithms – and is (despite the limitations) feasible in real-time.

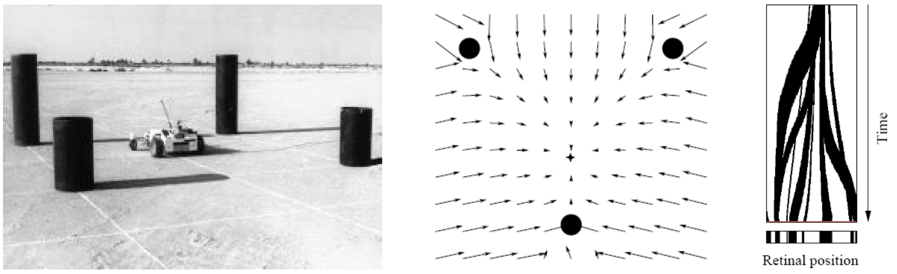


Figure 1.5: Biologically-inspired approach [21] using homing vectors. **Left:** Example of a landmark array used for the robot experiments. **Middle:** Homing vectors can be assigned to each point on the plane. **Right:** Transformation of the current view over time (upper part) towards the snapshot (lower part).

Purely appearance-based approaches

The class of appearance-based approaches works so efficiently because they treat the converted images as a single feature vectors; that is somehow produced by the underlying vision system. This motivates purely **appearance-based** approaches, with a minimum on image preprocessing.

An interesting approach using no preprocessing (except for image rectification) is presented in [14, 15]: a mobile robot is equipped with both a omni-directional camera and a magnetic compass. The robot then compares each captured image with the previous one, by computing the (Manhattan) distance between the two images. The new image is now pixel-wise rotated horizontally, to find the rotation under which the distance gets minimal. This rotation is then reported to the robot as differential compass information. In relatively short runs (around 400 images), an accuracy better than $10°$ degrees was measured. This was significantly better than the magnetic compass, that occasionally showed derivations of up to $20°$. However, this approach to visual compassing is prone to long-term drift, because only the two previous images are evaluated. A second drawback is that the proposed distance measure is computationally extremely expensive; the visual compass could therefore only evaluate images at $2fps$.

Another interesting example with minimal preprocessing is described in [10, 9]: a mobile shopping assistant equipped with an omni-directional camera is moved along a known path through a store. The robot splits the panoramic images into a small number of sectors (around 10) and computes the mean RGB value of every sector which it subsequently stores in its internal map. After sufficient training spots have been visited (around $4spots/m^2$), the robot is kidnapped to an unknown place and moved along a unknown path through the store. A particle filter then removes ambiguities and subsequently tracks the robot's pose. The robot was able to localize successfully, and the measured mean error in localization was around $25cm$. All computations could easily be performed in real-time. This contemporary research indicates that the domain of such light-weight approaches is not yet fully explored and presumably holds hidden potential.

Summary In this section, a review about contemporary research in the domain of mobile robot vision and localization was given. Successful approaches from the world of RoboCup have been analyzed that had been designed for real-time applications in soccer robots. Although this means that the applied approaches are somehow outdated, the RoboCup challenge implicitly filtered the scientific output efficiently in two categories: well-working in practice, and only successful under laboratory conditions.

At the same time it is clear, that novel approaches from research exist that can perform much better: this motivates the analysis of promising approaches from outside the RoboCup world. However, such

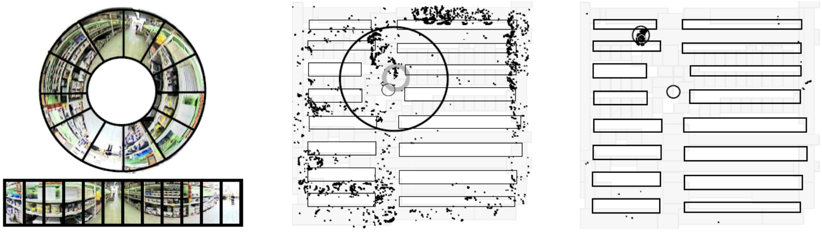


Figure 1.6: Appearance-based localization. **Left:** A panoramic image is split into sectors. A single sector is represented by its mean RGB values. **Middle:** Localization after 3 frames. **Right:** Localization after 8 frames.

approaches only work at low frame rates or make simplifying assumptions about the environment. Nevertheless, the ideas behind theses approaches certainly will contribute to the spectrum of possible solutions in the future.

1.4 Thesis goal and research question

A central goal of robotics and AI is to be able to deploy a mobile robot that acts autonomously in the real world. As no specific assumptions can be made about natural environments, it is commonly asserted that the robot must be able to learn from its surroundings.

The most striking limitation of current RoboCup approaches seems the dependency on artificial environments, but interesting approaches designed for natural environments (outside the RoboCup world) are already within reach. Switching from artificial to natural environments puts the primary focus on vision:

Focusing on natural instead of artificial environments has its main focus on perception:

The research question therefore is to determine in which way approaches to vision (and localization) can be extended for a real-time reactive robot to cope with natural environments. The thesis goal is then to develop a novel approach that enables a mobile robot to quickly adapt to unknown environments while at the same time requiring as little as possible human assistance.

1.5 Outline

Chapter 2 gives a short theoretical introduction to probabilistic robotics. The main focus hereby lies on motion models, sensor models and localization filters. In **Chapter 3**, the mathematical foundations of a novel appearance-based approach is derived that can be used for mobile robot localization in natural environments. A real-time implementation of this approach for a Sony Aibo is developed in **Chapter 4**, bridging the gap from theory to practice. **Chapter 5** describes the conducted experiments and presents the results. In **Chapter 6**, the approach is analyzed in terms of the research question and discussed using existing techniques. Finally, this chapter concludes by summarizing directions for future research that have been identified throughout the project.

The work described in this thesis has been carried out at the Autonomous Intelligent Systems group at the Informatics Institute of the Universiteit van Amsterdam. It builds on software developed by the Dutch Aibo Team.

Publications

Parts of this thesis have been published in the proceedings of international conferences:

The pre-requisite of automatic color calibration has been introduced in [44] (along with an extension to make the approach usable in classical, landmark-based robotic soccer). The main contribution (the visual compass approach) has been described in [45]. Finally, the extension towards positional localization has been presented in [47].

Chapter 2

Probabilistic robotics

In this chapter, an introduction about probabilistic robotics will be given [31]. In particular, the basic concepts for probabilistic motion, sensors and localization techniques will be discussed. At the same time, a clear vocabulary and corresponding mathematical nomenclature will be established that will be used consistently in the remainder of this thesis.

The introduction starts with a definition of **motion models** that try to describe the action dynamics of a mobile robot. After that, the concept of **sensor models** will be explained. Sensor models try to relate the perception of the sensors to the world state. Finally, the concept of **localization filters** is introduced that apply the motion and sensor models to update the robots beliefs.

2.1 Motion

As most mobile robots stand on the ground, their **pose** $x =< \phi, \tau >$ can usually be described by three degrees of freedom: their two-dimensional translational position $\tau = \begin{pmatrix} x & y \end{pmatrix}^T$ on the ground relative to some external coordinate frame plus their angular rotation ϕ (orthogonal to the ground plane). The translational part it is commonly referred to as the robot's **location**, while the rotational part is also called the robot's **bearing** or **heading direction**.

The robot can alter its pose by sending **control actions** to its **effectors** (like wheels or legs). The calculus describing the effects of control actions on the robot pose is called **kinematics**. In contrast, the calculus of **inverse kinematics** is concerned with deriving the necessary control actions that will (reversely) allow the robot to change from one pose into another.

In real world situations, the **deterministic calculus** of classical kinematics becomes inadequate quickly. In practice, the outcome of a control action is therefore only predicable within probabilistic terms. The reason for this is the **noise** of various forms inherent to real-world robotics. Noise occurs internally in the robot (for example due to friction in the thread of the motor), in-between at the moment of power transmission (like slippage of the legs) or it can also be purely external (for example due to a sudden push by another robot).

The only possibility to deal with these kinds of noise (and uncertainty) is by refining the deterministic motion models into probabilistic ones. **Probabilistic motion models** describe the relationship between the previous and current pose and the issued set of commands by a probability distribution.

As motion models descend from the field of dynamic control theory, it is common to consider the robot's pose as a state $x(t)$ changing continuously over time. For the sake of tractability, the state is typically only evaluated at discrete time steps, and therefore the pose is usually indicated by x_t.

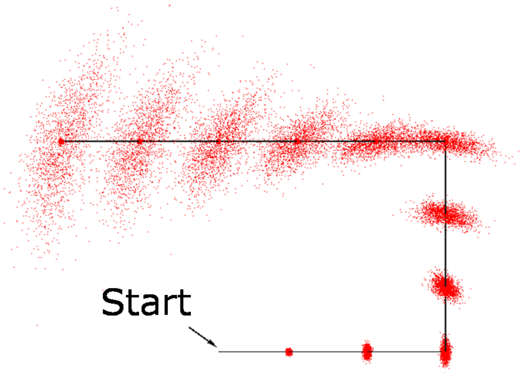


Figure 2.1: Visualization of the effects of a probabilistic motion model [31]. Here a mobile robot moves along a simple path. Its uncertainty about its pose grows the further the robot moves. Each red dot corresponds to a pose sample that has probabilistically been sampled from the motion model.

Definition 1. *A probabilistic* ***motion model*** *defines the state transition between two consecutive time steps* $t-1$ *and* t *after a control action* u_{t-1} *has been carried out. It is expressed by the posterior distribution* $p\left(x_t|u_{t-1},x_{t-1}\right)$.

In order to derive this conditional, it is necessary to derive the kinematics from its mechanics on the one hand and its possible sources of noise and deviation on the other hand. The calculation of the (deterministic) kinematic equations is the easy part; in practice, it turns out to be much more difficult to model the relevant sources of noise adequately. An example of a noisy motion model on the robot pose is visualized in fig. 2.1.

To begin with, the robot's actions might have already been carried out imprecisely by the hardware. Motors have certain acceleration and de-acceleration times which can be hard to predict. Additionally, the robot's wheels or legs could encounter mechanical difficulties by slipping on smooth surfaces. Furthermore, the robot could get stuck to an object next to it which can change its orientation or even block its way entirely without noticing this. It could even happen that a robot is suddenly kidnapped (for example by a referee in a soccer game) and put back at a totally different place. In the end, the fuzziness of the transition model should nicely reflect the probabilistic outcome of the robot's actions in the real world.

In the basic form of motion models no feedback is involved. Therefore, this class of motion models is also referred to as **open-loop**. Thanks to the simplicity, open-loop models are an attractive choice; however, they have a serious drawback: pure open-loop approaches are intrinsically prone to let the robot's pose estimate drift away fast from its true position (or **ground truth**).

A solution to this problem are **closed-loop** approaches. These approaches take advantage of a motion feedback mechanism – for example by additional sensors in the robot's actuators. Closed-loop approaches do not blindly assume that their control actions have been executed correctly. Sensors in the motors or joints can give feedback to a robot about the current configuration of its legs. This means that instead of formulating the motion model in dependence of the control actions, it can be expressed in terms of motor data (or **odometry**). Closed-loop motion models $p\left(x_t|u_t,x_{t-1}\right)$ use instead of the control action u_t the data from odometry (which is in practice also denoted by u_t).

As an example, the motion model as used in the soccer code of the Dutch Aibo Team is modelled as

$$x_t = x_{t-1} + u_t + \epsilon_{error} \tag{2.1}$$

in which the noise ϵ_{error} is assumed to be

$$\begin{aligned}\epsilon_{error} &= <\epsilon^{\phi}_{error}, \begin{pmatrix} \epsilon^{x}_{error} \\ \epsilon^{y}_{error} \end{pmatrix}> \qquad (2.2)\\ &= <0.002\sqrt{(u^x_t)^2+(u^y_t)^2}+0.2u^{\phi}_t \cdot X^{\phi}, \begin{pmatrix} 0.10\sqrt{(u^x_t)^2+(u^y_t)^2}\cdot X^x \\ 0.02\sqrt{(u^x_t)^2+(u^y_t)^2}\cdot X^y \end{pmatrix}> \qquad (2.3)\end{aligned}$$

with $X^x, X^y, X^{\phi} \sim U\,[-1;1]$ being independent uniformly-distributed random variables.

2.2 Vision

A robot typically possesses several sensors to perceive its environment. Researchers can choose to equip their robots with a variety of different sensor modalities. For example there are **tactile sensors** that react to touching, **range sensors** using ultrasone or laser beams, **microphones** that record sound, **cameras** that can take snapshots or capture video and **wireless receivers** such as networking and GPS (for communication and localization).

From the spectrum of possible sensors, laser range scanners are often used for robot localization. However, these sensor modalities have a number of problems – particularly for the use in natural environments. So for example the limitation in range makes them difficult to apply in large open areas, where in general no stable landmarks can be observed. Additionally, they tend to be easily confused in dynamic environments (for example by people walking around that occasionally block the laser beams). Lastly, sensors using distance measurements are considered to be counterintuitive to human perception.

Therefore, vision has long been advertised as providing a solution to these problems. From a conceptual perspective, it is desireable that a robot could perceive the world in the same way that humans do. The real world is full of visual indications that have especially been installed to facilitate (human) localization, such as public signage, arrows or other artificial landmarks. Moreover, the behavior of a straying robot under difficult (or confusing) circumstances then becomes easier to trace.

Cameras have become a lot cheaper in the past years, and the existence of digital signal processors (DSPs) have made the application of cameras even appealing in non-computerized devices (such as an optical mouse). Mobile robots can easily be equipped with cameras, thanks to standardized hardware protocols (like USB and FireWire). In general, mobile robots are equipped with one (or more) cameras; the two camera types most used for mobile robots are:

- **Conventional cameras** (also called directed cameras) capture the scene by projecting it on the internal image sensor. Although in principle a pinhole aperture would suffice, typically one or more lenses are used to improve the light sensitivity. Conventional cameras have a limited field of view which is typically denoted by their opening angle (or inversely related, their focal length). The opening angle of the Aibo camera for example is 57^o in the horizontal and 45^o in the vertical direction – which is close to the opening angle of typical consumer photo cameras (comparable to a $f = 35$mm lens).

- **Omni-directional cameras** have a hemispherical field of view, usually thanks to a convexly-shaped mirror. Therefore, they have an (almost) 360^o view in the horizontal direction and a normal opening angle in vertical the direction. Sometimes, two such cameras are mounted together back-to-back, to form a truly omnidirectional sensor. As conventional cameras are used internally (below the mirror), the images are usually rectified into **panoramic images** using epipolar image geometry [30]. Panoramic cameras are used with great success in the RoboCup midsize league and for research of SLAM [1] approaches.

---

[1] Simultaneous Localization and Mapping

Whatever kind of camera is used with a mobile robot, it is obvious that the robot has to reduce the amount of data significantly before the data becomes meaningful to the robot. The images are therefore converted into an intermediate representation. This conversion from raw images to a **feature space** can take place at different levels of complexity:

- **Low-complexity** approaches feature only minimal preprocessing of the sensor data. A good example are **appearance-based** approaches. The image transformation can typically be implemented fast and efficient, making such approaches interesting for real-time vision in mobile robots. Examples of such approaches were presented in the introduction [21, 10, 9].
- **Intermediate complexity** preprocessing can for example be found in the class of **landmark-based** approaches. Although feature detectors for static landmarks can be implemented efficiently (as in the case of soccer-playing robots [41]), dynamic landmark detectors (using salient features like SIFT [18]) are typically computationally more expensive – and are not yet applicable in real-time reactive mobile robots.
- **High-complexity** preprocessing and feature extraction can for example be found in (human) **pose, gaze or gesture recognition**. In these settings, visual objects not only have to be recognized, but also their pose and orientation has to be determined yielding much more details. However, for mobile robot localization these approaches are not (yet) relevant.

Usually, this intermediate feature-space representation is then the primary output of a visual sensor. In order to derive useful information, the robot has to know how the world state and the sensor percepts are related.

The concept of **sensor models** (or observation models) is to express the relationship between the world state x_t and a **sensor reading** (or observation) z_t mathematically. Typically, in addition a map m of the environment is provided, which allows the formulation of location-independent (or **generic**) sensor models.

In a deterministic world this relationship could be expressed directly by a **sensor function** that maps a world state to sensor readings: $z_t = f_m(x_t)$. As however all sensor data is inherently subject to noise under real-world circumstances, a deterministic sensor model is normally insufficient. Therefore, sensors are commonly described by **probabilistic** sensor models.

Definition 2. *A probabilistic* ***sensor model*** *describes the relation between a world state x_t and a sensor reading z_t given an environmental model (or map) m in form of a posterior distribution $p\left(z_t|x_t, m\right)$.*

The Dutch Aibo Team uses in fact several maps. The first map is a pre-supplied lookup table used to convert real colors into color classes in the feature conversion step. The other maps basically contain the positions of the landmarks; those maps are stored in a pre-cached form such that the sensor model can be evaluated faster (see fig 2.2).

If the maps are supplied to the robot from an external source, they are called **a-priori** knowledge. Manually created maps often increase the performance of an approach, but require human assistance. Moreover, the robot is strictly bound to the domain where the maps were designed for. A remedy form approaches that learn the required maps autonomously.

Approaches can be classified as passive or active vision. **Passive vision** analyzes incoming images completely from top to bottom – without any memory and/or feedback. Approaches applying **active vision** however, dispose of a notion of attentional focus. Such approaches can steer their attention, for example by tracking an object of interest. This requires the existence of attentional feedback. This feedback can take place at different places:

- The feedback is generated by the sensor and used only **internally in the sensor** itself. This means that the sensor has some kind of memory, where it for example stores the coordinates of tracked

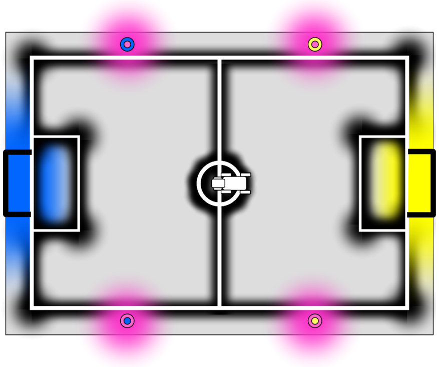


Figure 2.2: Visualization of the probabilistic sensor models used by the Dutch Aibo Team. The sensor models expect the their corresponding landmarks within fuzzy probability clouds around their true position to explicitly allow for camera (and head motion) noise. The sensor models for different landmark percepts have been marked with different colors.

objects (maybe even augmented by motion vectors). When a new image arrives, vision can efficiently update those coordinates by a local search without scanning the complete image over and over again. Active vision can therefore speed up computations enormously. Moreover, as vision then already integrates over multiple frames, such approaches are much more robust against wrong percepts. However, this concept is difficult to implement in the 4-Legged league: firstly, the robot's head turns rapidly and secondly, the environment is highly dynamic. Therefore, this kind of inter-frame object tracking would certainly not improve the performance substantially.

- Another possibility is that the robot has some **actuators that can be controlled by vision** – as for example in the case of a turnable head camera: when the behavioral layer of the robot decides to put focus on the ball, vision then can steer the head camera in such a way that the ball is kept accurately within its view [41]. Due to the limited opening angle of the Aibo head camera, this kind of active vision is used by almost all teams in the 4-Legged league.
- Moreover, the feedback can have its **source outside of vision**, like for example in the behavioral layer. The robot could decide to walk back to its goal and tell vision not to get distracted from an approaching ball. A promising approach of this type of active vision has been developed by the Dutch Aibo Team in [20]: here the behavioral layer can specify on which types of objects vision has to focus. As a result, the performance of the goalkeeper has been increased significantly.

2.3 Localization

Robot localization is the problem of estimating the robot pose relative to a map of the environment. Several subproblems of the general localization problem exist, and in the following a short overview of the four main problem classes is given (ranked according to the degree of difficulty):

1. The localization problem in terms of **position tracking** assumes that both that a map of the environment is available and the initial position of the robot is known. Of course, the robot cannot sense its pose directly, but has to use the sensor model to update its beliefs.
2. **Global localization** starts without prior knowledge of the initial pose. Depending on the map and the kind of sensors the robot possesses, the robot may have to deal with ambiguities until sufficient

information from the sensors has been aggregated. As soon as the robot's beliefs about is pose converges to a single position, the robot can switch to (simpler) position tracking.

3. In certain applications, it can happen that the robot is – suddenly and unnoticeable to the robot – transported to a unknown situation. If **kidnapping** can occur, the robot has to reconsider its global localization constantly. The robot then needs mechanisms to be able to detect and recover from suddenly wrong beliefs. In the 4-Legged league, kidnaps for example happen when the referee decides to remove a robot due to foul play (like pushing a striker or an illegal defense).

4. In a SLAM[2] type problem, neither the map nor the initial pose is known a-priori. The robot has to explore its environment and thereby construct its own map – relative to its origin. As the robot has to simultaneously estimate its own pose and build the map, both of them become increasingly inaccurate the further the robot travels. Such errors due to long-term drift can partially be corrected when the robot detects that it has crossed its own path. Then the robot can apply **loop-closing** techniques in order to rectify the map. When multiple robots are used to explore the map, additional issues such as team coordination and subsequent map merging can be addressed.

Most approaches assume a **static environment**, meaning that nothing in the map changes over time. Such an assumption is not always a valid, and serves therefore as an approximation for **dynamic environments** – like the real world. In the soccer domain, moving robots, referees or the audience can change the appearance significantly by hiding landmarks or producing false positives. The Dutch Aibo Team for example had at the 2005 championships in Osaka the problem that in some games the goalkeeper lost completely its orientation. After thorough logfile analysis, it turned out that while some games, a score board next to the field was activated – and been mistaken as a flag.

Some approaches use **active localization** (which is the opposite to **passive** localization) that allows the robot to move to a location where it can remove ambiguities. Some of the behaviors of the Dutch Aibo Team soccer robots reflect this principle: for example in the ready-phase[3], they look around and try to establish their localization. If this fails, they turn around with the aim to see additional landmarks.

2.3.1 Belief distributions

Due to the real-world assumption, a robot can never reside be completely certain that it is residing in a single world state x_t. It is more likely, the robot will believe to be close to a estimated world state $\hat{x}_t$ associated with a measure of certainty in a probabilistic manner.

A common way to express this uncertainty is by a belief distribution that associates with a state x at time t a belief value $Bel_t(x)$. In such a general belief distribution the robot for example could also represent when that it is ambivalent about two or more different poses – for example due to prevailing field symmetries.

The robot can include maximally all its previous sensor readings and issued motion actions in its belief distribution about the current world state. Statistically, this can be formulated by the following dependency:

$$Bel_t(x) = p(x|z_t, u_{t-1}, z_{t-1}, ..., u_1, z_1, u_0, z_0) \tag{2.4}$$

In general, belief distribution are real distributions and are therefore supposed to sum up to 1. Depending on the further processing, such a constraint can be superfluous. Then $Bel_t(x)$ becomes a likelihood distribution. This could be the case when a robot subsequently only is interested in the world state associated with the strongest belief. The belief representation used by the Dutch Aibo Team is for example not normalized, and the soccer robots estimate the validity of a certain pose solely relative to their beliefs.

---

[2] simultaneous localization and mapping

[3] In the ready-phase, the soccer robots have to walk on the kick-off positions.

2.3.2 Bayesian filters

Most probabilistic localization techniques are based on the assumption that both control motion commands (or odometry readings) and sensor measurements are conditionally independent of previous readings, given the current state x_t. In this context the localization problem becomes a **HMM**[4] **state estimation problem**: the hidden state to be estimated is the current pose x_t: it is assumed that it depends solely on the previous state x_{t-1}, the current motion command u_t and sensor reading z_t.

This approximation is called the **Markov assumption**: it is assumed that the statistics about what happened in the past $(x_{0:t-1}, z_{1:t-1}, u_{1:t-1})$ is already completely summarized in the belief Bel_{t-1} of the previous state.

Definition 3. ***Markovian localization filters*** *maintain the robot's beliefs Bel_t about the current pose by some internal representation. Moreover, localization filters contain methods to update these beliefs corresponding to a control action u_{t-1} and a sensor reading z_t. They apply the motion and sensor models to estimate the new posterior Bel_t based on the previous beliefs Bel_{t-1}. A Markovian localization filter is then defined by the posterior distribution $Bel_t(x) = p(x|Bel_{t-1}, u_{t-1}, z_t)$.*

This belief update is typically split up further, by conditioning firstly solely on the motion command, and subsequently on the current sensor reading:

$$Bel_t(x) = p(x|z_t)p(x|Bel_{t-1}, u_{t-1}) \tag{2.5}$$

This is of course requires the additional assumption stating that the sensor readings and motor commands are mutually independent, which is in practice a valid approximation. The important point about eq.2.5 is that the motion and sensor models now appear separately.

However, the motion and sensor models typically exist only opposite conditionality $p(x|Bel_{t-1}, u_{t-1})$ and $p(z_t|x)$ respectively (see definition 1 and 2). The posteriors as required by (2.5) can be derived by applying Bayes' Theorem:

$$p(x|Bel_{t-1}, u_{t-1}) = \frac{p(u_{t-1}|Bel_{t-1}, x)p(x)}{p(u_{t-1})} = \delta_t \cdot p(u_{t-1}|Bel_{t-1}, x) \tag{2.6}$$

and

$$p(x|z_t) = \frac{p(z_t|x)p(x)}{p(z_t)} = \epsilon_t \cdot p(z_t|x) \tag{2.7}$$

The priors for a particular world state $p(x)$, a motion command $p(u)$ and a sensor reading $p(z)$ are typically not in the focus of interest and therefore they are usually replaced by two normalization constants δ_t and ϵ_t. When the priors are assumed to be distributed uniformly, then the main effect of these factors is to ensure that $Bel_t(x)$ stays a belief distribution (i.e. sums to 1).

In practice, the normalization constants δ_t and ϵ_t are often omitted – especially in the case that $Bel_t(x)$ is only a likelihood distribution (section 2.3.1). The normalization of the belief distribution is then either not performed at all, or only after the new belief distribution has been computed (which is computationally more effective than calculating the priors in the first place).

Thereby, the belief distribution becomes computable by the update rules of Bayesian filtering:

1. The robot's beliefs are updated according to the estimated motion u_t. This is accomplished by propagating the belief distribution from each state through the motion model $p(x|Bel_{t-1}, u_{t-1})$ (the odometry-corrected, intermediate belief is here indicated with $Bel_t^-(x)$):

$$Bel_t^-(x) = p(x|Bel_{t-1}, u_{t-1}) \tag{2.8}$$

---

[4] hidden Markov model

2. After that, information from the sensor reading is incorporated into the intermediate belief by applying the sensor model $p(x|z_t)$, using a mixing constant λ:

$$Bel_t(x) = \lambda p(x|z_t) Bel_t^-(x) + (1-\lambda) Bel_{t-1}^-(x) \tag{2.9}$$

The mixing constant λ determines how much weight is put on the new sensor reading $p(x|z_t)$. For example, $\lambda = 0$ rejects new sensor readings completely and solely trusts on the earlier (but odometry-corrected) beliefs, while $\lambda = 1$ fully adopts the posterior of the sensor and immediately forgets previous beliefs.

2.3.3 Solution techniques

In practice, it is not possible to compute the posterior analytically (as stated in (2.5)). Therefore, a number of approximations exist. In the following, the most common numerical solution techniques applied in robot soccer will be presented in brief.

The main problem for implementation is that the belief distribution $Bel(x_t)$ may stand for any arbitrary probability distribution that is constantly changing over time, which in practice will have to be approximated in one way or the other. In the following, popular approximations for belief representation will be introduced briefly. All of them impose additional assumptions or simplifications mostly concerning the shape of the belief distribution.

These approaches can in general be split into two categories: **single-hypothesis trackers and multi-hypotheses trackers**. As they originate from the domain of signal processing, they are also called *filters*.

Single-hypothesis trackers

Single-hypothesis trackers model the belief distribution Bel_t in such a way that only a single world state is captured. This estimate is then associated by a measure of certainty (or variance), allowing the tracker to broaden (or narrow) the estimated region in the state space.

Kalman filter The most popular single-hypothesis tracker is the so-called **Kalman filter** (or variance filter). The idea is to approximate the belief distribution $Bel(x_t)$ by a multivariate Gaussian distribution defined by its mean μ_t and covariance Σ_t:

$$Bel(x_t) = \frac{1}{\sqrt{(2\pi)^N |\Sigma|}} \exp\left(\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right) \tag{2.10}$$

The Kalman filter further assumes that the state transition A, the motion model B and the sensor model C are linear functions solely depending on the world state or action command, plus a Gaussian noise model Q:

$$\begin{aligned}
&\text{(a-priori state estimate)} & \hat{x}_t^- =& \; Ax_{t-1} + Bu_{t-1} && (2.11)\\
&\text{(a-priori sensor estimate)} & z_t =& \; Cx_{t-1} && (2.12)\\
&\text{(a-priori covariance estimate)} & P_t^- =& \; AP_{t-1}A^T + Q && (2.13)
\end{aligned}$$

The Kalman filter updates its parameters by first computing the Kalman-gain (expressing the dissimilarity between expected and actual sensor readings) and then refining the world state estimate $\hat{x}_t$:

$$\begin{aligned}
&\text{(Kalman gain)} & K =& \; P_t^- C^T \left(CP_t^- C^T + R)^{-1}\right) && (2.14)\\
&\text{(posterior state estimate)} & \hat{x}_t =& \; \hat{x}_{t-1} + K\left(y_k - C\hat{x}_k^-\right) && (2.15)\\
&\text{(posterior covariance estimate)} & P_k =& \; \left(I - KC)P_k^-\right) && (2.16)
\end{aligned}$$

The typical behavior of a Kalman-type filter might be summarized from a bird's eye view as follows: the covariance of the Gaussian belief is initially relatively big. After a while, the belief becomes tightly focused around a single spot due to the sharpening effect of the sensor model. This process comes to an end when the sharpening of the sensor model and the blur of the motion model are in balance. The robot has then reached maximal localization. When the robot is suddenly kidnapped, the predicted sensor estimates do not match the actual ones any more, which increases the uncertainty of the Gaussian belief and thereby enlarges the probability cloud of the state estimate. As soon as the observed sensor readings match well with the estimated ones, the filter will eventually converge again around the new estimate.

The standard Kalman filter forms an optimal solution in the sense that it that it minimizes the mean of the squared error over time. Another appealing property is that its computational requirements are minimal (a few matrix multiplications and additions). The linearity assumption is of course difficult to achieve in many real-life problems, therefore a lot of variations of the original algorithm have been proposed that can cope with different levels of non-linearity. The class of Kalman-based algorithms still builds on Gaussian representations of the robot's belief, but in general it is the aim to approximate the motion and sensor updates in one way to make them linear again.

The **Extended Kalman Filter** (EKF) aims to approximate the non-linearity of real-world models by a first-order linear Taylor approximation. In other words, the motion and sensor models are – for each time step separately – approximated by a linear projection. The non-linear transition function is analyzed locally at the current mean of the Gaussian belief for its tangent, yielding a linear projection matrix that can subsequently be used by the update equations of the linear Kalman filter. Sometimes the Taylor approximation can be solved analytically in advance; otherwise Monte-Carlo techniques for efficient estimation can be used.

Instead of using a first-order Taylor approximation of the models, one could also consider a more complex way of linearization. The **Unscented Kalman Filter** (UKF) approximates the non-linear functions at each time step by extracting a number of so-called sigma-points from the current Gaussian belief. These sigma-points (usually $2N + 1$ points, where N is the dimensionality) are then projected through the non-linear model functions (which usually is feasible for a small number of points), and finally transformed back into a Gaussian representation.

The UKF is of course due to the higher dimensionality of the approximation much more precise than the EKF. However, depending on the kind of non-linearities, it can be hard for both filters to still converge correctly. This is, however, in practice a minor problem.

A major problem, however, emerges from the representation of the current uncertainty in terms of covariance Σ_t. If the robot has no information about the current world state then the covariance becomes infinite, which easily can result in numerical problems. Moreover, Kalman filters have in the case of absolute information absence still a current world state estimate $\hat{x}_t$, which is odd from a theoretical point of view.

Information Filter A better solution are the the so-called **Information filters** that estimate instead of the covariance and estimated state the (Fisher) **information matrix** and **information vector** respectively.

Information filters have the nice property that Fisher information can simply be added up. This has additionally the advantage that independent state variables can be tracked separately and subsequently they can be easily combined – which makes Information Filters particularly interesting for scenarios where multiple robots have to merge their local information with the information collected by other robots.

In general, information filters involve a full matrix inversion for each update step. In particular, for high-dimensional state spaces, this can be numerically instable and moreover, it can be computationally more expensive than the updates performed by Kalman filters.

Multi-hypothesis tracking

The biggest shortcoming, however, is that all single-hypothesis trackers fail to model ambiguities adequately: when, for example, in a symmetrical environment (like a soccer field) two robot poses are equally probable, then the tracker is only be able to track one of them, leaving the other one disregarded.

This restriction is resolved by the class of multi-hypothesis trackers: their representation of $Bel(x_t)$ allows the robot to believe in multiple poses simultaneously, allowing the robot to track and reject different hypotheses independently. As long as ambiguities exist, the multi-hypothesis filters are able to represent them in an explicit way.

Gaussian multi-hypothesis tracks A classic approach to the aforementioned problem is the **multi-hypothesis tracking filter** (MHT) which is a straight-forward extension of the Kalman filter: instead of representing the belief only by one Gaussian, the MHT filter uses a weighted Gaussian mixture. Each underlying Gaussian distribution is tracked independently. For each new observation that has an ambivalent interpretation, each of the constituting Gaussians is split up into two or more (depending on the amount of ambiguity) new child-Gaussians. This of course would lead to an exponentially growing set of Gaussians; therefore, pruning mechanisms can be used to keep the number of distinct tracks computable, by for example removing all tracks with a mixture weight below a certain threshold.

Direct numerical evaluation The **grid localization** methods rely on approximating the belief distribution $Bel(x_t)$ by a discrete distribution on a finite grid of cells. Each grid cell corresponds to a small region of the state space, and each cell is associated with a belief value. The updates from the motion model and sensor model can then be computed directly by applying the update formulas for Bayes filering (as derived in (2.8) and 2.9). When the grid is fine enough, this method works well.

However, in order to obtain an adequate representation of the state space, a huge grid might be necessary that draws both lots of memory and computation time.

Although over the years many ideas have evolved that can speed up these computations (like model pre-caching, delayed updates or selective updating to name a few), grid localization is considered to be a brute-force method that uses processing power in an inefficient way.

Monte Carlo localization A computationally more efficient and even more accurate method is called **Monte Carlo localization** (MCL). Similar methods appear also under the names of bootstrap filters, condensation, sequential Monte Carlo filters, particle filters and survival of the fittest.

The basic idea in Monte Carlo simulation is that a set of weighted samples is used to approximate the belief distribution [8]. These particles can then be updated efficiently according to the Bayesian motion update ((2.8)) and sensor update ((2.9).

In the update step, each particle is processed separately. Each particle is put through the probabilistic motion model, yielding a new (stochastically drawn) particle. Then the likelihood is computed that assuming this particle as the current pose could have produced the observed sensor readings by evaluating the sensor model. Finally, the particle set is re-sampled based on the computed likelihoods. A particle with high likelihood may be re-sampled into multiple particles whereas unlikely particles are completely removed from the set.

Particle filters solve the global localization problem, but are prone to get stuck in local minima. In particular, this has the implication that they recover only poorly from sudden kidnappings. This is due to the fact that, when no particles reside in a region of state space of high likelihood, it can take a long time before particles rediscover this region.

Different extensions have been developed to liberate particle filters from this limitation. For example, it is possible to occasionally inject new particles completely at random in the particle set. Most of them will be directly removed again at the next re-sampling, but some of them may discover the true state faster than the re-sampled particles. Technically, this corresponds to the global addition of ambient background noise to the motion model.

Instead of injecting random particles, it is also possible to sample a small fraction of the particles directly from the sensor model [17]. In the case of landmark-based sensor models, this means that direct triangulation can be used to directly derive candidate poses from the latest landmark sights.

Additionally, the size of the sample set can be adapted dynamically. The idea is that the more focused the particle set becomes (for example by monitoring the Kullback-Leibler distance of the belief distribution [7]), the less particles are necessary for accurate position tracking.

In practice, it turns out that only few particles are necessary for good localization. For example, the localization module of the Dutch Aibo Team only uses 70 particles.

2.4 Summary

In this chapter, the basic foundations of probabilistic robotics – concerning motion, sensors and localization – have been presented. In the following, the main aspects will be summarized:

Motion The motion apparatus determines how the robot moves. It defines the **kinematic** equations of which the robot's probabilistic **motion model** can be derived.

Sensors The sensors determine in which way the robot perceives its environment. Mathematically each sensor can be expressed by its probabilistic **sensor model**. The raw data from the sensor needs in most cases to be somehow **pre-processed**, typically by the conversion into an intermediate **feature space**. The computational costs of pre-processing depend on the choice of the feature space, and this implicitly has effect on the real-time suitability of an approach. **A-priori knowledge** like color classes or pre-defined landmarks can facilitate this conversion, but restrict the use to **artificial environments**. **Active vision** allows the robot to focus its sensors on a region of interest, and can significantly increase the information content of sensor readings.

Localization The general **localization problem** ranges from pure **position tracking** up to **simultaneous localization and mapping** problems. **Localization filters** define how the robot maintains its **beliefs** about the current world state which, in turn, determines whether and how the robot is able to represent **uncertainty** and **ambiguity** in its beliefs. Furthermore, localization filters regulate how information from motion and sensors updates these beliefs. Important properties of localization filters are their **computational effort**, **robustness** against distortions (like slippage, noisy sensor data or kidnaps) and **accuracy**. Filters can be designed to be capable of handling **dynamic environments** by updating their (internal) **map**, and be extended to support **active localization** by for example suggesting disambiguating behaviors to the robot.

Chapter 3

Approach

In this chapter, the visual compass approach will be presented. Robots using the visual compass can quickly learn the appearance of arbitrary natural environments and subsequently estimate their current heading (that is additionally associated with a confidence value).

The basic idea behind the visual compass approach is intuitive: as the robot should be able to adapt to any given environment, a direct approach would be to store a 360^o panoramic image of its surroundings and use it subsequently as a map. Then the robot could match every new incoming image with this map to estimate its orientation.

This matching is not trivial because the new image (or its feature space equivalent) can be translated, rotated, stretched and projectively distorted when the robot is not standing at the spot anymore where the snapshot of the panorama was originally taken. Additionally, regular cameras have a limited field-of-view, and therefore cannot take panoramic pictures.

The compass approach can be divided into two logical parts. In the first part, the compass sensor is derived, which a robot can use to determine its current heading from incoming images. In the second part, a localization filter is described that allows the robot to integrate the raw compass readings over time and merge its motion data.

Throughout the whole chapter, the complexity classes of all major processing steps will be indicated.

3.1 Compass sensor

In the following, the compass sensor is constructed, which can provide heading information by matching camera images f_t with its map m. The approach is introduced incrementally: it starts by assuming a single-direction, single-transition sensor and is subsequently extended to full camera images of a robot heading in an arbitrary direction. Subsequently, it is shown how this approach can be used to estimate (learn) a map $\hat{m}$ in an unknown environment. Finally, the sensor model is derived, that a robot can use to estimate its heading ϕ_t from an incoming camera image f_t (given a map m).

3.1.1 Heading

The robot has at each point in time a heading ϕ_t, which is the rotational part of its pose $x_t =< \phi_t, \tau_t >$. Its pose (including its heading) changes over time, as long as the robot moves. In probabilistic terms, the robot would then like to know the likelihood of that a current image f_t was taken under a certain rotation ϕ (given a map m), which is expressed by the following conditional probability:

$$p\left(f_t|\phi, m\right) \tag{3.1}$$

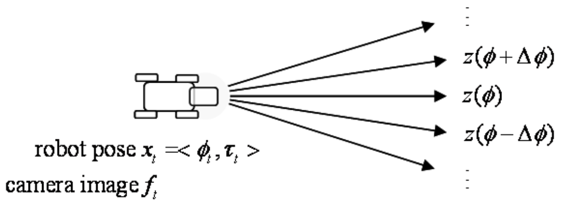


Figure 3.1: A robot is standing at pose $x_t =< \phi_t, \tau_t >$. It measures the frequency patterns $z_t(\phi)$ in different directions ϕ in the current image f_t from its camera.

3.1.2 Single transition

The approach is based on the idea that the colors in a natural environment are not equally distributed. More specifically, the approach exploits the assumption that the occurrence pattern of color class transitions strongly depends on the orientation.

For the moment, it is assumed that a robot has access to a probabilistic sensor that can directly measure the relative frequency of a transition between two (preliminary fixed) color classes i and j given an orientation ϕ. A measurement, taken at time t and under orientation ϕ will be denoted by $z_t^{ij}(\phi)$. These assumptions will be relaxed in the following subsections so that finally a regular color camera can be used.

The relative occurrence frequency of a transition between the two color classes i and j given a direction ϕ is then modelled by the random variable $Z^{ij}(\phi)$ (see fig. 3.1.2). Each time the robot takes a measurement, it reads a slightly different frequency measurement, indicated by the series $(z_1^{ij}, z_2^{ij}, \ldots)$. For localization purposes, the robot needs to estimate the likelihood that these readings were produced by the underlying distribution $Z^{ij}(\phi)$.

In this approach, it has been chosen to approximate $Z^{ij}(\phi)$ by a histogram distribution, defined by the absolute frequencies of n logarithmically scaled bins $m_{(1)}^{ij}(\phi), m_{(2)}^{ij}(\phi), \ldots, m_{(n)}^{ij}(\phi)$.

$$Z^{ij}(\phi) \sim \mathcal{H}\left[m_{(1)}^{ij}(\phi), m_{(2)}^{ij}(\phi), \ldots, m_{(n)}^{ij}(\phi)\right] \tag{3.2}$$

This histogram distribution is then defined by the following probability distribution:

$$p\left(Z^{ij}(\phi) = z^{ij}(\phi) \middle| m^{ij}(\phi)\right) = \frac{1}{\sum\limits_{k=1}^{n} m_{(k)}^{ij}} \begin{cases} m_{(1)}^{ij}(\phi) & \text{if} \quad 2^{-1} < \quad z^{ij}(\phi) \quad \leq 2^{0} \\ m_{(1)}^{ij}(\phi) & \text{if} \quad 2^{-2} < \quad z^{ij}(\phi) \quad \leq 2^{-1} \\ \vdots & \vdots \\ m_{(1)}^{ij}(\phi) & \text{if} \quad \quad z^{ij}(\phi) \quad \leq 2^{-(n-1)} \end{cases} \tag{3.3}$$

The parameters of this distribution $m^{ij}(\phi) = m_{(1)}^{ij}(\phi), m_{(2)}^{ij}(\phi), \ldots, m_{(n)}^{ij}(\phi)$ can be seen to define the map $m^{ij}(\phi)$ – concerning the transition frequency sensor $Z^{ij}(\phi)$.

If the robot now gets the measurements z_t^{ij} from its frequency sensor of unknown direction, it can deduce the conditional probability (or posterior, or likelihood) that this observation was taken under a certain heading ϕ – given a map m^{ij} – by evaluating

$$p\left(z^{ij} \middle| \phi, m^{ij}(\phi)\right) \tag{3.4}$$

using (3.3).

ij	**1**	**2**	**3**
1	37%	8%	5%
2	7%	21%	3%
3	5%	1%	15%

Table 3.1: Visualization of a transition pattern. The robot measures in direction ϕ the relative frequencies $z(\phi)$ of all color class transitions ij for $m = 3$ color classes. The matrix is not necessarily symmetrical, because the transitions frequencies are measured in the images in vertical direction (from bottom to top). The diagonal then corresponds to the relative frequency of occurrence of the color classes themselves.

For a single direction, this evaluation complexity of this expression is linear in the number of histogram bins n ($p\left(z^{ij}\middle|m^{ij}(\phi)\right) \in O(n)$).

3.1.3 Transition patterns

The approach so far only relates to the frequencies $\left(z_1^{ij}(\phi), z_2^{ij}(\phi), \ldots\right)$ of one specific color class transition ij, given the map m^{ij}. This is now generalized by introducing a compound sensor $Z(\phi)$ that measures the relative frequencies of all color class transitions simultaneously:

$$Z(\phi) = \begin{pmatrix} Z^{11}(\phi) & \cdots & Z^{1n}(\phi) \\ \vdots & \ddots & \vdots \\ Z^{n1}(\phi) & \ldots & Z^{nn}(\phi) \end{pmatrix} \tag{3.5}$$

Furthermore, it is assumed that all transition frequency sensors $Z^{ij}(\phi)$ are mutually independent.

Analogously, a n^2-dimensional transition frequency pattern observed under a certain direction ϕ is constituted of the measurements of all color classes transitions:

$$z(\phi) = \begin{pmatrix} z^{11}(\phi) & \cdots & z^{1n}(\phi) \\ \vdots & \ddots & \vdots \\ z^{n1}(\phi) & \ldots & z^{nn}(\phi) \end{pmatrix} \tag{3.6}$$

An example of such a pattern $z(\phi)$ is given in table 3.1.3.

Nontheless, the elements of the n^2-dimensional compound sensor $Z(\phi)$ can still be described by the parameters of the underlying histogram distributions (3.3), thus $Z(\phi)$ is parametrized by the generalized map

$$m(\phi) = \begin{pmatrix} m^{11}(\phi) & \cdots & m^{1n}(\phi) \\ \vdots & \ddots & \vdots \\ m^{n1}(\phi) & \ldots & m^{nn}(\phi) \end{pmatrix} \tag{3.7}$$

If the robot now reads a measurement z of its sensor, it can compute the posterior of $p\left(Z = z\middle|m(\phi)\right)$ – expressing the likelihood that the robot was facing in direction ϕ while measuring z – by exploiting the mutual independency of the individual transition frequency sensors $Z^{ij}(\phi)$.

$$p\left(z\middle|\phi, m(\phi)\right) = \prod_{i,j}^{n} p\left(z^{ij}\middle|\phi, m^{ij}(\phi)\right) \tag{3.8}$$

The evaluation complexity of this expression grows quadratically with the number of color classes m ($p\left(z\middle|m(\phi)\right) \in O\left(m^2 \cdot n\right)$).

3.1.4 Sampling transition patterns from camera images

Of course a robot typically is not able to directly measure the color class transition frequencies. However, such a sensor can be easily constructed if the robot is equipped with a camera.

The images a robot perceives when pointing its camera in a direction ϕ can be denoted by $f_t(\phi)$. Without loss of generality, the images are assumed to be aligned horizontally and vertically, have been cropped in such a way that they include only pixels from above the horizon and moreover it is assumed that they are continuously sampled, and have the unity length in both dimensions. Then, a particular point $< x, y > \in [0,1] \times [0,1]$ can be evaluated by denoting $f_t(\phi) < x, y >$.

In the case of a color camera, the evaluation of a point in the picture will yield a color of a three-dimensional color space $[0,1]^3$.

In order to measure the transition frequencies between two color classes, a function g is required that maps a color $c = f_t(\phi) < x, y >$ on its color class $[c]$:

$$
\begin{aligned}
g :\in [0,1]^3 &\rightarrow (0, \ldots, m) && (3.9)\\
c &\mapsto g(c) && (3.10)
\end{aligned}
$$

For the moment, it is assumed that the robot somehow has access to such a function (see appendix B).

It was chosen to measure the transition frequencies in vertical direction (from bottom to top). A transition at $< x, y >$ between i and j then is defined as a change (in vertical direction) of color class in the image at this point, or mathematically equivalently as

$$
transition^{ij}(< x, y >) = 1 \Leftrightarrow \lim_{\epsilon \to 0} g\left(f_t(\phi) < x, y - \epsilon >\right) = i \wedge g\left(f_t(\phi) < x, y + \epsilon >\right) = j \quad (3.11)
$$

With this proposition available, simply the absolute frequencies of a particular color class transition on the complete image can be counted:

$$
z_t'^{ij}(\phi) = \int_0^1 \int_0^1 transition^{ij}(< x, y >) dx dy \quad (3.12)
$$

These absolute frequencies can be divided by the sum in order to obtain relative frequencies:

$$
z_t^{ij}(\phi) = \frac{1}{\sum_{i',j'}^{m} z_t'^{i'j'}(\phi)} z_t'^{ij}(\phi) \quad (3.13)
$$

Now the robot can measure $z_t(\phi)$ by evaluating an image $f_t(\phi)$ from its camera by evaluating (3.13).

3.1.5 Refining the resolution

The conversion (3.13) from a camera image into the transition frequencies is rather coarse: the whole image $f_t(\phi)$ is sampled into a single frequency measurement $z_t(\phi)$.

A camera image, however, holds much more information, therefore it is reasonable to measure $z_t(\phi)$ with a finer resolution. From now on, it is assumed that the camera points at an angle ϕ horizontally of which it takes images $f_t(\phi)$, while the frequency measurement $z_t(\psi)$ takes place within the picture, and therefore in a slightly different direction $\psi + \phi$.

Because a regular camera has a limited opening angle, a captured image will limit the relative measurement direction ψ to a certain subinterval around the absolute camera direction ϕ:

$$
\psi \in [-FOV/2, +FOV/2] \quad (3.14)
$$

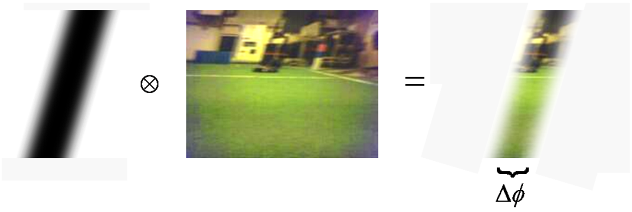


Figure 3.2: The image is scanned in vertical direction for color class transitions at an angular resolution of $\Delta\phi$ by the convolution with a Gaussian stripe.

with FOV being the opening angle of the camera's field-of-view.

Then, the underlying transition detection can be weighted by a Gaussian kernel (with its mean $\mu = \psi$ in the desired sensor direction and a particular width (or resolution) $\sigma = \Delta\phi$:

$$z_t''^{ij}(\psi) = \sum_{<x,y>\in[0,1]^2} transition^{ij}(<x,y>) \cdot \frac{1}{\Delta\phi\sqrt{2\pi}} \exp\left(-\frac{(x-\psi)^2}{2(\Delta\phi)^2}\right) \tag{3.15}$$

expressing the absolute frequency of a color class transition i and j in direction ψ with a certain angular resolution $\Delta\phi$ (see fig. 3.1.5). Subsequently (3.12) can be applied to obtain the relative frequencies $z_t(\psi)$.

Because the Gaussian kernel with variance $\Delta\phi$ blurs the underlying image at a certain resolution $\Delta\phi$, it does not make sense (nor does the robot have time) to evaluate $z_t(\psi)$ in indefinitely many directions $\psi \in [-FOV/2, +FOV/2]$. Rather it is more reasonable to sample the transition frequencies $z_t(\psi)$ from the image at a comparable resolution. Therefore, the sample directions can then be reduced to the set

$$[-FOV/2, +FOV/2] \cap \Delta\phi \cdot \mathbb{N} = \Phi \tag{3.16}$$

This implies that now – per camera image $f_t(\phi)$ – several transition frequencies measurements $\left\{< \psi, z_t(\phi+\psi) > \middle| \psi \in [-FOV/2, +FOV/2] \cap \Delta\phi \cdot \mathbb{N}\right\}$ in different directions ψ exist. Consequently, the robot can evaluate all of them simultaneously after it has taken an image f_t that was captured in an unknown direction. Because the measurements lie substantially far away from each other (namely have a distance of $\Delta\phi$), these measurements can be assumed to be independent, and therefore can be denoted in the following equation:

$$\begin{aligned} p\left(f_t \middle| \phi, m\right) &= p\left(\left\{< \psi, z_t(\phi+\psi) > \middle| \psi \in \Phi\right\} \middle| \phi, m\right) && (3.17) \\ &= \prod_{\psi\in\Phi} \prod_{i,j}^{n} p\left(Z^{ij}(\phi+\psi) = z^{ij}(\phi+\psi) \middle| m^{ij}(\psi)\right) && (3.18) \end{aligned}$$

Additionally, the evaluation complexity of this evaluation of camera images given a map m then depends on the angular resolution $\Delta\phi$ reciprocally: ($p\left(f_t \middle| \phi, m\right) \in O\left(m^2 \cdot n \cdot \Delta\phi^{-1}\right)$).

3.1.6 Map learning

In an unknown environment, the map m is of course not available, and has therefore to be learned by the robot from the measurement series $\left(z_1^{ij}(\psi), z_2^{ij}(\psi), \ldots\right)$. The robot can learn the map by estimating its constituting parameters $m^{ij}(\psi)$. The estimated map and the estimated parameters will be denoted by $\hat{m}$ and $\hat{m}^i j(\phi)$ respectively.

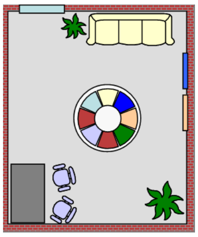


Figure 3.3: Visualization of map learning: a robot learns a map $\hat{m}$ by estimating in each direction ϕ the underlying distribution $Z(\phi)$ from the observed frequency patterns $z_t(\phi)$.

As the measurements $z_t^{ij}(\psi)$ are taken at a certain angular resolution $\Delta\phi$, the map can be estimated at a comparable resolution, and it thereby consists of a finite amount of single-orientation maps $m(\psi)$:

$$m = \left\{ < \psi, m(\psi) > \big| [0, 2\pi] \cap \Delta\phi \cdot \mathbb{N} \right\} \tag{3.19}$$

Fortunately, histogram distributions are particularly easy to estimate. Their parameters $\hat{m}_{(1)}^{ij}(\psi), \hat{m}_{(2)}^{ij}(\psi), \ldots, \hat{m}_{(n)}^{ij}(\psi)$ are direct estimators, as they are actually representing the absolute frequencies of the corresponding bin.

Each time, a measurement $z^{ij}(\psi)$ falls into a certain bin k, its corresponding counter $\hat{m}_{(k)}^{ij}$ of the map can be increased by 1

$$\text{if } 2^{-k} \leq z^{ij}(\psi) < 2^{-(k-1)}, \text{ then } \hat{m}_{(k)}^{ij} \leftarrow \hat{m}_{(k)}^{ij} + 1 \tag{3.20}$$

to update the map (see figure fig. 3.1.6).

To update the map with a complete image $f(\phi)$, the elements $< \psi, z >$ of a taken image $f(\phi)$ can then be used to learn the map $\hat{m}$ by updating the corresponding counters $m_{(k)}^{ij}$.

The evaluation complexity of learning from a single camera image is then of the same complexity class as the evaluation of a camera image given a map $(\hat{m}_t(f_t) \in O\left(m^2 \cdot n \cdot \Delta\phi^{-1}\right)$, see section 3.1.5).

3.2 Localization filter

A localization filter, as defined in def. 3, maintains the robot's beliefs $Bel_t(x)$ about its pose $x =< \phi, \tau >$ (consisting of its rotation ϕ and translation τ). The localization filter updates these beliefs per time step t by applying the sensor and motion model using Bayesian filtering (section 2.3.2).

In the following two subsections, the corresponding motion and sensor models will be constructed. Subsequently, the first part of the localization filter is presented that estimates the robot's heading.

Obviously, a single compass sensor does not yield enough information to additionally estimate the robot's translation. The idea is to use multiple compass sensors: if each sensor is trained at a different spot, the robot can interpolate its (translational) pose by mutually comparing the reported likelihoods.

3.2.1 Motion model

The motion of the robot is simply assumed to be sufficiently be described by the differential data from odometry Δx_t. The motion model (as defined in def. 1 then can be written as:

$$x_t = x_{t-1} + \Delta x_t \tag{3.21}$$

or equivalently, by splitting the pose $x =< \phi, \tau >$ in its rotational ϕ and translational τ components

$$\begin{aligned} < \phi_t, \tau_t > &= < \phi_{t-1}, \tau_t t - 1 > + < \Delta\phi_t, \Delta\tau_t > && (3.22) \\ &= < \phi_{t-1} + \Delta\phi_t, \tau_t t - 1 + \Delta\tau_t > && (3.23) \end{aligned}$$

3.2.2 Sensor model

The posterior probability distribution that a compass sensor (given a map m) estimates when supplied with an image f_t has been stated in (3.24). This distribution can directly be used as a sensor model:

$$p\left(f_t \middle| x_t, m\right) = p\left(f_t \middle| < \phi, \tau >, m\right) = p\left(f_t \middle| \phi, m\right) \tag{3.24}$$

using (3.17). Then the sensor model can express possible ambiguities in a natural way – thanks to the underlying posterior distribution.

The posterior stated in (3.24) forms the sensor model of the visual compass, by relating the captured images f_t to the world state ϕ, given a map m. It can subsequently be used by a localization filter to integrate the likelihoods over time, merge them with data from odometry and eventually select the orientation with the highest likelihood that can be provided to higher-level layers.

3.2.3 Orientational belief filter

The filter then has to maintain the rotational (or compass) beliefs $Bel(\phi)$ of the robot. The belief distribution is represented by a one-dimensional circular grid with an angular resolution of $\Delta\phi'$.

At each time step, the beliefs $Bel_t(\phi)$ are updated according to the motion and sensor model, using the update rules from Bayesian filtering (see (2.8) and (2.8)) and a mixing constant λ:

$$\begin{aligned} Bel_t^-(x) &= Bel_t\left(x - \Delta x_t\right) && (3.25) \\ Bel_t(x) &= \lambda p\left(f_t \middle| x, m\right) Bel_t^-(x) + (1-\lambda) Bel_t^-(x) && (3.26) \end{aligned}$$

This belief distribution may in principle have an arbitrary shape. In order to extract an expedient single rotational estimate that the robot can use (for example for planning and navigation), the belief distribution can be approximated by a Gaussian distribution with its mean in the estimated heading $\hat{\phi}_t$ and a variance $\hat{\sigma}_t$:

$$Bel_t \sim \mathcal{N}\left(\hat{\phi}; \hat{\sigma}\right) \tag{3.27}$$

or, equivalently

$$Bel_t(\phi) = \frac{1}{\hat{\sigma}\sqrt{2\pi}} \exp\left(-\frac{(x-\hat{\phi})^2}{2\hat{\sigma}^2}\right) \tag{3.28}$$

In fig. 3.2.3, a this process is sketched graphically.

The evaluation complexity of both belief updates is reciprocally linear in the angular resolution $\Delta\phi$, so $\left\{Bel_t^-(\phi), Bel_t(\phi)\right\} \subset O\left(\Delta\phi^{-1}\right)$.

The robots can now estimate its current heading by estimating the Gaussian parameters:

$$\begin{aligned} \hat{\phi}_t &= \underset{x}{mean}\, Bel_t^R(\phi) && (3.29) \\ \hat{\sigma}_t &= \underset{x}{var}\, Bel_t^R(\phi) && (3.30) \end{aligned}$$

The orientational belief filter yields per time step t a stable, accurate and odometry-corrected estimate $\hat{\phi}$ of the robot's current heading, associated with an estimate of the variance $\hat{\sigma}$.

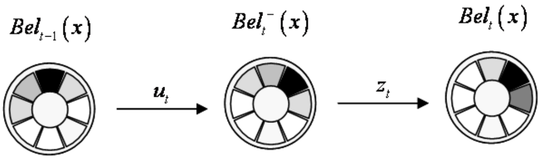


Figure 3.4: Visualization of the Bayesian filtering process. In the first step, the belief distribution $Bel_{t-1}(x)$ is updated according to the motion model (odometry correction). In the second step, the sensor model is applied and a new belief distribution $Bel_t(x)$ becomes available.

3.2.4 Four-spot sensor model

Translational localization is not possible with only a single compass sensor defined by $p\left(f_t|x,m\right)$. However, the shape of this posterior probability distribution can be used as an indicator that correlates reciprocally with the distance to the original training spot.

If now several (say $q = 4$) compass sensors were used (for example trained on the corners of a quadratic area, see fig. 3.2.4), the robot could interpolate its position from the individual likelihoods.

It is assumed that each compass sensor i has its own map $\hat{m}^{(i)}$ learned at a distinct training spot $T^{(i)}$. Then for each compass sensor, the maximal likelihood in the posterior probability distribution can be regarded as an indicator of the goodness of the match (which is inversely related to the distance). A single goodness-of-fit indicator can thus be extracted by extracting the maximum likelihood $L_t^{(i)}$ from corresponding posterior distribution $p\left(f_t|\phi,\hat{m}^{(i)}\right)$:

$$L_t^{(i)} = \max_{\phi} p\left(f_t|\phi,\hat{m}^{(i)}\right) \tag{3.31}$$

From the combined sensor readings z'_t (consisting of the indicators of all q compass sensors)

$$z'_t =< L_t^{(1)}, L_t^{(2)}, \ldots, L_t^{(q)} > \tag{3.32}$$

the translation can be estimated from such a reading directly by computing the weighted sum of the training spots $T^{(1)}, \cdots, T^{(q)}$ according to the corresponding likelihoods $\hat{L}^{(1)}, \cdots, \hat{L}^{(m)}$:

$$\hat{\tau}(z'_t|\hat{m}^{(1)}, \ldots, \hat{m}^{(q)}) = \frac{1}{\sum_i^q \hat{L}_t^i} \sum_i^q \hat{L}_t^i T^i \tag{3.33}$$

The evaluation complexity of the such a combined compass sensor is in addition linear in the number of sensors used $\hat{\tau}(z'_t|\hat{m}^{(1)}, \ldots, \hat{m}^{(q)}) \in O\left(q \cdot m^2 \cdot n \cdot \Delta\phi^{-1}\right)$).

A Bayesian filter can subsequently directly use (3.33) as a translational sensor model in its the sensor belief update step.

3.2.5 Translational belief filter

The translational belief of the robot is expressed by a direct estimate $Bel_t(\tau) = \hat{\tau}_t$.

The filtering can subsequently be implemented in the typical Bayesian fashion (see (2.8) and (2.8)). In each time step, the beliefs are updated first by the motion model (3.21) and the sensor model (3.33),

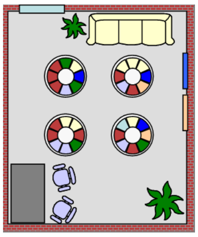


Figure 3.5: Visualization of four compass maps created at the different training spots $T^{(i)}$. Due to projective distortions, each map is slightly different. The translational belief filter exploits this difference in order to estimate the robot's translation τ.

using a mixing constant λ:

$$
\begin{aligned}
Bel_t^-(\tau) &= Bel_t(\tau - \Delta\tau_t) && (3.34)\\
Bel_t(\tau) &= \lambda\hat{\tau}(z_t') + (1-\lambda)Bel_t^-(\tau)\\
&= \lambda\frac{1}{\sum_i^n \hat{L}_t^i}\sum_i^n \hat{L}_t^i T^i + (1-\lambda)Bel_t^-(\tau) && (3.35)
\end{aligned}
$$

Its complexity class is simply linear in the number q of compass sensors used, so $\left\{Bel_t^-(\tau), Bel_t(\tau)\right\} \subset O(q)$.

At each time step t, the translational filter outputs a stable translational estimate $\hat{\tau}_t = Bel_t(\tau)$. Together, the compass filter and localization filter yield per time step t a full estimate of the robots pose:

$$\hat{x}_t = <\hat{\phi}_t, \hat{\tau}_t> \tag{3.36}$$

This estimate can subsequently be used by the robot for planning and navigation.

3.3 Summary

The main contribution of this chapter is the mathematical construction of the **compass sensor** that can provide a mobile robot with **heading likelihoods** for each supplied camera image.

The **compass sensor** has been constructed incrementally: at first, a model $p\left(z^{ij}\middle|\phi, m^{ij}(\phi)\right)$ has been described that applies to a single transition frequency sensor Z^{ij} , by using a map $m^{ij}(\phi)$ of frequency histograms. It has then been extended for a sensor $Z(\phi)$ that simultaneously reports all transition frequencies $Z(\phi)$ between every two color classes i and j. Subsequently, it has been explained how these transition frequencies can be sampled from the robot's camera images $f_t(\phi) < x, y >$: the frequencies can be measured at arbitrary angular resolution $\Delta\phi$ by using a scanning grid $< \Delta x, \Delta y >$. Finally, an algorithm has been devised that a robot can use to learn autonomously new maps $\hat{m}$ in unknown environments. The derived sensor model $p\left(f_t\middle|x_t, m\right)$ can be used for a robot to determine the likelihood that a taken picture $f_t(i)$ originates from a certain heading ϕ.

Two **localization filters** have been introduced that a robot can use to estimate its orientation $\hat{\phi}$ and translation $\hat{\tau}$, respectively (forming together the robot's pose $x =< \phi, \tau >$). As a pre-requisite, a **motion model** has been defined that the filters can use to update their beliefs according to the data from odometry. The **orientational filter** maintains the robots beliefs in form of a circular belief buffer $Bel_t(x)$ at a resolution $\Delta\phi'$. It uses the motion and sensor models to update its beliefs – by applying the update rules of Bayesian belief filters. A mixing constant λ has been introduced that regulates how much of the robot's beliefs are replaced by new sensor readings. The **translational filter** requires four sensors, and uses the maximal likelihood $L_t^{(i)}$ of the individual sensor posteriors $p\left(f_t \middle| \phi, \hat{m}^{(i)}\right)$ to interpolate the robots translational pose $\hat{\tau}$.

The approach falls in the category of **appearance-based** approaches, because the compass sensor only performs low-complexity preprocessing of the camera images. The compass filter has in particular been designed for the use in mobile robots equipped with a **rotatable camera**. No further constraints are imposed on the environment[1]. In particular, the visual compass works **without artificial landmarks**.

---

[1] A weak (and implicit) constraint is that the environment is natural. A clinical-white room would certainly violate the assumption (stated in section 3.1.2) that the colors in a room are never equally distributed.

Chapter 4

Implementation

In this chapter, the implementation of the visual compass approach on a Sony Aibo robot will be described. This chapter will follow the lines of the previous chapter, but then focus on the specific issues inherent to a real robot. Next to the implementation itself, several behaviors will be presented that allow the robot to autonomously calibrate, learn and localize in natural environments.

All parts of a robot have their particular properties, and therefore a thorough hardware and software framework analysis was carried out on the Sony Aibo (see appendix A for the details). Several grave problems like dark corners or camera blur in the images or slipping and inaccuracies in the walking engine were given special attention in the implementation.

Along with the required adaptions necessary for the implementation, the initial choice of parameters will be motivated.

4.1 Compass sensor

The compass sensor converts the incoming color images into the feature space (of transition frequency patterns). This intermediate representation is used both for learning (of the map) and subsequent matching. Finally, it outputs the posterior likelihood distribution $p\left(f_t \middle| \phi, m\right)$.

In order to discretize ϕ, an angular resolution of $\Delta\phi = 4.5°$ was chosen. The resulting discretization error of $\Delta\phi/2 = 2.25°$ then corresponds to the measured angular error $\sigma_\phi = 1.5°$ in the estimate of the camera matrix (see section A.4). This also equals the accuracy of the (classical) landmark-based localization used by the Dutch Aibo Team [49]. This parameter determines in the first place the resolution of the transition sensor (see section 4.1.2). This has downward consequences: it is subsequently sufficient when the map $\hat{m}(\phi)$ is learned at the same resolution, and correspondingly the likelihoods have only to be evaluated at an angular resolution of $\Delta\phi$.

4.1.1 Color classes

In section 3.1.4 the existence of a color-to-colorclass mapping function g is assumed (3.10). This mapping can be created autonomously by the robot, by sampling colors from camera images and clustering those (by using an Expectation-Maximization algorithm based on a mixture of Gaussians) into the most prominent color regions (see fig. 4.1). In the course of this project, an algorithm has been implemented [44] that can create such a mapping within seconds. As it is not in the direct focus of the thesis, its details are presented only in the appendix (appendix B) and have been published elsewhere [44]. This mapping is provided to the robot in the form of a lookup table.

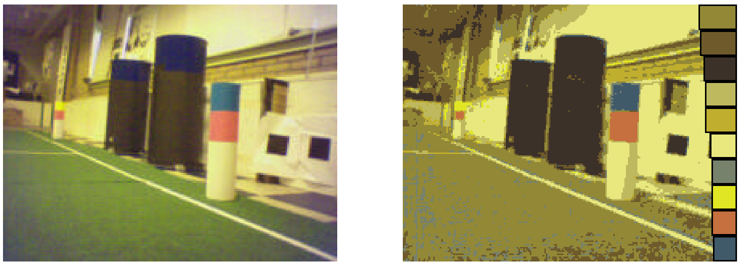


Figure 4.1: Automatic color clustering. **Left:** Original image. **Right:** Resulting color class image. The color classes found by the EM-algorithm are listed at the right border.

The question still remains into how many color classes the color space should be split. As a starting point, it was chosen to use $m = 10$ color classes, corresponding to the original number of (artificial) color classes on the field. As an anticipation of chapter 5, empirical experiments indicated that this number can be reduced significantly, i.e. that the minimum of two color classes can already be sufficient.

4.1.2 Sampling transition patterns from camera images

Splitting the image into these sectors involves a number of geometrical transformations to compensate for some of the weaknesses of the vision system as indicated in appendix A. In particular, as the head is rotatable in three degrees-of-freedom, the horizon is not necessarily horizontal in the images, but may have any slope. Also the position of the horizon may vary, depending on the viewing direction of the head. Moreover, the built in CMOS camera uses a rolling shutter. When the Aibo is scanning with its head, then this results in tilted images (where the horizon is not perpendicular anymore with the up-direction). Lastly, the Aibo camera shows a strong vignetting effect (darker colors in the image corners).

The images have discrete dimensions $< w, h > = < 208, 160 >$, and capture the scene with an opening angle of $FOV = 50°$. Instead of convolving the camera image $f_t < x, y >$ with a Gaussian kernel, it has been decided to simply split it up in vertical sectors of $\Delta\phi = 4.5°$ (see fig. 4.3). A single sector then corresponds a vertical stripe of $\Delta\phi \cdot FOV^{-1} \cdot w \approx 19$pixels. It should be noted that by discretizing the sensor readings in sectors of $\Delta\phi = 4.5°$, the implemented compass sensor has an intrinsic discretization error of approximately half the sector width, or equivalently $\Delta\phi/2 = 2.25°$.

As the images are typically not horizontally aligned, nor is the up-direction perpendicular, it was necessary to implement a scanning grid that implicitly rectifies the image, while scanning it for color class transitions. The scanning grid has a parameter called the scanning resolution $< \Delta x, \Delta y >$. The scanning algorithm projects the horizon onto the image, and then starts to initialize scan lines extending from the horizon in up-direction. Each scanned pixel has to be color corrected (because of vignetting) before it can be mapped on a color class using the pre-computed lookup table. If the scanner then detects a change in color class, it increases the corresponding absolute frequency counted in $z_t^{\prime ij}(\phi)$. Finally, the relative frequencies are computed by division through the sum of scanned pixels, analogously to (3.13). The evaluation of the scanning grid has a complexity that is linear in its dimensions $< w, h >$ and scanning density, or equivalently $< \Delta x, \Delta y >$ ($z_t^{ij}(\phi) \in O\left(xy \cdot (\Delta x \Delta y)^{-1}\right)$.

The scanning resolution has initially been chosen to examine every pixel in horizontal direction and only every $4th$ pixel in vertical direction ($< \Delta x, \Delta y > = < 1, 4 >$), yielding a coverage of 25%. Other resolutions have been tried, and it will be shown in chapter 5 that visual compassing even yields accurate results with a coverage of less than 1%.

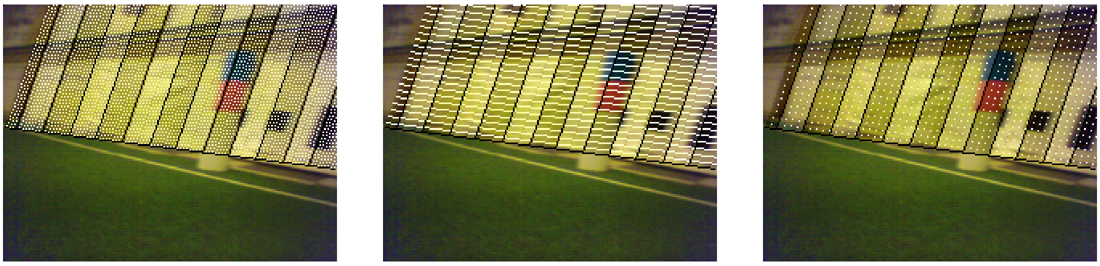


Figure 4.2: Visualization of the scanning grid at different resolutions. The scanning grid already compensates for the current head posture and motion. The scan points are alternately displayed in white and gray, visualizing the underlying (even and uneven) sectors. **Left:** Scanning resolution at 2x2. **Middle:** Scanning resolution at 1x4. **Right:** Scanning resolution at 4x4.

4.1.3 Learning

The map $\hat{m}$ can be learned efficiently by using the update rule (3.20) stated in section 3.1.6. With each incoming image, the distribution estimates of color class transitions become more and more accurate. It is difficult to say how often a single sector has to be seen before the distribution estimate converges, as this depends on the amount of noise present and the complexity of the scene. Anticipating from the results section, the time the robot needs to perceive a sector before it can be recognized robustly is about a few seconds. At a camera frame rate of 30fps, this means that each sector is perceived around 100 times.

A full map $\hat{m}^{ij}_{(k)}$ is then constituted of $360° / \Delta\phi \cdot m^2 \cdot n = 40.000$ integers, with an angular resolution of $\Delta\phi = 4.5°$, $m = 10$ color classes and $n = 5$ logarithmically-scaled histogram bins. It fits easily in a $80KB$ file which the robot stores after successful learning on its memory stick[1].

4.2 Localization filter

Two filters have been successfully implemented: the **compass filter** supplies the robot with a rotational estimate (including a variance estimate), and the **multi-spot localization filter** provides translational

---

[1] `open-r/app/conf/panorama.dat`

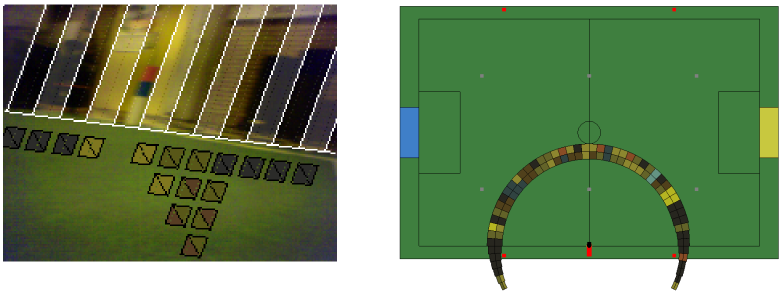


Figure 4.3: The robot divides virtually in sectors of $\Delta\phi = 4.5°$. **Left:** Camera image with sector boundaries superimposed. The dominant color class transitions were displayed below each sector. **Right:** Visualization of a map learned at the UvA Robolab. Per sector, only the most frequent color class transition is displayed.

estimates to the robot.

4.2.1 Orientational filter

The orientational filter models the robot's beliefs in form of a circular belief buffer with an angular resolution of $\Delta\phi'$. This resolution depends on both the resolution of the sensor and the motion model. While the compass sensor operates at a discrete resolution of $\Delta\phi = 4.5°$, the odometry feedback is supplied in a continuous form, but a simple computation can help to determine its magnitude. At moderate speed, the Aibo turns a complete round (360°) in less than $10s$, a differential feedback from odometry of $360°/\left(10s \cdot 30fps\right) \approx 1°$ can be expected per frame. Therefore, an angular resolution of $\Delta\phi' = 1°$ has been chosen for the belief buffer.

Subsequently, the belief update was implemented according to the update equations 3.25 and 3.26 of the motion and sensor model, respectively.

For interactive demonstrations, the parameter λ that regulates how much of the beliefs are replaced by the new sensor information was chosen in such a way that half of the beliefs vanished after $0.3s$ ($(1-\lambda)^{0.3s \cdot 30fps} = 0.5$). For the compass filter experiments in chapter 5 (where a single experiment lasted for $5s$), λ was chosen much smaller, to let the filter accumulate information over a longer period of time.

The localization filter requires per time step only the evaluation of its 360 belief cells, which is performed in practically no time.

4.2.2 Translational filter

The implementation of the multi-spot grid filter primarily aimed at providing a homing skill, meaning that the robot is able to find its way back to the center between the training spots successfully – in order to provide a proof-of-concept that compass sensors can even be used for position localization. In the basic setting, four spots in $1m$ distance around the center were selected for training, and the robot subsequently walked back to the center after a kidnapping.

Due to the evaluation time of the scanning grid and the compass sensor, per frame only a limited amount of compass sensors (or maps) can be evaluated simultaneously. Therefore the current implementation only selects a single training spot $T^{(j)}$ per frame and evaluates the current image f_t given the corresponding map $\hat{m}^{(j)}$. Then the robot cycles through the learned training spots one-by-one and one per frame.

The update rules as stated in (3.35) remain the same; what changes is that only one sensor model $p\left(f_t|\phi, \hat{m}^{(j)}\right)$ is evaluated at a time, yielding a new value $L_t^{(h)}$, and leaving the rest untouched:

$$L_t^{(i)} = \begin{cases} \max \phi p\left(f_t|\phi, \hat{m}^{(i)}\right) & \text{if } i = j \\ L_{t-1}^{(i)} & \text{otherwise} \end{cases} \tag{4.1}$$

The compass sensor thus evaluates each map $\hat{m}^{(j)}$ at $30fps/4 = 7.5fps$. It should be noted, that this has no impact on output frame rate; the translational estimate τ_t is still updated at $30fps$.

An identical to the rotational filter mixing parameter λ was chosen; after $0.3s$, half of the belief was replaced by the sensor update ($(1-\lambda)^{0.3s \cdot 30fps} = 0.5$).

4.3 Behaviors

Several behaviors have been implemented that create, test or use the visual compass. Moreover, a button interface was added that allows the user to quickly activate one of these behaviors (like resetting, calibrating, learning, localizing or demonstrate visual homing).

- The **calibration behavior** activates the preparation phase: then the robot cycles through different camera settings, and selects the one that is best-suited. After that, the robot starts collecting colors and finally creates a color table.
- The **single-spot learning behavior** lets the robot reset its map, and then starts learning by scanning with its head. After 5 seconds, the robot turns by $45°$, then re-estimates its current position by activating the localization behavior for 3 seconds. With the newly estimated pose, the robot iteratively extends the compass map, until it has reached its starting position again. Including walking, the learning behavior takes approximately $90s$ for a single spot. To anticipate the results of an experiment in chapter 5, it turned out that the time needed for learning at each step can be reduced to less than $1s$.
- The **multi-spot learning behavior** lets the robot walk to each of the 4 training spots by pure odometry. There the robot waits for 2 seconds in order to allow the user to correct its pose – which can be off significantly, due to slippage and bad odometry. Then the robot creates a new compass map and starts learning, using the single-spot learning behavior.
- In the **localization behavior**, the robot scans with its head and matches incoming images with one or more compass maps. The localization behavior has been used for most numerical results presented in the next chapter. For an experiment, the robot flushed its belief distribution, the mixing (using the mixing constant λ) was completely disabled (so that nothing is forgotten). An experiment lasted typically $5s$, and after this period the robot stores the estimated heading and variance values in a text file on the memorystick[2]. Optionally, the raw likelihoods of the sensor model can be saved[3].
- In the case of a single available map, the **visual homing behavior** lets the robot active localization and subsequently tries to maintain the former 0°-direction. By a tip on its back, the robot can also be taught to maintain a different heading. In the case that four maps are available, then additionally the translational filter is activated – providing also translational information. The robot then tries to walk back to the center. The visual homing behavior is well-suited for live demonstrations, as it shows intuitively the capabilities of the approach.
- Moreover, **save** and **load behaviors** have been implemented that automatically save the camera setting, color table and compass maps to the memorystick; allowing to restart a robot with the same settings.

4.4 Discussion

The visual compass approach is – when regarded at a higher level – comparable to approaches using scan matching techniques for a laser scanner: typically, the distances on a full circle around the robot are measured in steps of 1^o[23]. Such a laser scan delivers an ordered set of measurements, consisting of the distance to obstacles. One distance measure says nothing about the current location, but the complete set of distance measurements provides a characteristic pattern.

The visual compass makes measurements on a half circle with steps of 4.5^o, with as feature the frequency of color-transitions in that sector. This frequency of color-transitions does not have to be unique for the room, as long as a complete set of patterns (from a whole image) is unique. From a single spot, a sector-based map can be derived that can subsequently be used for localization. When the robot is placed somewhere in the room, it can find the orientation under which the map matches best with the pattern of color-transition frequencies – the orientation with the best correspondence is selected. This consideration indicates that SLAM algorithms that originally were designed to work with laser range scanners could be modified to work (theoretically) also on visual input.

---

[2] `open-r/app/conf/console.txt`

[3] `open-r/app/conf/rawlog.txt`

The visual data from the camera is reduced significantly by a feature space transformation, which makes processing at a full-frame rate possible even on a computationally weakly-equipped robot like the Aibo. Another interesting feature of this approach is that partial panoramas can already be used for localization. The (imprecise) walking engine of the Aibo is therefore already supported by panoramic localization while the robot is turning (and yet learning) on a training spot.

4.5 Summary

In this chapter, the necessary adaptions to make the compass approach work on a real robot (i.e. the Sony Aibo) were worked out:

Compass sensor The continuous approach depicted in the previous chapter was discretized in order to become applicable in an implementation. A few parameters have been chosen as a reference point: an angular resolution of $\Delta\phi = 4.5°$ was chosen, and the number of color classes was assumed to $m = 10$ be a good starting point. Lastly, the scanning grid was chosen to evaluate by default 25% of the camera image.

Localization filters The orientational filter could be implemented without adaptions. Its circular belief grid operates at an angular resolution of $\Delta\phi' = 1°$, and the mixing constant λ was chosen that half of the beliefs are replaced after $0.3s$.

The translational filter uses a grid of 4 training spots to estimate the robot's position. Due to the limited processing power of the Aibo, it was decided to update only a single sensor model $p\left(f_t \middle| \phi, \hat{m}^{(j)}\right)$ per frame, and leaving the others as they were. The filter updates its beliefs with the same mixing rate λ as the orientational filter.

Behaviors Several behaviors have been implemented that allow the robot to calibrate and learn natural environments autonomously. The learning behavior allows the robot to learn a map in 8 steps of $45°$. Per default, the learning time in each direction is $5s$. Additionally, two localization behaviors were created that can be used to measure and demonstrate the capabilities of the visual compass approach. A simple button interface was implemented that allows the user to activate the most important functions.

Chapter 5

Results

In this chapter, the performance of the visual compass approach (as implemented in chapter 4) is analyzed in a series of empirical experiments.

In the first part, the compass functionality will be validated qualitatively in natural environments. Additionally, the experiences gathered from public demonstrations will shortly be summarized.

Subsequently in the second part, the quantitative performance of the compass approach will be determined in a series of lab experiments: the impact of distance on the compass variance will be measured, and the robustness with respect to controlled illumination changes is evaluated. Subsequently, the parameter choices of chapter 4 will be evaluated: the operational range is explored by adapting the parameter until the compass sensor fails.

In the third part, the performance of the translational filter will be assessed qualitatively and quantitatively in a visual homing experiment.

Finally in the fourth part, the effective processing time of compass localization will be measured empirically on the Sony Aibo.

5.1 Experimental setup

For the empirical experiments, the test behaviors introduced in section 4.3 have been used. The robot therefore chose a camera setting and created a color table by clustering. Then it learned autonomously the appearance of a room, incrementally in 8 steps of $45°$, for 5 seconds per direction. After each step, the robot used the partial map to re-estimate its heading in order to reduce the motion noise while turning to a minimum[1].

The conducted experiments can be activated on the Aibo through its back buttons, and the result of an experiment is mostly directly visible through the reaction of the robot (by rotating or moving itself into the target pose). Most of the experiments also produced quantitative output (like rotational estimates, likelihoods and computation times), that was sent via the wireless network to a connected computer or optionally was written to the robot's memory stick. Storing the experimental data on the memory stick allowed an easy setup for the field experiments.

The rather raw textual output of the robot has then been processed interactively by different programs to visualize the results in an intuitive way.

---

[1] In the earlier experiments (section 5.2), the robot was taught its environment by being aligned manually in 4 different directions (heading north, west, south, east). For each orientation, the robot learned the panorama by turning its head for 10 seconds. All later experiments (section 5.2-5.6) have been conducted using autonomous map learning.

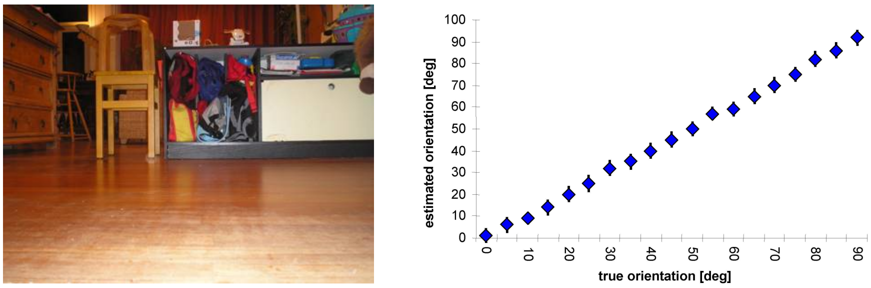


Figure 5.1: Experiment conducted in a ordinary, but brightly lit living room at a private home. The robot was rotated manually in steps of 5°. The robot's heading estimates have been plotted against the true orientation.

5.2 Orientational filter: qualitative results

The orientational filter aggregates the output of the vision module in a natural way. The orientational filter accumulates the orientational likelihoods in a buffer consisting of 360 cells, each cell corresponding to 1°. In the original implementation for demonstration purposes, the filter slowly replaces old beliefs by new readings. This has been disabled for the experiments: the buffer is flushed when a new measurement starts, and all subsequent estimates are added in the same proportion. After the measurement interval of 5 seconds is over, the content of the buffer is evaluated as described in section 4.2.1: the belief distribution is approximated by a Gaussian, yielding both a heading and variance estimate.

Each of the following subsections corresponds to a single environment. First the environment itself will be described, including an account of the particular difficulties. Then the results will be presented and (sometimes) the found curiosities will be pointed out. Finally, the found results will be compared to what in theory could have been expected, and a brief interpretation will be given.

5.2.1 Home environment: Living room

The first presented experiment was conducted in an ordinary, but brightly lit living room at a private home. It is ordinary insofar as it contains a variety of colorful objects at different locations. The most central spot was chosen for map learning, and this had a mean distance of two meters to the surrounding furniture. Although the room was exceptionally bright lit, the robot configured its camera in the slow shutter time setting, which probably resulted in somewhat blurred images. In the course of the experiment, the robot was rotated manually in steps of 5 degree and its estimated rotation was recorded.

In fig. 5.2.1, the error in the estimate is plotted against the true orientation: the robot is never more off than 2° and the standard variance is around 7°, which is close to the maximal theoretical resolution $\Delta\phi/2 = 2.25^\circ$ of the compass sensor (see section 4.1.2).

5.2.2 Office environment: DECIS Lab

The second experiment was conducted on the first floor in the DECIS Lab, in a typical office environment. Big windows to both sides bring much daylight to the inside and let the room appear very bright. A long corridor passes through the open-plan office, with small working islands both to the left and right. Each working island typically accommodates four work places, one to each side.

The robot was put on the corridor on a relatively central spot, so that it could be moved along the corridor for approximately 7 meters in both directions. The experiment was conducted in the late morning

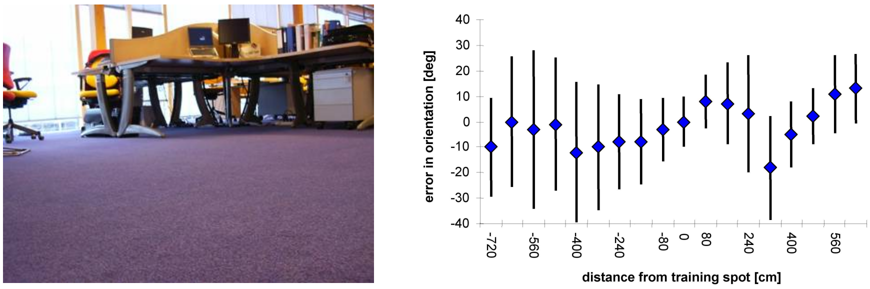


Figure 5.2: The first floor of the DECIS Lab in Delft has typical properties of an office environment. The robot has been moved along a $14m$ long corridor passing working islands on both side. While to the left side (towards the lift), the variance increases steadily, an interesting phenomenon was observed to the right side: the repetitive patterns of the working islands lead to a decrease of the variance after approximately $5m$ – when the robot has reached the next working island.

with the sun shining brightly.

The results have been depicted in fig. 5.2. The rotational error was lower than $10°$ for most measurements. At least for the left side of the diagram, the size of the variance is slowly increasing the further the robot is moved away. However, to the other side of the corridor, the interval increases slightly, but then almost reaches its original value.

The reason for this decrease of uncertainty is interesting: the panorama to the right side periodically looks the same, due to the repetitive patterns of the similarly-looking working islands. So while at first the panorama becomes slightly distorted as the robot is moved on, the panorama repeats after around 5 meters, where the robot has reached the next working islands. This in turn leads to an decreased variance in the rotational estimate.

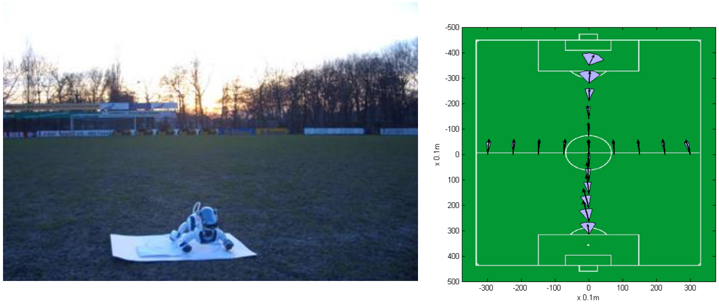


Figure 5.3: Experiment conducted at the Zeeburgia outdoor soccer field in Watergraafsmeer, Amsterdam. The robot was moved across the whole field on its primary axes. The arrows display the reported heading, while the blue arcs around the arrow visualize the associated variance. The heading estimates point nicely in direction of the opponent goal.

5.2.3 Outdoor soccer environment

The third experiment was performed outside on a real soccer field. The soccer field has a size of approximately $40m$ x $60m$, and has like a typical soccer field a colorful side fence with advertisements. At

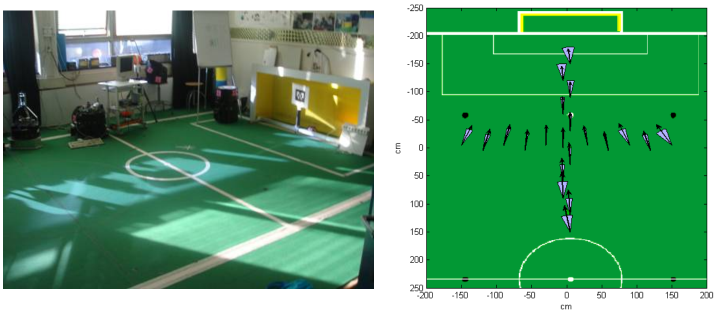


Figure 5.4: Indoor experiment conducted at the Midsize field of the Technical University of Delft. The results are comparable to the outdoors experiment, but have a much lower variance.

some places, grandstands with seats are installed. Around the field some trees (without leaves, because conducted in the winter) are visible. Above and behind the trees, also the sky and some clouds can be seen.

The robot was trained on the center spot, and then was moved along the axes both left/right and forward/backward across the field. The robot was adjusted at all measurements with the same heading on visual judgment.

The results are displayed in fig. 5.3. The heading estimate points as expected into the direction of the opponent goal, while the reported variance is increasing the further the robot is moved away from the center.

The interesting point about this is that in elongation the arrows meet in a point maybe 30 meters behind the opponent goal. This actually corresponds to the line of trees behind the field. Such an effect could certainly be exploited for pose triangulation (as described in appendix C).

5.2.4 Indoor Midsize field

The fourth experiment was conducted on the Midsize field in the Robolab of the Technical University of Delft (see fig. 5.4).

The heading estimates point nicely in the direction of the opponent's goal. The variance is much lower than in the latter outdoor experiment. Again the arrows meet in a point approximately $1m$ behind the goal, which could be exploited for pose triangulation.

5.2.5 Indoor 4-Legged field

The fifth experiment of this type was conducted in the Robolab of the Universiteit van Amsterdam. It is situated in a small room of about 7x7 meters. On one side of the room, there are tables, chairs and computers. The field itself has a size of 6x4 meters and therefore occupies the major surface of the room.

In principle, the experiment was conducted in the same way as the previous two. The robot was trained in the center of the rear field half, and then the robot was moved across the field and its rotational estimates were recorded. However, instead of only measuring along the axes, the robot was moved across the complete rear half of the field ($3m$ x $3m$) at a resolution of approximately $33cm$.

The results are displayed in fig. 5.5. All heading estimates point towards the opponent goal. Moreover, the variances increase the further the robot gets away from the training spot.

Again the imaginary intersection point of all arrows meet in a point on the wall, slightly behind the opponent goal. This corresponds to what can be expected from the previous two experiments.

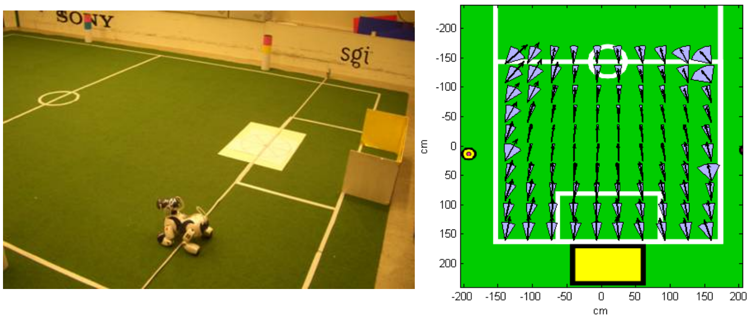


Figure 5.5: Exhaustive compass experiment conducted at the UvA Robolab. The heading estimates point towards the opponent goal. Close to the training spot, the variance is relatively small, but increases steadily the further the robot is moved away.

5.2.6 Demonstrations

Demonstrations were given in our Robolab (for example to the Dutch winners of the RoboCup junior participants or at the Open Dag), demonstrations of panoramic localization were given at international conventions such as the IK 2006[2] and at the RoboCup Open Challenge 2006. From these demonstrations, much additional knowledge was gained about the robustness of the approach.

At the Open Challenge of the 4-Legged League at the RoboCup 2006 competitions in Bremen, the complete approach was demonstrated within three minutes time: the robot learned autonomously the panorama of the field. After that, the robot was repeatedly kidnapped, and had to restore its original heading. Then all landmarks were removed, to demonstrate that the visual compass was independent of the artificial landmarks. This proved that the compass approach was usable on a 4-Legged field under the official circumstances as found in 4-Legged soccer competitions. This was recognized by the jury (constituted by the team leaders), and so our approach was rated 4th out of 25 participating teams (by which we finally won the 3rd prize of the technical challenges).

Two videos are also available on the internet[3]. The first one has been recorded in the UvA Library, and shows the orientational filter. The robot has been trained on one spot of that room in advance, and the visual homing behavior demonstrates that the robot is able to restore and maintain a certain heading (after being kidnapped). It is also shown, that the compass does not only work on the training spot, but also on other places. Furthermore, the robot is taught to orientate in a different directions, that it subsequently tries to maintain. The second video has been recorded in the 4-Legged Robolab, and shows the functioning of visual homing with four training spots. The robot is kidnapped repeatedly and finds confidently its way back to the center. It is also demonstrated that the robot is independent of the soccer landmarks around the field: first, the landmarks are removed, and later put back in completely wrong places – with the robot walking confidently back to the center.

5.3 Orientational filter: quantitative results w.r.t. variables

In the previous section, the results of the compass experiments were presented and discussed in qualitative way. The focus of this section is to assess quantitative measures that describe the performance of the compass approach with respect to certain variables: the effect of the distance of the robot to its training

---

[2] Interdisciplinary College, an annual one-week spring school on Artificial Intelligence held in Günne, Germany (`http://www.ik2006.de`)

[3] `http://student.science.uva.nl/~jsturm/resources.html`

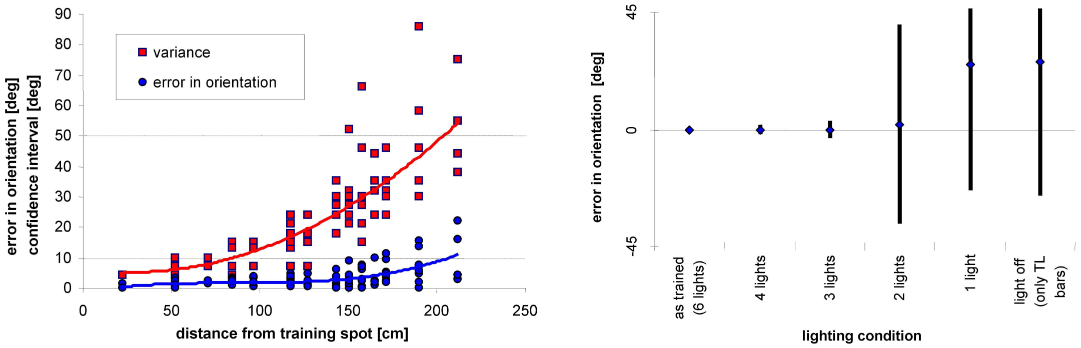


Figure 5.6: Quantitative results of orientation filter experiments. **Left:** True error and variance plotted versus the distance from the training spot. **Right:** The robustness against environment changes has been tested by switching of so many ceiling spots until compass localization fails.

spot will be investigated, and the influence of controlled illumination changes will be measured. All of these experiments were conducted at the UvA Robolab.

5.3.1 Variance w.r.t. distance

The results of the compass experiment (presented in section 5.2.5) are displayed in fig. 5.6(left) in numerical representation: the average of the true error in rotational estimate indeed only increases slightly, and exceeds the 10° mark only after the robot is more than $2m$ away from its training spot. The average variance crosses approximately at this point the 50° mark.

5.3.2 Variance w.r.t. changing environmental conditions

To investigate the robustness of the approach against environmental changes, the spotlights were turned off one by one after training to see when and how the compass localization would fail.

The results are presented in fig. 5.6(right): under training circumstances, both the rotational estimate are as exact as they can be, and the variance is rather sharp. Switching off some of the spotlights increases the variance again only insignificantly. After two third of the lamps have been switched off, the variance estimates suddenly increase to around 90 degrees, which should probably be interpreted as a mismatch, because turning off even more lights (until complete darkness) does not make this result any worse.

In theory, it is clear that the algorithm is able to handle a certain proportion of noise. Illumination changes until a certain degree can then also considered as uniform noise, with no direct influence on the variance – as all directions are affected in the same way. At a certain point, this noise outbalances, and localization is not possible anymore.

5.4 Orientational filter: quantitative results w.r.t. parameters

In this section, the influence of the algorithm's parameters on both the performance as well as the computation time are investigated. The parameters will be adapted so far that the compass approach is brought to its limits: the scanning resolution $< \Delta x, \Delta y >$ will be reduced, the number of color classes m will be lowered and the training time will be shortened as far as possible. The corresponding experiments have been carried out in the UvA Robolab.

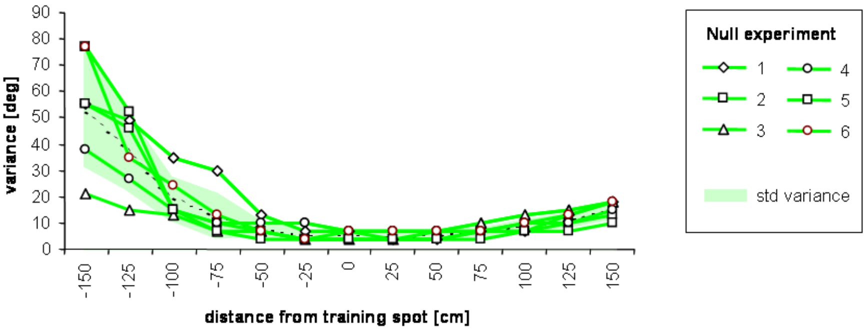


Figure 5.7: Null hypothesis: the green lines show the measurement curves of the the experiment with unchanged parameters. For each position, the mean and variance were computed and visualized in the diagram by the green area.

In each of the following experiments, the robot created autonomously a map at the center of the field and was then moved along the center line for $3m$ in $25cm$ steps. Each single experiment thus consisted of 13 heading and variance measurements. As the heading was already found to be very stable (see fig. 5.6), in the subsequent experiments the variance is visualized as it gives a good impression of the orientational filter's certainty.

5.4.1 Null hypothesis

As every time a slightly different color table is created and the generated maps rely on this color table, the maps do not even approximate the same underlying distributions. Therefore, a null experiment was carried out to determine the natural variance of the probabilistic compass approach.

The same experiment was repeated 3 times with 2 different robots (see fig. 5.4.1), yielding 6 measurement curves. Of these curves, the mean and variance was computed and visualized in the image. All curves look rather similar: variance is lowest at the training spot at the center (around $5°$), and monotonously increasing to both ends. The variance increases much faster to the left side ($\approx 50°$ at $x = -1.50m$) than to the right side ($\approx 15°$ at $x = +1.50m$). The reason for this is presumably that in the Robolab to the right side only a white wall is seen by the robot (yielding good matches resulting in low variance) while to the left side tables and chairs seemingly disturb the trained panorama the closer the robot comes.

5.4.2 Variance w.r.t. number of colors

Given this null hypothesis, the influence of the number of color classes m on the variance was evaluated.

From a theoretical point of view, the choice of the number of color classes m has several effects: first of all, the computation times of many parts of the algorithm strongly depend on this number. Secondly, the more color classes are used, the more information the model contains, and therefore the bigger the map gets.

Many experiments with different numbers of color classes were run, but none of them let the compass filter fail to find the right orientation. Therefore, only the results of the smallest possible number of color classes ($m = 2$) is depicted in fig. 5.4.2(top). The measured variances however show certainly that the reduction of the color classes has an impact on the resulting certainty.

These results are rather unexpected and seriously call into question both the initial assumption that

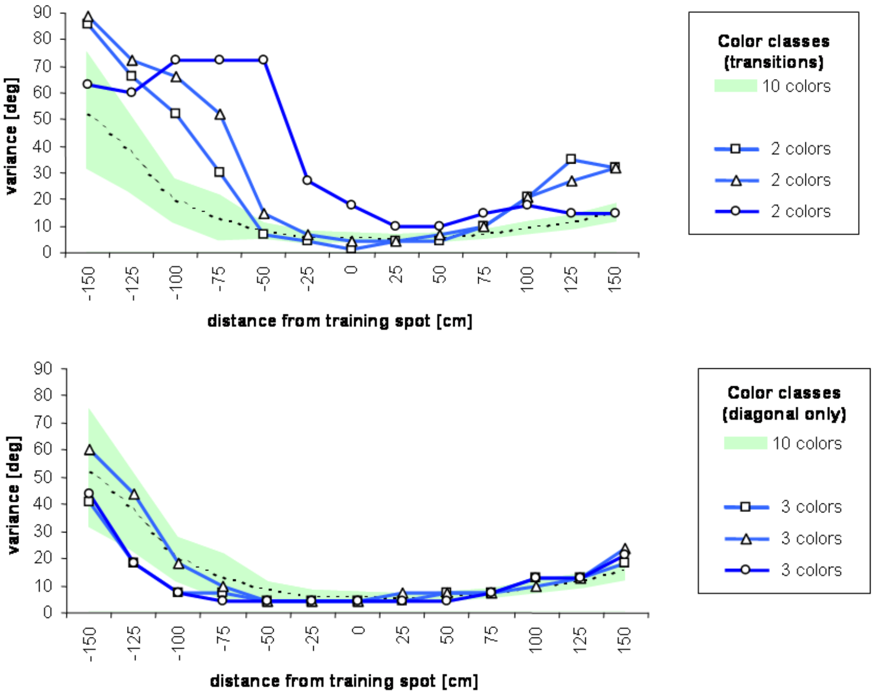


Figure 5.8: Experiments on effect of the number of color classes. **Top:** The number of color classes was reduced to smallest possible number $m = 2$. Although the measurement curves look significantly different w.r.t. the null hypothesis, the compass filter successfully found the correct headings. **Bottom:** To reduce the sensor model further, only the diagonal of the transition pattern $z^{ii}(\phi)$ (and the corresponding diagonal of the map $\hat{m}^{ii}$) was evaluated. The smallest number of color classes for successful operation of the visual compass was then found to be $m = 3$.

several color classes are needed and that the emphasis of the feature space should be placed onto the color class transitions. Rather, these results indicate that the feature space might have been chosen too big, and therefore raise the question to how much it can be reduced.

In order to reduce the feature space even further, instead of all transitions patterns $z^{ij}(\phi)$ only the frequencies of the individual color classes $z^{ii}(\phi)$ can be used (which corresponds to the diagonal of the transition pattern $z(\phi)$). The feature space is thereby reduced by the factor m.

Accordingly, a new sensor model only evaluates the diagonal of the transition pattern $z^{ii}(\phi)$ and the diagonal of the map $\hat{m}^{ii}(\phi)$.

With this modified sensor model, the experiment was repeated. This time, compass localization failed with only $m = 2$ color classes. However, robust localization was already achieved with $m = 3$ color classes. The corresponding measurements are presented in Fig. 5.4.2(bottom).

These results are quite amazing and show that the visual compass approach can be scaled down seriously: the reduction of the number of color classes by a factor of 5 results in a reduction of the computational time for learning and sensor model evaluation by factor $5^2 = 25$ (as $p\left(z \middle| m(\phi)\right) \in O\left(m^2 \cdot n\right)$, see section 3.1.3).

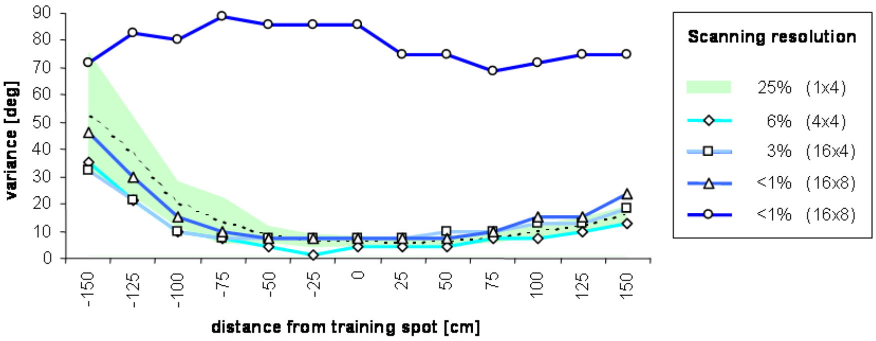


Figure 5.9: In this experiment, the effect of different scanning grid resolutions $< \Delta x, \Delta y >$ was evaluated. From these results, it can be concluded that the original scanning resolution of $< 1, 4 >$ can safely be increased by factor 16 – corresponding to the evaluation of only 40 pixels per sector.

5.4.3 Variance w.r.t. scanning resolution

Correspondingly, the effect of the scanning resolution on the variance was evaluated. The scanning resolution determines how much of the raw image is actually scanned in order to create the feature vectors. The initial chosen parameter $< \Delta x, \Delta y > = < 1, 4 >$ scan of each vertical line every 4^{th} pixel, resulting in a 25% coverage.

Interestingly, the scanning distance could be enlarged significantly before a change in the variance measurements became visible (see fig. 5.9). Increasing the scanning resolution by factor 2, 4, 8 and 16 were without impairment of the heading and variance estimates. Compass localization failed for the first time at a scanning resolution of $< \Delta x, \Delta y > = < 16, 8 >$.

This means that the scanning grid could – if desired – safely be scaled down by factor 16, speeding up the grid evaluation by the same factor ($z_t^{ij}(\phi) \in O\left(xy \cdot (\Delta x \Delta y)^{-1}\right)$, see section 4.1.2). This proves that the evaluation of only 40 pixels per sector is sufficient for the visual compass approach to work.

5.4.4 Variance w.r.t. learning time

Finally, the learning time (of originally $5s$) was reduced to find the lower bound. Surprisingly, it turned out that the learning time was of lesser influence than expected: even after it has been reduced by 80% (or equivalently, to $1s$), no significant change in the measurements could be detected (see fig. 5.4.4). Learning times lower than $1s$ are not reasonable, because a single head scan of the robot approximately takes $1s$. The robot then perceives every direction of the scene approximately approximately $50° / 230° \cdot 30 fps = 6.5$ times per step and iteratively in total $50° / 360° \cdot 30 fps = 33$ times until learning is complete. With a different learning behavior, it could even be possible to scale the (total) learning time down even further.

5.5 Orientational filter: computation time

Finally, the relationship between various parameters and the actual computation time of the implementation was measured. The two major processing steps (transition pattern sampling and sensor model evaluation) were evaluated separately.

First, the runtime of the transition pattern sampling $z_t^{ij}(\phi)$ from the image has been profiled. Fig. 5.11 shows that the processing time needed depends almost linearly on the number of scanned pixels: using

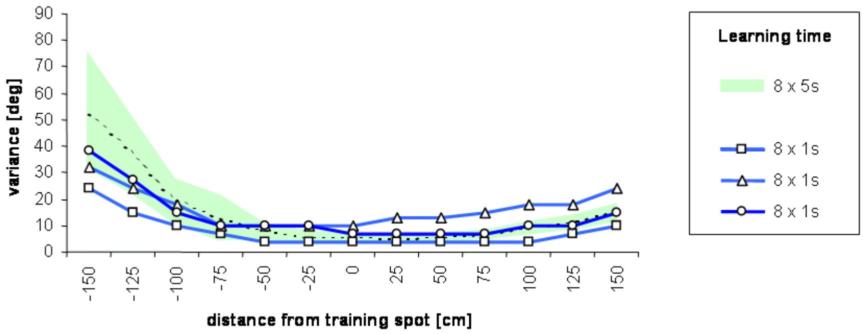


Figure 5.10: The learning time of originally $5s$ per step was reduced by 80% to $1s$. The resulting variance curve shows no difference compared to the null hypothesis.

all pixels for the conversion takes around 8ms, while in the default setting (25%), only $3ms$ are spent to analyze a single image. Much weaker, the feature space conversion also depends on the number of color classes: the size of the transition pattern depends (quadratic) on the number of color classes, and therefore takes longer for the robot to access (while clearing and filling). These results correspond to the theoretical expectations ($z_t^{ij}(\phi) \in O\left(xy \cdot (\Delta x \Delta y)^{-1}\right)$, see section 4.1.2).

Then, the time needed for learning the map was experimentally determined. The measured runtimes are shown in fig. 5.12. It can be seen that the processing time develops quadratic with the number of color classes. This is also in line with the theoretical expectations (as $\hat{m}_t(f_t) \in O\left(m^2 \cdot n \cdot \Delta\phi^{-1}\right)$, see section 3.1.6).

Lastly, the processing time was measured for sensor model evaluation (see also fig. 5.12). The red curve shows the resulting processing time when the sensor model as defined in (3.24) is fully evaluated. The time needed for evaluation depends then quadratically on the number of colors ($\Delta\phi$: $(p\left(f_t \middle| \phi, m\right) \in O\left(m^2 \cdot n \cdot \Delta\phi^{-1}\right))$, see section 3.1.5). However, a huge number of color class transitions is relatively unlikely and therefore can be evaluated in the sensor model faster by partial map pre-caching [4]. The

---

[4] These zero-distributions (e.g., distributions with $\sum_{k=1}^{n-1} m_{(k)}^{ij} \ll m_{(n)}^{ij}$) can be pre-cached after learning by a mark in a bit mask, and subsequently be left out in the evaluation of (3.24) – when the sensor measurement also reports the most infrequent bin

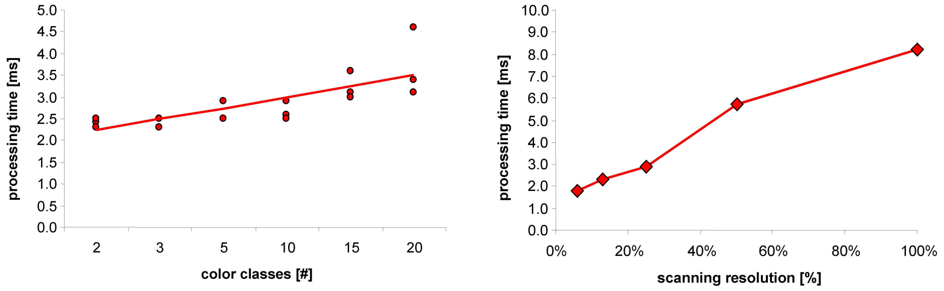


Figure 5.11: Measured computation times of the transition pattern sampling, with respect to the number of color classes m **left image**) and the scanning resolution $< \Delta x, \Delta y >$ (**right image**).

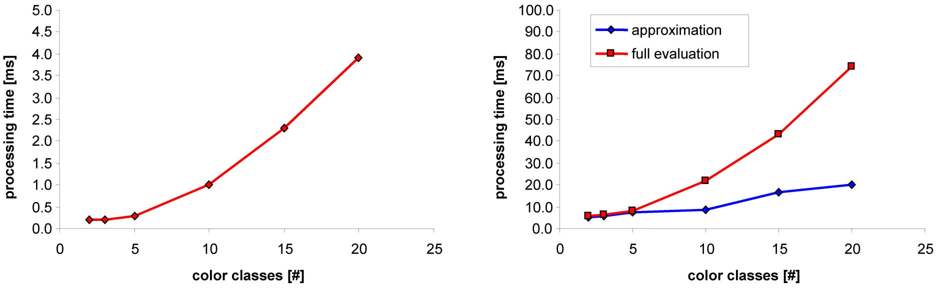


Figure 5.12: Measured computation times of map learning $\hat{m}(\phi)$ (**left image**) and sensor model evaluation $p\left(z \middle| m(\phi)\right)$ (**right image**).

resulting processing time after this approximation is indicated by the blue line: it can be seen that the processing time then only rises linearly in the number of colors, which constitutes a great saving.

The computational time needed for the orientational filter turned out to take no time (below $0.1ms$), and is therefore not interesting.

In sum, the total processing time with default parameters averages to about $5ms$ for learning and $11ms$ for matching a frame on a standard Sony Aibo. This is fast when compared to the original image processing module of the Dutch Aibo Team which took around $17ms$ per frame, and the particle filter which consumed another $6ms$ when using the same hardware.

5.6 Translational filter

Finally, experiments were conducted to examine the functionality and performance of the translational filter.

A visual homing experiment was carried out on the soccer field of our Robolab. The kickoff circle was chosen as the target spot, and the robot learned the panorama of four orthogonal spots around the center in the distance of 1 meter autonomously. Although the robot walked to the training spots using the latest map as in its orientational filter to support odometry, the robot sometimes got a bit astray and then had to be aligned manually, before the training on the next spot started. Finally, the robot walked back to the target spot in order to perform a normalization of the individual likelihoods (so that unity likelihoods $L_t^{(i)} = 1$ were reported from all compass sensors at the target spot).

After learning was complete, the robot was kidnapped and placed somewhere randomly on the field (with random orientation). From there, the robot had to walk back to the center using both the orientational and translational filter, and the position τ_t where the robot stopped (for longer than 5 seconds) was marked with a sticker on the carpet. In total, 33 kidnappings have been executed and the reached positions have been measured.

5.6.1 Qualitative results

The reached positions τ_t are plotted in fig. 5.13(left).

---

n $(z^{ij}(\phi) \leq 2^{-(n-1)})$. After evaluation of the product, the c omitted factors can be multiplied as a single approximated factor $2^{-(n-1)c}$ to the intermediate sensor posterior. The interested reader can inspect this approximation in the DoxyGen documentation or directly in the source code: `http://www.science.uva.nl/~jsturm/resources.html`

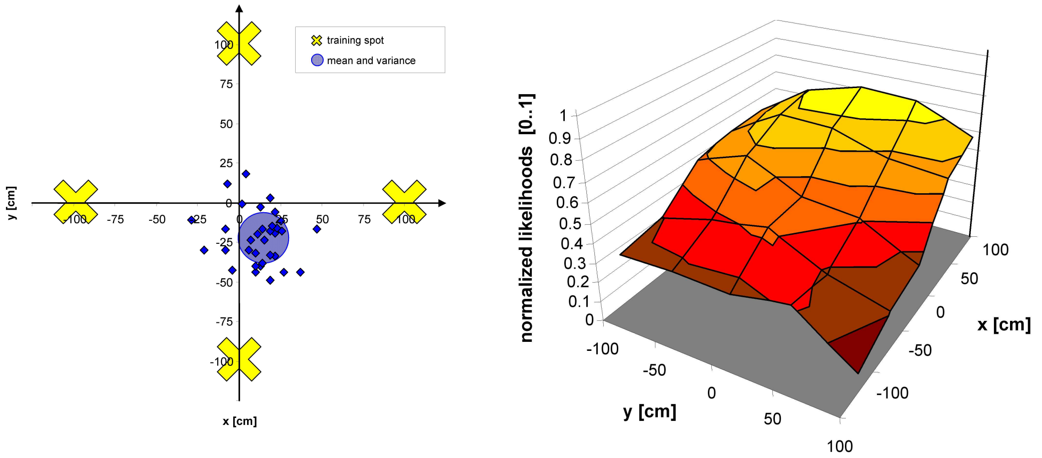


Figure 5.13: Experiments of translational filtering. **Left:** Measured positions in a visual homing experiment. The four training spots are marked with a yellow cross. The blue dots correspond to the positions where the robot stopped. **Right:** Reported likelihoods $L_t^{(i)}\left(f_t\middle|m(i)\right)$ when the robot is moved across a grid with a resolution of $50cm$. The robot was trained at a spot $T = \left(100cm \quad 0cm\right)^T$.

In most cases the robot stopped close to the target spot. However, it can also be seen that the robot walked (in average) to a spot slightly behind and right to the center spot. Finally, because not visible in the graph, it has to be stated that the robot never left the field or stopped on a point outside the displayed area.

5.6.2 Quantitative results

The measured positions τ_t of this experiment can also be expressed numerically, by computing their statistical mean and variance. The average position where the robot stopped can then be expressed as $mean\,\tau_t = \begin{pmatrix} -22cm & +12cm \end{pmatrix}^T$ with a variance of $var\,\tau_t = \begin{pmatrix} 17cm & 15cm \end{pmatrix}^T$

A standard derivation in this magnitude is acceptable when compared to other techniques. The Dutch Aibo Team [49] for example reports a standard derivation of $13.5cm$. It should be noted however, that in the current state of the translational filter, visual homing is only possible to a single (central) spot, while other localization approaches directly yield a positional estimate for each position of the field.

In order to be able to get more than a qualitative estimate on the current position even for other spots than the center, it would be necessary to gain more insight in the kind of shape of the likelihoods when the robot is moved away from the training spot (see fig. 5.13(right)).

It remains a problem to interpret the likelihoods more precisely in order to deduce the robot's absolute position. In the figure, a monotonic decrease in all directions exists. However, it is also visible that this decrease in the likelihoods does not seem to have a linear shape. Therefore, in order to achieve full translational localization it would be necessary to find ways that link the likelihoods more accurately to the robot's distance.

The systematic derivation to the bottom right can be interpreted in such a way that the likelihoods of the individual sensor models (or maps) do not decrease equally fast; the likelihood normalization on only a single spot does not seem to be sufficient.

Considering these results, it can be concluded that the visual compass approach can in principle be used for translational localization. Several aspects remain for further research.

5.7 Summary

In this chapter, the empirical experiments conducted with the visual compass approach have been described and the measured results have been analyzed. Thereby, a good overview of the characteristics of the approach could be established:

- The **qualitative results** prove that the visual compass approach can supply a mobile robot with robust heading estimates. The experiments have been conducted in a broad spectrum of natural environments (like a living room, a office in daylight and on an outdoor soccer field). Moreover, of the given demonstrations a certain robustness can even be asserted to the visual compass approach in dynamic environments.
- Subsequently **quantitative results** were presented from experiments conducted in the Robolab. The heading estimates showed an accuracy better than $12° \pm 5°$, even after the robot was moved away for more than $2m$ from the original training spot. Secondly, the influence of illumination changes was examined under controlled circumstances. The ceiling spotlights were turned off one-by-one, and only after more than half of the spotlights were switched off, the compass approach began to fail (indicated by an increased variance).
- Then the **parameter choices** of chapter 4 were evaluated. First, a null experiment was conducted to determine the natural variance of the experiment under unchanged conditions. Subsequently, each parameter was changed so far that the compass approach was brought to its limits. Thereby, surprising insights were gained: the number of color classes could be seriously reduced. Additionally, a reduction of the coverage by factor 16 still yielded comparable results. Finally, it was shown that the learning time can also significantly be reduced: the original $5s$ were stepwise reduced to the practical lower bound of $1s$, and no impact on the variance estimate could be detected.
- The measurements of the required **computation times** prove the visual compass approach to be efficient. It runs (with the initial parameter) set easily in real-time on the moderately-equipped Sony Aibo. The parameters allow the algorithm to be scaled in many directions: so the angular resolution could be increased if a better camera would be available. Additionally, the empirical parameter evaluation shows that the computational effort can be reduced in the order of one magnitude.
- Lastly, the translational filter (using 4 training spots at a distance of $1m$ from the center spot) was tested in a visual homing experiment. The robot could return in all cases back to the center.

Chapter 6

Conclusions

In this chapter, the content of the thesis will be summarized and discussed. In particular, this means that the research question will be revisited and the actually achieved results will be presented in short propositions along with references to the experiments that validated these propositions. Subsequently, it will be discussed to which extent my approach can contribute to specific problems in particular and to the field of mobile robotics in general. After that, the achievements will be put into perspective to other solutions of related work to point out the novel aspects. Finally, several interesting directions for future research based on my own research will be motivated.

6.1 The answer to the research question

As stated in the thesis goals in section 1.4, this Master's thesis project was intended to find an alternative approach for mobile robot localization particularly applicable in natural environments. The approach was designed for robots equipped with a (scanning) camera as their main sensor and that have only moderate hardware resources at their disposal. Additional emphasis was put on the demand that the robot should be able to learn quickly about new and previously unknown environments and to do so as autonomously as possible.

In what follows, the most important features of my approach will be summarized in short propositions, and will be subsequently discussed and related to the validating experiments.

1. The visual compass approach can supply a mobile robot with accurate heading information.
2. Visual compassing is applicable in a vast variety of different environments.
3. The reported heading information is robust to environmental changes.
4. Robots using visual compassing can learn quickly (and almost autonomously) to orientate in new environments.
5. Multi-compass localization can provide (absolute) positional information.
6. The visual compass algorithm is efficient and economic; it can be executed in real-time on-board a mobile robot, and even works with low-quality cameras.

Proposition one states that the visual compass approach provides accurate heading information. This is proven in a two-fold way: on the training spot itself, the algorithm reaches its optimal angular resolution (living room experiment, section 5.2.1) corresponding to an error of less than 2° (or equivalently, half of the angular resolution parameter $\Delta\phi$). This error subsequently increases only very slightly when the robot is moved away from the training spot (section 5.3.1): the average rotational error remains below 10° even after the robot has been displaced further than $2m$ away from the training spot. Along with the rotational estimate, a confidence interval is computed, which is an indicator for the uncertainty (or the level of mismatch). It increases monotonously the further the robot moves away from its training spot. Finally, compass information is valuable for a mobile robot; a soccer playing robot could easily shoot the ball towards the opponent's goal simply by using the compass estimates as depicted in the magnetic field diagrams (section 5.2.3, 5.2.4 and 5.2.5).

Proposition two is validated by the fact that the visual compass experiments were conducted in a variety of different environments, like outdoors on a soccer field, or indoors in human environments like at home or at an office with either daylight from outside or clean laboratory conditions like in our Robolab. The measured data of these extraordinarily different locations still yielded comparable results (as far as a soccer field of $4m$ width is comparable to an office corridor of $14m$ length or an outdoor soccer field of $60m$ width; see section section 5.2.2, 5.2.3 and 5.2.5).

The approach's robustness as claimed in proposition three has been proven in a two-fold way. On the one hand, the daylight/outdoor experiments (section section 5.2.2 and 5.2.3) showed that the measurements of the visual compass were robust enough to cope easily with (smaller) changes in illumination due to a constantly changing angle of incidence of the sunlight and/or moving clouds. On the other hand, the robustness experiment conducted at the Robolab (section 5.3.2) certainly showed that turning off some of the lighting or changing the environment by removing the beacons was not of much influence on the compass accuracy (and its associated confidence).

All of the quantitative results of the visual compass experiments indicated a strong relationship between the distance of the robot from its training spot and the associated likelihoods (as expressed by proposition five). In section 5.6 showed how this monotonic increase can be exploited in order to obtain absolute positional information and to let the robot confidently find its way back to the center spot after a kidnap (visual homing).

The learning process (as addressed in proposition four) consists of multiple steps, in which the robot first is prepared to the lighting and color situation, before it starts map learning at one or multiple spots. The camera and color calibration work in a totally stable and fully autonomous way (taking around $40s$ to complete, appendix B), and the subsequent map learning is relatively quick ($90s$ per spot, section 4.3) and also works without human intervention. A remarkable characteristic is also that a partially learned map can (and actually is) already used to support (rotational) walking, as turning on four legs is extremely prone to slippage or even obstruction. For multi-spot localization, the robot has to reach the four training spots by walking, where it is even more prone to slippage and mostly requires a manual correction of the robot's pose (once for each spot). This is why proposition four calls the learning process to be only almost autonomous. However, if the robot's body odometry was more robust in the first place, learning could surely take place fully autonomously. In sum , the robot needs only minimal human assistance when put into a new environment for learning.

Finally, it has been proven that the hardware requirements necessary for the visual compass algorithm are relatively low (section 5.5). The Aibos used for testing have only a low-quality webcam and already outdated processing power. On this hardware, a single camera frame can be processed in less than $4ms$ for learning and $13ms$ for localization. It is obvious that future robots will easily outperform the capabilities of the Aibo, therefore in many cases it would be possible to equip visually-guided robots with this form of visual compassing.

6.2 Applicability of the concept

In this section, the main contributions of this research attained for the field of mobile robotics are summed up in order to point out to what extent visual compassing can be useful to other researchers or research fields.

Above all, the visual compass approach is simple and intuitive. Several parameters have been pointed out that can be used to scale the algorithm to one's needs. Low hardware requirements make it applicable to a broad range of mobile robots. The ability to function in natural environments and to rely only on a scanning camera certainly points in the right direction for future research within the field of mobile robotics. By learning iteratively new spots, it would even be thinkable that a robot creates a mental map of even larger areas that allow it safe navigation.

The produced heading information can either be used by robots just for orientation purposes, or could serve higher developed robots as a low-level intuition about their current heading. The experiments on multi-compass localization can be regarded as a proof-of-concept that visual compassing can be used additionally to give robots an indication that a particular spot has already been visited.

The implementation of this approach serves as a practical contribution. Proven to work and already integrated in the soccer framework used by the Dutch Aibo Team it stands as a promising basis to be built upon by succeeding projects. In this line, it is interesting to know that a fourth-year project has already re-used the camera and color calibration part of my approach with the aim to make manual color calibration for playing soccer superfluous [44]. Finally, my implementation has already been published according to the open source agreement of the 4-Legged league meaning that it can be easily adapted by others. Especially the teams building upon the German Team code (like the Italian, Turkish, and Chinese teams) can seamlessly adopt the visual compass algorithm into their existing source code.

To summarize, the presented approach for a visual compass has been proven to be a lightweight and yet powerful approach to localization. The applicability has been validated by numerous experiments. The availability of a reference implementation makes it attractive for succeeding projects to further explore this class of approaches.

6.3 Discussion

The way how the visual compass approach derives heading information is relatively unique. Of course there are approaches that have certain similarities with my approach, but they differ in other aspects. Therefore it is quite difficult to compare the results directly with the work of others.

In this subsection, the visual compass approach will be discussed comprehensively. The approach will be discussed along the similarities (and dissimilarities) with respect to some of the approaches already presented in chapter 1.

In what follows now, the characteristics of the visual compass (as summarized in the answer to the research question, section 6.1) will be compared to several existing approaches which already have been presented in section 1.3.

First of all, a natural comparison is the original soccer localization module [49] of the Dutch Aibo Team itself. Of course it relies both on artificial landmarks as well as manual color calibration, and therefore can be categorized as a class of feature-based approaches (and thus is opposite to the visual compass approach). However, the resulting performance turns out to be comparable: the rotational accuracy of the Dutch Aibo Team soccer code was measured to be around 5° , while its positional accuracy was within $13.5cm$. The relatively complex algorithm resulted in a reduced frame rate of (sub-optimal) $17.2fps$. All in all, and neglecting all the differences between both approaches, the resulting accuracies are of comparable magnitude.

The subsequently presented approaches use panoramic cameras, and therefore allow the robot to take $360°$-panorama pictures.

The visual compass as described in [14, 15] yields comparable results (error below $10°$), but shows a long-term drift as it for each time only compares the two previous images from the camera. Interesting about this research is the comparison in accuracy with a traditional magnetic compass that was easily outperformed by their method: magnetic compasses rendered to be useless in their indoor experiments, and to be comparable with their results from outdoors. Therefore, the conclusion can be drawn that my form a visual compassing also outperforms a traditional magnetic compass. Moreover, the prosed distance measure is computationally very expensive, and so the visual compass could only be evaluated at $2fps$.

Another approach is described in [shopping-assistant]. Incoming images from a panoramic camera are split into sectors. Only the sector's average color is subsequently used for learning and matching[1]. However, instead of relying on a single (or small number) of training spot(s), they used a set of thousands of training images associated with the true pose, that were shot beforehand on long tours through the test environment (a small supermarket). The achieved accuracy was relatively low, namely never better than 31cm, but on average the localization error was around 50cm. Although their approach is probably lightweight enough to be used in real-time on a mobile robot, the evaluation was done offline (using the datasets).

To finalize this matter, two novel approaches coming from the 4-Legged league will be discussed. Both of them have been demonstrated at the open challenge of the RoboCup 2006 competitions, but for both no technical reports have yet been published. Tsinghua Hephaestus from China demonstrated that their robots could localize on only the field lines and the blue and yellow goals, while Wright Eagle (also from China) presented an approach where the robots could localize on a totally symmetrical field with two yellow goals. The robots removed ambiguity collaboratively by seeing each other, communicating their positions and fuzzy reasoning. Although these approaches build upon the localization of field lines (and teammates), they show that different teams in the 4-Legged league independently developed techniques applicable to landmark-free localization in the near future of the RoboCup competitions.

6.4 Future research

In this final subsection, possible directions for future research will be proposed. As there are already two officially announced succeeding projects that build on the visual compass approach, they will be described first. Then several extensions will be proposed to the visual compassing approach.

The two announced projects are:

- A DOAS project group [2] is supposed to start in January 2007 to adapt the soccer code of 2007 such that it complies with the new 4-Legged field rules (only two flags instead of four). Half of the group will focus on how to adapt the original code, whereas the other half will evaluate to what extent a full switch to compass localization is possible.
- The proposal for a Master's project [3] aims at combining the visual compass approach with the technique of manifolds from the SLAM domain (i.e. the RoboCup Virtual Robot league). A mobile robot that explores a new environment could use a visual compass to stabilize its heading estimate. After the the variance in the estimate increases, the robot could decide to learn a new map (or a

---

[1] Note that this in a way corresponds to the case when my approach uses instead of transition frequencies only the frequencies of plain colors (see section 5.4.2)

[2] Project related to the master course "Design and Organization of Autonomous Systems given at the UvA, `http://www.science.uva.nl/~arnoud/education/DOAS/`

[3] with the title "Panoramic Localization, `http://www.science.uva.nl/research/ias/masterProjects/PanoramicManifold.html`

so-called patch of a manifold). It would even be thinkable to mutually compute distances between the stored manifolds, to detect circles on the path and to close loops. In this way, a topological map could be created that would allow a visually-guided robot to explore and navigate through large environments.

Beyond these proposals, other interesting trajectories seem promising:

- For one thing, the environment is assumed to be static. However, natural environments tend to change over time and therefore have to be considered to be dynamic. In this sense, it would be a challenge to think about ways how a robot could update its correspondingly maps in a dynamical fashion. Of course the robot should not be mistaken when updating the map, which could for example be guaranteed by monitoring the likelihood values before and after the frame that is chosen to be used for an update. Anyhow, it is probable that through constant updates, the map will "wash out after a while, so attention must be paid towards mechanisms to occasionally sharpen the map by somehow increasing the contrast. Maybe the primary map has to be kept in the background to avoid rotationally drift in the long run.

The grid localization experiment has shown that visual compassing can indeed even be used for positional localization. However, for other applications like playing soccer, the availability of absolute localization is mandatory. Some ideas for extensions are:

- Instead of the simple grid filter, typical approaches like Kalman or particle filters could be used for localization. Those approaches also incorporate odometry information, and therefore integrate sensor and body readings better over time. Each training spot can simply be considered as another percept where classical filters can be attached to. With these filters, it also becomes easier to integrate other percepts. Particularly in the robot soccer domain field lines are interesting: they are easy to detect, as they are realized as clear white stripes on a green floor. Several teams of the 4-Legged league have developed approaches to localization that can even extract different types of field lines, like crossings and corners, which can also serve well as an extra feature. These approaches can be combined with the visual compass approach to either remove ambiguities or even help with (qualitative) positional estimation.
- A totally different idea has been sketched in appendix C: this approach exploits further the projective distortions that occur (small) rooms. The only prerequisites are two training spots that have a known distance, and because the visual compasses then would report slightly different heading estimates (due to different projective distortions in the learned maps), the robot could triangulate its position from this angular disparity directly. The idea seems simple and promising, so it could be an interesting project for further elaboration. An implementation could prove the validity of this approach in a series of empirical experiments.
- While the aforementioned proposition aims at establishing absolute localization either by better interpolation or a finer grid, it could also be attractive to focus more on goal-driven behavior and therefore experiment with direct navigation (which then falls in the category of active localization). The basis of this idea is the insight that in most practical cases absolute localization is only an intermediate requirement for the purpose of navigating a robot towards a certain target pose. For example in the case of a soccer playing robot, from a task-dependent point of view, only absolute localization is necessary for a small set of points on the field: the goal-keeper for example has to be able to position himself as central as possible in the goal, while the striker needs to position himself behind the kickoff circle before the game starts. Moreover, both robots need a rough estimate of how far away they are from their target point (the goal-keeper should not leave his goal box too much while interacting with an attacking robot, and the striker should not leave the soccer field

while looking for the ball). They could detect this from the likelihoods. Sometimes the robots then can actively decide to walk back to their target point (the keeper could reposition himself after ball interaction, and the striker could decide to walk back to the center if he has looked for the ball too long). This approach, would integrate two distinct research areas (localization and navigation) with the aim to create a holomonic solution with a new problem factorization in mobile robotics.

With a working implementation at one's fingertips, many of the suggested directions for future research on visual compassing can be explored with low effort. However, a complete re-implementation would maybe also reveal legacies and streamline the approach further. Anyhow, I can say that I am very curious whether new ideas and projects will arise from this approach.

Bibliography

References

[1] *World Robotics: Statistics, Market Analysis, Case Studies and Profitability of Robot Investment*, edited by United Nations Economic Commission for Europe and International Federation of Robotics, United Nations Publications, Geneva, Switzerland, november 2005.

[2] M. Asada, H. Kitano, I. Noda and M. Veloso, “RoboCup: Today and tomorrow – What we have have learned”, *Artificial Intelligence*, volume 110:pp. 193–214, 1999.

[3] R. A. Brooks, “A robot that walks: Emergent behavior from a carefully evolved network”, *Neural Computation*, volume 1(2):pp. 253–262, 1989.

[4] R. A. Brooks, “Elephants Don’t Play Chess”, *Robotics and Autonomous Systems*, volume 6(1&2):pp. 3–15, June 1990.

[5] R. A. Brooks, “How to build complete creatures rather than isolated cognitive simulators”, *Architectures for Intelligence*, pp. 225–240, 1991.

[6] M. L. Fassaert, S. B. M. Post and A. Visser, “The common knowledge model of a team of rescue agents”, in “1st International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster”, July 2003.

[7] D. Fox, “Adapting the sample size in particle filters through KLD-sampling”, *International Journal of Robotics Research (IJRR)*, volume 22(12), 2002.

[8] D. Fox, W. Burgard, , F. Dellaert and S. Thrun, “Monte Carlo Localization: Efficient Position Estimation for Mobile Robots”, in “Proceedings of the National Conference on Artificial Intelligence”, 1999.

[9] H.-M. Gross, A. Koenig, C. Schroeter and H.-J. Boehme, “Omnivision-based probabilistic self-localization for a mobile shopping assistant continued”, in “Proceedings of the International Conference on Intelligent Robots and Systems, 2003. (IROS 2003)”, volume 2, pp. 1505– 1511, IEEE omnipress, Las Vegas, USA, 2003.

[10] H.-M. Gross, A. Koenig, C. Schroeter and H.-J. Boehme, “Vision-based Monte Carlo self-localization for a mobile service robot acting as shopping assistant in a home store”, in “Proceedings of the International Conference on Intelligent Robots and Systems, 2002. (IROS 2002)”, volume 1, pp. 256– 262, IEEE omnipress, Piscataway NJ, USA, 2003.

[11] B. Hengst, D. Ibbotson, S. B. Pham and C. Sammut, “Omnidirectional Locomotion for Quadruped Robots”, in “RoboCup 2001: Robot Soccer World Cup V”, pp. 368–373, Springer-Verlag, London, UK, 2002.

[12] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda and E. Osawa, “RoboCup: The Robot World Cup Initiative”, in “Proceedings of the First International Conference on Autonomous Agents (Agents’97)”, (edited by W. L. Johnson and B. Hayes-Roth), pp. 340–347, ACM Press, New York, 5–8, 1997.

[13] J. R. Kok, M. T. J. Spaan and N. Vlassis, “Multi-robot decision making using coordination graphs”, in “Proceedings of the International Conference on Advanced Robotics (ICAR)”, (edited by A. T. de Almeida and U. Nunes), pp. 1124–1129, Coimbra, Portugal, June 2003.

[14] F. Labrosse, “Visual compass”, *Proceedings of Towards Autonomous Robotic Systems*, 2004.

[15] F. Labrosse, “The visual compass: performance and limitations of an appearance-based method”, *Journal of Field Robotics*, volume 23(10):pp. 913–941, 2006.

[16] R. Lastra, P. Vallejos and J. R. del Solar, “Self-localization and Ball Tracking for the RoboCup 4-Legged League”, in “Proceedings of the 2nd IEEE Latin American Robotics Symposium (LARS 2005)”, Sao Luis, Brazil, september 2005.

[17] S. Lenser and M. Veloso, “Sensor Resetting Localization for Poorly Modelled Mobile Robots”, in “Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA)”, 2000.

[18] D. G. Lowe, “Object Recognition from Local Scale-Invariant Features”, in “Proceedings of the International Conference on Computer Vision ICCV, Corfu”, pp. 1150–1157, 1999.

[19] A. K. Mackworth, *On Seeing Robots*, chapter 1, pp. 1–13, World Scientific Press, Singapore, 1993.

[20] F. Mantz, P. Jonker and W. Caarls, “Behavior-Based Vision on a 4 Legged Soccer Robot.”, in “Proceedings of the 9th International Symposium on RoboCup 2005”, pp. 480–487, 2005.

[21] R. Moller, D. Lambrinos, T. Roggendorf, R. Pfeifer and R. Wehner, “Insect strategies of visual homing in mobile robots”, in “Proceedings of the Computer Vision and Mobile Robotics Workshop CVMR’98, Heraklion, Greece”, 1998.

[22] F. Oliehoek and A. Visser, “A hierarchical model for decentralized fighting of large scale urban fires”, in “Hierarchical Autonomous Agents and Multi-Agent Systems (H-AAMAS’06)”, May 2006.

[23] M. Pfingsthorn, B. Slamet, A. Visser and N. Vlassis, “UvA Rescue Team 2006; RoboCup Rescue - Simulation League”, in “Proceedings CD RoboCup 2006, Bremen, Germany”, June 2006.

[24] S. Russel and P. Norvig, *Artificial Intelligence, a Modern Approach*, Prentice Hall, 1995.

[25] M. Sahota and A. Mackworth, “Can Situated Robots Play Soccer”, in “Proceedings of Canadian AI-94”, pp. 249–254, Banff, Alberta, Canada, 1994.

[26] S. Se, D. Lowe and J. Little, “Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks”, 2002.

[27] M. Sridharan and P. Stone, “Color Learning on a Mobile Robot: Towards Full Autonomy under Changing Illumination”, in “The 20th International Joint Conference on Artificial Intelligence”, January 2007, to appear.

[28] P. Stone, “Layered Learning in Multiagent Systems”, in “AAAI/IAAI”, p. 819, 1997.

[29] P. Stone and M. Veloso, “Beating a Defender in Robotic Soccer: Memory-based Learning of a Continuous Function”, in “Advances in Neural Information Processing Systems”, (edited by D. S. Touretzky, M. C. Mozer and M. E. Hasselmo), volume 8, pp. 896–902, The MIT Press, 1996.

[30] T. Svoboda and T. Pajdla, “Epipolar Geometry for Central Catadioptric Cameras”, *Int. J. Comput. Vision*, volume 49(1):pp. 23–37, August 2002.

[31] S. Thrun, W. Burgard and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*, The MIT Press, September 2005.

[32] J. H. U. Düffert, “Omnidirectional Locomotion for Quadruped Robots”, in “International Workshop on RoboCup 2005”, LNAI. Springer, 2006.

[33] A. Vardy and F. Oppacher, “Low-level visual homing”, in “Proceedings of the 7th European Conference on Artificial Life (ECAL)”, pp. 875–884, Springer Verlag Berlin, 2003.

[34] J. J. Verbeek, *Mixture models for clustering and dimension reduction*, Ph.D. thesis, Universiteit van Amsterdam, December 2004.

Referenced Technical Reports of the 4-Legged league

[35] H. L. Akin, C. Mericli, T. Mericli, K. Kaplan and B. Celik, “Cerberus05 Team Report”, Technical report, Bogazici University, Istanbul, 2005.

[36] N. Bertling, J. Dubber, S. Haiduc, B. Koch, V. Krichevskyi, M. Niess, J. Reese, P. Rossmeyer, J. Schönefeld, G. Selke, B. Seppke and M.-N. Sörensen, “Hamburg Team Report 2005”, Technical report, Universität Hamburg, 2005.

[37] W. Chen, A. North, A. M. Sianty, N. Morioka, J. Shammay and W. Uther, “Technical Report CM-Dash 2005 (Collection of BSc. Theses)”, Technical report, University of New South Wales, September 2005.

[38] L. Iocchi, D. Nardi, A. Cherubini, L. Marchetti and V. A. Ziparo, “S.P.Q.R. + Sicilia RoboCup 2005 Report”, Technical report, Università di Roma, 2005.

[39] M. J. Quinlan, S. P. Nicklin, K. Hong, N. Henderson, S. R. Young, T. G. Moore, R. Fisher, P. Douangboupha, S. K. Chalup, R. H. Middleton and R. King, “NuBots Team Report”, Technical report, University of Newcastle, 2005.

[40] M. J. Rey, R. S. Oropeza, A. A. Lopez and J. R. Uresti, “Tec Rams - Technical Report 2006”, Technical report, Tecnologico de Monterry Campus Estado de Mexico, 2006.

[41] T. Röfer, T. Laue, M. Weber, O. von Stryk, R. Brunn, M. Dassler, M. Kunz, T. Oberlies, M. Risler, H.-D. Burkhard, M. Jüngel, D. Göhring, J. Hoffmann, B. Altmeyer, T. Krause, M. Spranger, U. Schwiegelshohn, W. Nistico, S. Czarnetzki, T. Kerkhof, M. Meyer, C. Rohde, B. Schmitz, M. Wachter, T. Wegner and C. Zarges, “German Team RoboCup 2005”, Technical report, Universität Bremen, Humboldt-Universität zu Berlin, Technische Universität Darmstadt and Dortmund University, 2005.

[42] M. Veloso, P. E. Rybski, S. Chernova, C. McMillen, J. Fasola, F. von Hundelshausen, D. Vail, A. Trevor and R. R. E. S. Hauert, “CMDash05: Team Report”, Technical report, Carnegie Mellon University, 2005.

[43] Q. Wang, L. Liu, H. Li, C. Rong and G. Xie, “The 2005 sharPKUngfu Team Report”, Technical report, Peking University, 2005.

Author's Publications

Conference papers

[44] D. van Soest, M. de Greef, J. Sturm and A. Visser, “Autonomous Color Learning in an Artificial Environment”, in “Proc. 18th Dutch-Belgian Artificial Intelligence Conference, BNAIC’06”, Namur, Belgium, October 2006.

[45] J. Sturm, P. van Rossum and A. Visser, “Panoramic Localization in the 4-Legged League”, in “Proc. 10th RoboCup International Symposium”, Bremen, June 2006, to be published in the Lecture Notes on Artificial Intelligence series, Springer Verlag, Berlin.

[46] A. Visser, P. van Rossum, , J. Westra, J. Sturm, D. van Soest and M. de Greef, “Dutch AIBO Team at RoboCup 2006”, in “Proceedings CD RoboCup 2006, Bremen, Germany”, June 2006.

[47] A. Visser, J. Sturm and F. Groen, “Robot companion localization at home and in the office”, in “Proc. 18th Dutch-Belgian Artificial Intelligence Conference, BNAIC’06”, Namur, Belgium, October 2006.

[48] N. Wijngaards, F. Dignum, P. Jonker, T. de Ridder, A. Visser, S. Leijnen, J. Sturm and S. van Weers, “Dutch AIBO Team at RoboCup 2005”, in “Proceedings CD RoboCup 2005, Osaka, Japan”, July 2005.

Technical Reports

[49] J. Sturm, A. Visser and N. Wijngaards, “Dutch Aibo Team: Technical Report RoboCup 2005”, Technical report, Dutch Aibo Team, October 2005.

[50] A. Visser, J. Sturm, P. van Rossum, J. Westra and T. Bink, “Dutch Aibo Team: Technical Report RoboCup 2006”, Technical report, Dutch Aibo Team, December 2006.

Appendix A

The Sony Aibo as a platform for robotics research

A preliminary step is to find a suitable robot platform for which the new approach can be designed and on which it can be implemented. A natural choice is to use the Sony Aibo, which has been used successfully for years in the 4-legged soccer competitions.

In this chapter, the Sony Aibo robot is thoroughly evaluated as a robotic platform. Its hardware equipment is itemized, and the available software frameworks are examined. The body motion is tested, in order to get an estimate of the accuracy of both walking and turning. A close look is taken at the pictures captured by the Aibo's internal camera, and the specific problems arising from the fact that the camera is installed in the robot's nose are illustrated. Moreover, some preliminary experiments were conducted to get an impression of the computational power of the Aibo in practice.

Moreover, aspects like the availability and the existence of experience is examined at the end of this chapter.

A.1 The Sony Aibo robot

The Aibo is an entertainment robot developed and sold by Sony since the late 90's. The design and equipment of the Aibo has changed over the years, but the underlying idea stayed the same: the Aibo is a entertainment dog, originally intended as a playing companion for children. The newest model is the ERS7-3M. Unfortunately, the production of new Aibos (and all other robotic research) has been abandoned by Sony in January 2006.

A standing Aibo is both around 30 centimeters tall and long and weights around 2 kilogram. Twenty step motors allow the Aibo to perform a broad spectrum of motions. Next to the main servos in its legs and the neck, the Aibo is able to open its mouth, waggle its tail and its ears. Its face contains an array of colored LEDs that are used by the original software to express emotions. Sensors in the servos allow the Aibo to detect when it gets stuck (or is being held too tightly by a child) such that it can protect itself by disengaging its actuators.

The Aibo has lots of other sensors: next to the camera installed in its nose that the Aibo uses for finding its ball and toy bone, it has stereo microphones in its ears for voice commands or playing games with its owner. Acceleration sensors reveal the configuration of the Aibo's head and allow it for example to look parallel to the ground. Two distance sensors (one in the robot's chest and one in its head) are used by the original software to prevent the Aibo from falling off a table. After all, the Aibo is still intended for a home environment which – in general – can be dangerous for a robot. Furthermore, the Aibo can

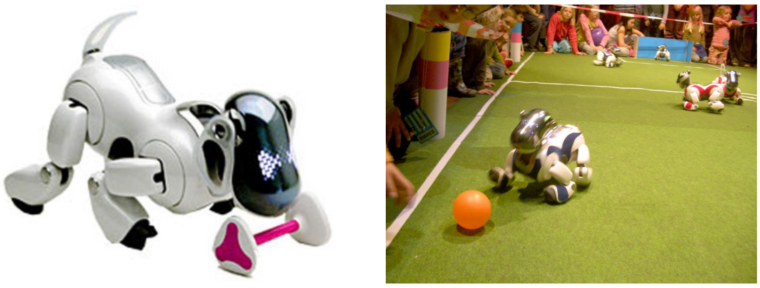


Figure A.1: Sony's Aibo entertainment robot, series ERS7. **Left:** The Aibo's default software (Aibo Mind) lets the Aibo play with its pink bone. **Right:** Soccer playing Aibos of the Dutch Aibo Team (picture taken at the Cinekids Festival 2005, Amsterdam).

feel whether there is ground under its feet. Buttons in its head and on its back can be used for simple user interaction.

Moreover, in its waist, the Aibo carries a fully equipped (once) state-of-the-art internal computer: a 64-bit RISC processor with 567 MHz with 64 MB of RAM. The program is supplied on a 8, 16 or 32 MB memory stick. An integrated 802.11b wireless card provides network connectivity with other robots or a PC in a simple way using the TCP/IP protocol.

At a price of around 2000 Euro, the Aibo as a toy has only been attractive for a small group of wealthy parents and their children. However, the technical equipment of an Aibo permits for much more: in 2001, Sony released the OPEN-R software development kit to the general public, which opened the Aibo as general playground platform for robotic scientists. Since then, the Aibo gained a growing interest from many research groups. The advantages of buying a ready-made robot dominate the benefits of developing own robots by far: Aibos are easy to acquire and maintain (by the implied warranty and the existence of several repair centers all over the world), particularly robust (because of its design as a toy), and contain a variety of sensors and actuators.

A.2 The software framework

On the Aibo Sony uses a multitasking operating system called Aperios. The OPEN-R software development kit contains a set of libraries that allow the user-space programs to issue system calls that access the robot's sensors and actuators. The data from and to the robot is at this level completely raw and unprocessed. For example, conversion algorithms are necessary to translate the control values (and the feedback PID values of the motors) into angles and vice versa. Furthermore, it is necessary to carefully obey certain acceleration limits, in order not to demolish the motors.

Although programming directly on this level of the operating system is possible, the development time can be reduced significantly by using one of the available frameworks that already come with certain abstractions.

The popular Tekkotsu framework is an event-driven extension for the OPEN-R libraries that already contains a lot of basic functions. The event model encourages the development of small behavioral modules, and already contains a number of primitive modules that can be combined with own modules. For example, Tekkotsu already contains modules that can segment incoming images into blobs. Primary movements, such as walking and turning, are already available. With these functions at hand, it is easy to, for example, let the Aibo follow an orange ball. It is therefore a preferred basis for educational projects, as the learning curve is rather steep, and in very little time impressive programs can be written. However, the debugging possibilities of Tekkotsu are rather limited.

Debugging facilities are crucial to robotic software development. Writing these (**offline**) test routines for all developed algorithms on the PC requires a lot of extra time. In most cases, it is also necessary to test the implemented algorithms after successful dry-runs directly on the robot. For such **online** tests, it is useful when the robot can be monitored and controlled remotely in an convenient way. Particularly difficult are bugs due to which the Aibo crashes in one way or the other. As soon as interprocess or eventually the wireless communication breaks down, it is hard to guess where a fatal error originated from. Such errors can have many reasons, infinite loops, buffer overflows, or other exceptions (such as a division by zero). In some cases, only good guessing or special developer's tools (like a serial cable) can help out.

The Tekkotsu framework does not contain inherent possibilities for local and remote debugging. It is therefore necessary to develop particular test programs on a PC in parallel. The same holds for monitoring and control applications that have to be programmed alongside the code for the Aibo. Typically, Tekkotsu developers implement Java control interfaces for this purpose. The Software Project [1] 2006 in Utrecht used Tekkotsu for the implementation of the new goal challenge in 2006.

The featured framework of the Dutch Aibo Team is based on the code release by the German Team in 2004. The framework contains hardware abstraction layers for both a PC and the Aibo, which means that the code can transparently run on both systems, making it possible to debug the Aibo code on a PC in a simple manner. While the code running on the Aibo controls directly its sensors and actuators, the same code – when executed on a PC – is attached to a simulated body in a simulated environment. The front-end application for the PC side is called RobotControl, and it integrates both the simulator as well as the control over remotely connected robots in a consistent manner.

A.2.1 Robot software

The soccer software for the robot is organized into independent processes which are executed by the operating system quasi-parallel. The **motion process** produces a continuous series of motion commands for the robot's joints at a frame rate of approximately 120 Hz. The **cognition process** starts a new processing cycle for each incoming camera image and therefore runs at around 30 Hz (or less). The **logplayer process** takes care of the asynchronous communication with the PC and other robots. The processes communicate through message channels that are provided by the operating system.

The major processes (like motion and cognition) are split up into smaller components, called **modules**. Each module is assigned a specific task with corresponding interfaces for data in- and output. In this context, a module refers to an empty slot in the architecture where an actual **solution** can be selected and activated. For most modules, multiple solutions are available. This software factorization facilitates collaboration between different programmers, research groups and universities.

This makes it possible for different universities and institutes to collaborate in the same framework by developing different solutions for a single module.

The modules communicate among each other using shared memory. Hierarchically, this can be seen as a data flow from top to bottom (and in very few cases vice versa). Figure A.2.1 shows the complete architecture of both the cognition and motion process. Processing takes place from top to bottom; arrows indicate the data flow between the modules. In the top row, the incoming sensor data is displayed, like images from the camera and tactile and acceleration data. The data is processed by different modules into intermediate abstraction levels. In the center layer, the behavior module decides on the current strategy and intermediate goals. These are finally processed by the walking engine into motion output commands.

---

[1] a half-year BSc project for computer science students at the University of Utrecht

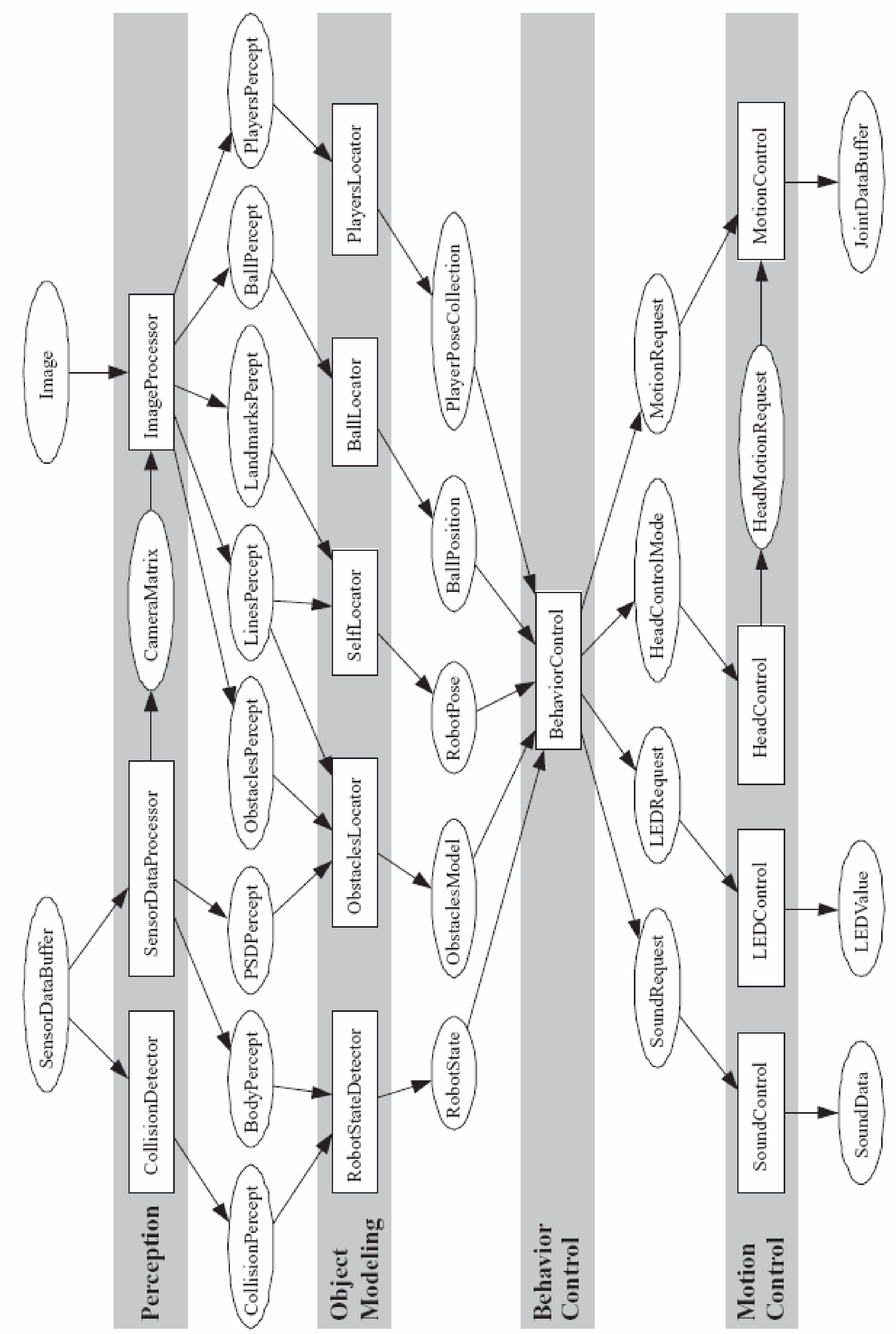

Perception
Object Modeling
Behavior Control
Motion Control
Image
SensorDataBuffer
ImageProcessor
CameraMatrix
SensorDataProcessor
CollisionDetector
PlayersPercept
BallPercept
LandmarksPerept
LinesPercept
ObstaclesPercept
PSDPercept
BodyPercept
CollisionPercept
PlayersLocator
BallLocator
SelfLocator
ObstaclesLocator
RobotStateDetector
PlayerPoseCollection
BallPosition
RobotPose
ObstaclesModel
RobotState
BehaviorControl
MotionRequest
HeadControlMode
LEDRequest
SoundRequest
MotionControl
HeadMotionRequest
HeadControl
LEDControl
SoundControl
JointDataBuffer
LEDValue
SoundData

Figure A.2: Architecture overview of the robot soccer software used by the Dutch Aibo Team. Arrows indicate the data flow (ellipses) between the modules (rectangles).

A.2.2 Simulator

The robot software can both be executed on the robot and on the PC. The PC-side creates a simulated environment for the robot that allows much easier debugging.

The simulator maintains a three-dimensional representation of the world (i.e. the soccer field). The map itself is static; the only moveable parts are the simulated Aibo's on the field.

Per motion frame, the simulated joints are adjusted according to the motion commands, and then the resulting position of all body parts is computed. Paws touching the ground experience friction, and thereby move the robot through the simulated world. The motion simulation is not yet very elaborate: no motion noise is present, and the robots cannot interact (or modify) the environment; so for example playing with the ball or pushing other players is not possible; the players slide through each other as if they were ghosts. Still, this motion simulation allows the programmer to efficiently test and debug motion solutions; for example the walking engine of the Aibo, or watch the robot performing higher-level behaviors.

The three-dimensional world map allows the simulator also to visualize the scene and the robot. The scene description contains the soccer field, including the colored flags, goals, and other robots. In principle, other objects like the audience or a referee could also be described. The simulator is not bound to a robot soccer specific scene, but by far nobody took the trouble to sketch something other than the regular field and the robots. Objects are described by a list of colored polygons, marked by lists of three-dimensional points. The simulator allows an arbitrary point of view, so for example the robot can be watched from the side or from above. Moreover, it is possible to move and turn the robot with the mouse to put it into a certain pose. Next to this three-dimensional view, the field is plotted from above in a simplified two-dimensional representation that allows the overlay of various debug drawings (like the perceived ball position, obstacles or other players, etc).

After the evaluation of the robot configuration, the position and direction of the robot's nose camera also becomes available. This information is used by the simulator to create a simulated camera image for the robot. No noise or clutter is added, so the images are as clean as the scene is described. Still this is useful to test image processing algorithms under perfect circumstances, and provides the opportunity for step-wise debugging.

The motion and camera modeling forms the basis for executing most tests within the simulator itself. Special library functions allow the creation of a variety of debug output: next to a text console, modules may send images back for display to the main application, or augment the field plot by debug drawings (lines, circles, etc).

The execution time of the simulated robots is much higher than that of the real robot. So for example, usually only two frames per second are reached on a normal computer (versus 30 on a real robot). When time and space (or processing power) is plentiful, then even multiple robots can be simulated at the same time. The Software Project 2005 in Utrecht used the simulator to test whether both teams each consisting of four robots finally could reach the correct kickoff positions on the field. Even though this simulation took more than an hour on a regular computer, but eventually it could be shown that the complete processing cycle from the image processing, over the localization, behavior creation and finally the walking engine worked fine at least under perfect circumstances.

A.2.3 Remote debugging

It is also possible to remotely tap into the information flow between the modules running on an Aibo to inspect and modify values there in real-time. For example, camera images can be sent to the computer for inspection, extracted landmark and object information can be shown or the estimated world state can be visualized by the main application. Solutions for modules can be switched over while the Aibo is playing soccer. Moreover, some modules even allow the online modification of certain parameters. So, for example, the image processor can be supplied with an updated color table, the locator module accepts new parameter values for the particle filter in real time, and the behavior module allows the manual activation of a behavior or the modification of behavioral output symbols next to the real-time inspection of the activated behavioral paths.

A.2.4 Debugging by playback

However, all of this does not help when a certain solution contains a bug that only appears in combination with real-world data. Therefore, incoming debug stream can also be stored in a file and subsequently be played back to a certain module of a simulated robot. Then a regular debugger on the PC can be used to track down the error in the code, on data that originally came from the robot.

Most programming errors can be resolved this way. When one is interested in high detail on what is going on inside one or more modules in slow motion on real-world data, this is probably a nice way to do so. However, bugs that only appear very occasionally (and harmful data for playback is difficult to acquire) or problems that only occur in combination with other robots (teamplay), it is still necessary to fall back on slower debugging techniques.

A.3 Body odometry

The task of the walking engine is to maneuver the robot from the current pose into a given target pose. In order to do so, the robot has to move its legs in a coordinated fashion; or to put it more technically, it has to generate a continuous motion sequence and to send it to its joints. As there are only 6 degrees-of-freedom in both the original pose and the target pose together, but already 12 degrees-of-freedom for all four legs at each time step (that need to be updated at $120fps$), it is obvious that many motion sequences can lead from one body pose to another. A particular way of moving the legs in order to get along is also called a walking style, and the selection of a certain **walking style** is necessary to keep the generation of the required motion sequence computationally feasible.

The walking engine of the DT2005 is based on the omni-directional walk developed by [32] in 2004 (and originally invented by rUNSWift from Sydney under the name pWalk in 2000 [11]). The main idea here is to treat the robot's legs as if they were omni-directional wheels. Each wheel can move in an arbitrary direction at an arbitrary speed (see fig. A.3(left)). By this level of abstraction, the difficult problem of walking on four legs is transformed into a problem of driving on four wheels – for which standard solutions (and controllers) exist.

Although, in theory, this approach looks very straight-forward, it turns out that the 12 joints in the robot's legs make it necessary to find a appropriate parameter set for the walking engine – due to incalculability of these parameters and robot-specific individualities. In 2004, the German Team used a genetic algorithm to find a (locally) optimal parameter set, but during the preparations for the RoboCup 2005 it turned out that all of the robots of the Dutch Aibo Team had a strong drift to the right when they actually should walk in a straight line. We corrected this in Osaka by a simple counter-rotation of the odometry feedback to the left [ref techreport2005], to at least ensure that walking requests on straight walking really resulted in a straight walk. This was particularly important for robots which had to approach the ball in a straight line: the actual drift to the right was perceived as a displacement of the ball to the left for which the robot corrected its movements, and so the robot's estimation on its heading (and the heading towards the opponent's goal) was confused.

However, by this simple correction, we have only shifted the errors in odometry estimation: the turning and sideward movements still yielded very incorrect odometry values. In order to test some basic features of the walking engine, a test behavior was implemented that let the robot walk blindly in a straight line 2m forward, then turn by 90 degrees to the left, then walk back sideward 2m to the original spot and finally turn back 90 to the right, purely based on their own odometry estimations. In this way, the

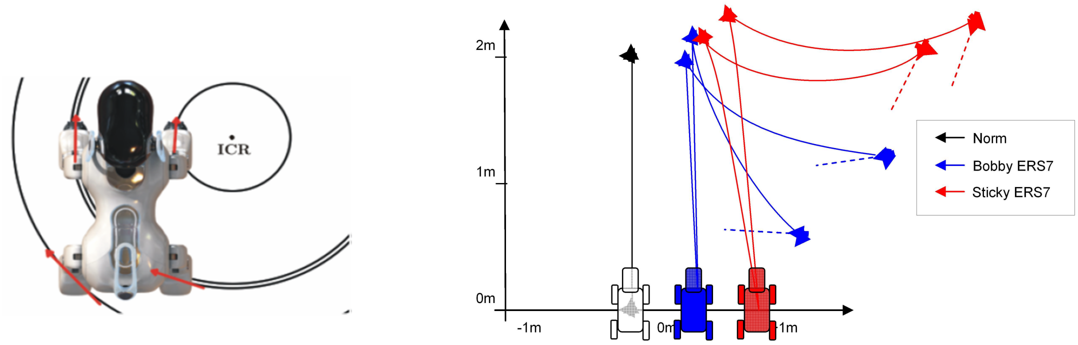


Figure A.3: **Left:** The walking engine of the Dutch Aibo Team treats every leg of the Aibo as an independently steerable wheel. **Right:** Walking on four legs is extremely prone to slippage and other motion noise. In this experiment, the robot should walk $2m$ in a straight line, then turn by 90° left, and walk sidewards back to the origin. Walking straight works fine, but turning and sideward movements show significant deviations.

odometry error of straight walking, sideward walking and left/right turning could be evaluated graphically by the resulting path. In the optimal case, when following this behavior, the robot should end up at the same place and with the same heading in which the robot started. Practical tests however revealed that actual paths differ from the desired walk significantly. Fig. A.3(right) displays the desired (control) path in black, and the actual traces of two robots (two runs on each robot, first robot in blue, second in red).

It can be seen that the straight walking works relatively well for robots, although it can clearly be seen that the motion traces of each robot have their particular characteristics. Furthermore it can be stated that turning requests fundamentally do not let the robots rotate far enough (neither to the left nor to the right). Finally, the sideward movements are particularly incorrect as they the robots actually moved in a circle of approximately 2m radius – especially under consideration that the field dimensions are 6x4m.

A.4 Head odometry

In order to correctly interpret images from the camera, it is necessary to know where the camera points at or equivalently which part of the world the robot perceives. In technical terms, the corresponding **camera matrix** has to be determined, for example to project the **horizon** into the image and to know in which direction the image is orientated (**up-vector**).

The correct estimate of the current body pose is only half one half of the solution. As the camera of an Aibo is installed in the point of its nose, the camera matrix additionally also depends strongly on the current pose of its head.

The head has (with respect to the body pose) three additional degrees of freedom: firstly, the Aibo may raise or lower its neck, secondly, it may raise or lower its head (which is attached upon the neck), and thirdly, may turn its head to the left or right.

Because of this flexibility of the neck and the fact that the opening angle of the Aibo's camera is only around 50 degrees, it became common practice in the 4-legged league to let the robot constantly look around in order to see more of its surroundings. With its three degrees of freedom, it is possible to

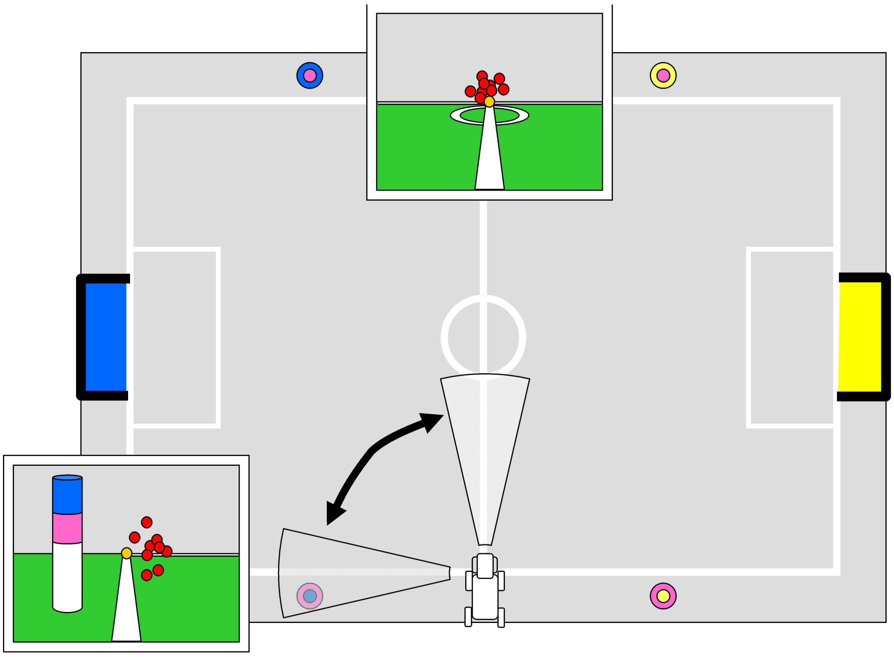


Figure A.4: The visual field of the Aibo can be greatly extended when the robot scans with its head from left to right. By doing so, additional motion noise is introduced in the camera matrix. In this experiment, this head motion noise has been measured empirically.

produce a scanning behavior that enables the robot to perceive more than 230 degrees of its surroundings in horizontal direction.

As the head scans approximately once per second from left to right and back again, it is reasonable to give some attention to the additional noise that is introduced in the camera matrix by the continuous scanning of the head. This turns out to be even more problematic when the robot is walking or running (and thereby shaking its head even more). Hereby it is important to know that we observed that some robots seem to be more prone to shaking with their heads than others when walking across the field. For the RoboCup world championships, we explicitly selected robots with accurate body odometry and low head shaking for the competition games.

To test the head motion under perfect circumstances, a still-standing robot was placed on the center of the sideline (see fig. A.4). It could see both the centerline (when looking straight ahead) as well as the sideline (to its left and right when scanning with its head). Two well-visible spots have been selected for the analysis and compared with the expected position: the intersection of the midline with the horizon, and the intersection of the left part of the sideline with the horizon. For these points, the expected pixel position has been marked (back-projected) into the camera images, and subsequently the derivation was measured (in pixels).

Table A.1 gives the results of the head-odometry experiment. The distances are displayed both in pixels and degrees for easier interpretation. It can be seen that the mean values differ somewhat between the different measurement series: there are differences in the magnitude of a few degrees between them.

One observation can clearly be made: the variances of both series are of the same magnitude; around 1.5 degrees. This is the angular error introduced by the camera motion and hence is always present in the estimate of the camera matrix.

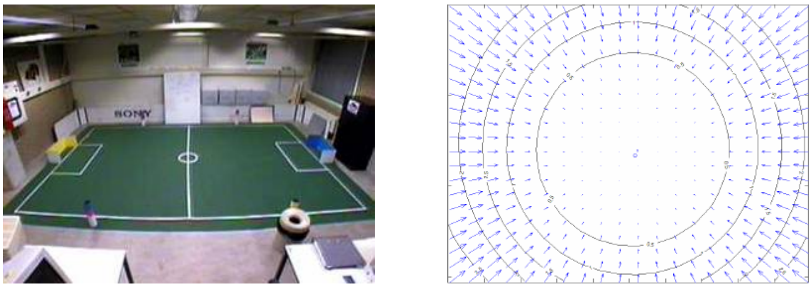


Figure A.5: Radial distortion occurs naturally in most optical systems. **Left:** The wide-angle field camera of the UvA Robolab shows extreme radial distortion. **Right:** Empirically measured [41] radial distortion of the Aibo's camera.

A.5 Camera Quality

A lot can be said about the quality of the internal camera of the Aibo. In fact, it suffers from most well known problems related to especially (but not only) cheap cameras: radial distortion, natural vignetting, color fault, motion blur and readout latency. In the following subsection, the major aspects of image deformation will be presented briefly and an explanation will be given of the countermeasures which can or already are taken to compensate for them.

Radial distortion occurs when the lens does not magnify the image uniformly, but the level of magnification depends on the distance from the image center. Some radial distortion is inherent in most optical systems, but usually better lenses (or better mounting) can diminish its effects significantly. The effects of radial distortion can usually be described by a polynomial of low order; its coefficients need only to be estimated once. Subsequently, the distortion of a pixel can quickly be removed. Fortunately, it turns out that all Aibo cameras have exactly the same radial distortion to a great extend, which means it is sufficient to calibrate these parameters once and use them for all Aibos. Fig. A.5(left) shows for example the wide-angle view of the ceiling camera of our Robolab that suffers obviously of extreme radial distortion.

Although the German Team laid the basis for the radial distortion correction in the base code of the Dutch Aibo Team, it is doubtful whether this correction is really necessary for the Aibo's camera. Fig. A.5(right) shows the original correction table for the Sony Aibo. Each circle stands for 0.5 pixels that the image is off due to radial distortion. It can be seen that no pixel is off more than three pixels, large parts of the picture even less than a single pixel. Compared to the noise of the head motion, the effects of radial distortion are comparatively negligible.

Natural vignetting (or illumination falloff) is also a typical property of the lens system and is, like the radial distortion, stronger marked with bad lenses, but inherent to this kind of projection. When thinking about pin-hole cameras the effect of natural vignetting can easily be understood: as the projection screen

Derivation	Horizontally		Vertically	
	mean	sigma	mean	sigma
Heading 0° (in front of robot)	4.7pix	6.2pix	8.2pix	4.0pix
or equivalently	1.1°	1.5°	2.0°	1.0°
Heading 90° (left of robot)	16.7pix	7.3pix	1.5pix	4.7pix
or equivalently	4.0°	1.8°	0.4°	1.1°

Table A.1: Camera odometry when robot is scanning with its head.

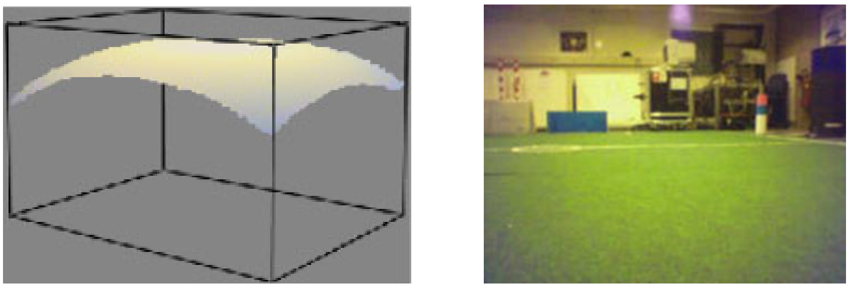


Figure A.6: Vignetting is another typical problem of (cheap) cameras. **Left:** Empirically measured [41] vignetting of the Aibo Camera. The Aibo looked at a constantly-illuminated white surface, but the reported Y-values significantly decrease in direction of the corners. **Right:** Pictures taken by the Aibo also exhibit strong vignetting.

is planar, the corners of the image per area unit are comparably less illuminated by the scene than the center region. Although this effect can be interesting in some cases for the ambitious photographer, it poses additional problems for automatic color classification: depending on the strength, countermeasures are inevitable.

When an Aibo looks at a uniformly white environment, the camera should report a uniform color on all pixels. However, the perceived color by the Aibo camera strongly depends on its distance from the center. Fig. A.6(left) shows the Y-channel when the Aibo looks at a white paper. Its color system actually already de-correlates the color values into YCrCb (brightness-red-blue), but as it can be seen all three channels suffer from the illumination falloff (which already indicates that the de-correlation is only partial). In order to correct such a distortion, a white-reference like the above (or generalization of it) can be and actually is used for normalization. However, normalizing every pixel using a lookup table is quite time consuming (cf. Processing Power). In practice (and the DT2005 code), color correction is only performed right before the color class lookup – by doing so, only a small percentage of every image has actually to be corrected.

The camera exposes the image sensor to the scene for a period of time (called the **exposure time**), before the accumulated electrical charge of a cell is read out. The exposure time is inversely related to the **shutter speed**. Objects in front of the camera that change their position significantly within the exposure time get blurred by this motion. Therefore, the shutter speed is usually chosen as fast as possible, to diminish the unwanted effects of motion blur. However, as fast shutter speeds result in small exposure times, less-exposed images are intrinsically darker. High-speed cameras therefore need to be especially light-sensitive.

Unfortunately, the Aibo camera is neither particularly light-sensitive nor is the shutter speed eligible from a broad range. Even with the highest camera sensitivity (also referred to as high gain) and the shutter speed set to high (and thereby satisfying both requests), the images appear both relatively dark on one side and the motion blur is still visible on the other side on pictures taken from the UvA Robolab.

With these preliminaries, it became necessary for the 4-Legged league to impose that the official soccer field is illuminated at 1000 lux (which is comparable to the illumination in a TV studio; even a bright office environment is lit by approximately only 400 lux). In the UvA Robolab, six spots each of 300W are used to illuminate the field which is sufficient[2].

The Aibo's camera consists of a CMOS imager. CMOS imagers are commonly used for webcams, and are the cheap alternative to CCD imagers as all electronics can directly be integrated in the image

---

[2] $6 \cdot 300W \cdot (17lux/W)/(6m \cdot 4m) \approx 850lux$

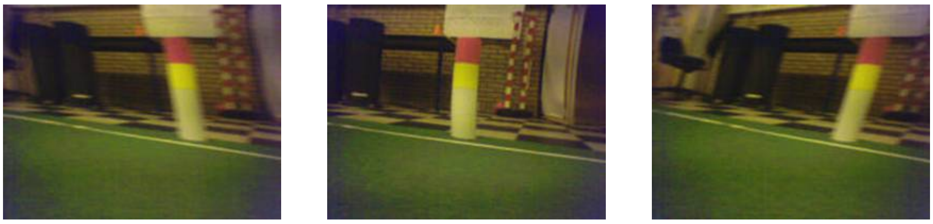


Figure A.7: The constantly scanning head motion of the Aibo leads to distorted images due to the rolling shutter of the Aibo's internal CMOS camera. The Aibo is scanning with its head at normal velocity from left to right (**left**), stops (**middle**) and scans back in the opposite direction (**right**).

sensor. In both cases, the sensor chip consists of a light sensitive pixel grid that converts incident light into electrical charge.

In CCD imagers, the light-sensitive matrix is exposed for the length of the shutter time, and then the electric charge is typically transferred quickly into a second analog buffer from where it is read out and digitized sequentially. The buffer ensures that the pixel cells are no longer exposed to the scene (and so preventing readout smear or the necessity for a mechanical shutter). This kind of exposure is also referred to as uniform shuttering, as the complete image is transferred into the buffer at the same time. The disadvantage of this approach is that the buffer covers for technical reasons part of the CCD surface and thus reduces the light-efficiency by the half.

CMOS pixel cells (as used in the Aibo) on the other hand digitize the electrical charge right away, and so the extra buffer becomes superfluous – at the price that uniform shuttering has to be given up: the synchronization of the individual cells would require extra transistors and therefore reduce again the surface for the light-sensitive cells. Although CMOS cameras with uniform shuttering exist, commonly shuttering is implemented using a rolling shutter: each pixel is exposed, read out, and reset individually in a sequential way, one after another. The disadvantage of this method is that moving objects (or a moving camera) result in distorted images.

As the Aibo is scanning continuously with its head, the complete scene is in motion, resulting in a uniformly distorted image. Fig. A.5 show examples of three snapshots while the robot was scanning with its head to the right, standing still, and scanning back to the left again. It can be seen that the horizon and the flag are only perpendicular in the middle image when the head was standing still. In the other images, the lower scan lines have been read-out a few milliseconds after the upper scan lines, when the head was already rotated further and therefore pointing in a different direction.

As the current head orientation can simply be approximated between two frames by linear interpolation for each scan line, it is easy to correct for this distortion. Either the orientation of world horizontal and vertical scan lines can be computed, or pixel coordinates can be corrected and mapped onto world coordinates by a simple linear correction.

A.6 Processing power

The processing power of the Aibo is defined by its 64-bit RISC processor of the MIPSEL family. It is clocked at 567 MHz, which was a reasonable speed for a mobile computer in the year 2001 when the latest Aibo model ERS7 was released.

As the processing power has to be shared by several processes running simultaneously (like motion planning, wireless communication and image processing), a reasonable experiment is to measure the

Function	Processing time
Image copy (memcpy of 192KB)	7ms
Image color correction	8ms
Image convert (uint8 to double)	18ms
Gaussian convolution (anisotropic)	
smoothing (0^{th} order Gaussian)	67ms
derivation (1^{st} order Gaussian, steered filter)	78ms
Convert to magnitude image ($M = \sqrt{Y^2 + Cr^2 + Cb^2}$)	146ms
Threshold magnitude image (edge detect)	11ms
Image convert (double to uint8)	21ms
Time between two camera frames	33ms
Maximal processing time for image processing	20ms-25ms

Table A.2: Measured processing times of Sony Aibo ERS7

execution time of typical operations. This can already indicate what kinds of algorithms are feasible on this platform and which are not.

In the first prototype of my approach, a new solution for the image processor was implemented. It turned out to run very slowly on the Aibo (around 2fps), and triggered this intermediate research on the actual processing power of the robot. Simple functions like copying an image, converting it to double, or convolving it with a Gaussian (for smoothing and derivation) were executed and their execution time was measured. The results showed that even the simplest image processing steps (like "memcpy-ing the image) already took an alarming amount of time (see table A.6). Converting it to double, or even computing its first derivative already exceeded the small timeslot available for image processing – at least when processing at the full Aibo frame rate is desired.

Given these measurements, a few conclusions can be drawn – concerning the approach and its implementation. Firstly, converting the image to double is completely out of the question – which means that color-invariant approaches are neither applicable. Secondly, even if the derivatives were computed directly on the integer image, its computational effort would have exceeded the available time for image processing by far. Image rotation or rectification is computationally at least as complex as computing the derivative, and therefore infeasible.

At this point, an important design decision of other teams of the 4-Legged league (see section 1.3) becomes comprehensible: either the complete image is processed, for example by merging pixels of the same color class to blobs and finally into rectangular bounding boxes (ignoring the orientation of the image), or only parts of the image are scanned, but then along a grid of scan lines (that are for example aligned parallel to the ground or to standing objects in the world).

Certainly when developing an image processing algorithm, it is wise to constantly keep an eye on its execution time, as even simple operations can take a dangerously long time. For the development of the main algorithm, constant attention has been paid to the execution time, to make sure that image processing is definitely possible in real-time on the Aibo.

A.7 Availability of Aibos

A simple, but necessary constraint is the question for the availability and popularity of Aibos itself, especially after Sony announced in March 2006 to discontinue the production of new Aibos and abandon the development of its potential successor, the QRIO. Fortunately, there are at least still 11 Aibos in the Netherlands for academic research (8.5 of them functional), three of which belonging to the Universiteit van Amsterdam. In the international comparison, there have been 28 teams participating at the yearly competitions of the 4-Legged league, and considering the amount of published articles about the Aibo in general (also in research directions of the Aibo as a social companion), this definitely proves that Aibos are (and have been) used by a broad academic community.

A.8 Experience at the UvA and in the Dutch Aibo Team

The experience that has been built up over the years in and around the Dutch Aibo Team both about the Aibos and about programming frameworks can be of great advantage compared with starting from scratch under a new framework or a new robot platform. Demonstrations, symposia and team meetings initiate the information exchange and stimulate helpful personal links between the Dutch universities.

As the Dutch Aibo Team is participating regularly in several international competitions (like the German Open/Dutch Open and the world championships), contact with other groups is available and discussions about new approaches can yield interesting insights. The software written by the Dutch Aibo Team is compatible not only with the German Team (from which it originally descends), but also all other descendants, like SPQR (Italy), Wright Eagle (China), sharPKUngfu (China), and also two German spin-off projects of the original German team, namely the Microsoft Hellhounds and the Hamburg Dog Bots.

A.9 RoboCup competitions

The technical challenges of the 4-Legged league offer the playground to show new approaches in science and technique in international comparison. Within this frame, we presented the compass function at the Open Challenge this year at the world championship in Bremen. The idea itself could also be presented in the poster session of the RoboCup symposium, along with a live presentation with the Aibo orientating on a table in the conference hall, opening discussions with other researchers.

A.10 Summary

In this chapter, the Sony Aibo is examined carefully with the aim to rate its suitability as platform for robotic research.

1. Important aspects of the Aibo's **hardware** have been evaluated empirically; like the accuracy of body and head motion, but also the characteristics of its internal camera and its computational power.

2. The **software** framework as used by the Dutch Aibo Team has been analyzed and described in detail. It forms a good framework for robotic soccer as it already contains the soccer functionality. The existence of a simulator and remote-debugging tools facilitate software development, and the good modularity allows the incorporation of own solutions in a stable and well-defined way.

3. Finally, aspects such as the **availability** of enough Aibos, **experience** within the Dutch Aibo Team and the 4-Legged league in general have been discussed. Moreover, RoboCup constitutes a stimulating forum for both exchange and **international competition**.

Appendix B

Camera calibration and preparation

Before the compass sensor can go into work, the robot has to set up its internal camera and establish a meaningful color-to-colorclass mapping. This is done in the **preparation step**, shortly after the robot has been activated.

B.1 Auto-Shutter

The first parameter to be determined is the right adjustment of the Aibo's camera. While there is no need to set the white balance (as the most important colors are found by clustering in the next step anyway), the camera gain and the camera gain and the shutter time influence greatly the quality of the images. Too dark images contain mostly noise in all three channels, while too bright images tend to saturate and therefore loose information at the upper end.

Choosing the right setting is always a compromise. The images should be well-balanced and the brightness distribution should use as much as possible of the available range. As the Aibo camera is known to be light-insensitive, this means in particular that – with camera default – the images tend to be too dark and require extra brightening. To a certain extend, this can be compensated by increasing the camera gain, but beyond that we have to increase the shutter time. However, this leads to more motion blur in the images and therefore decreases the sharpness. Table B.1 lists the camera options though that the Aibo cycles while searching for an appropriate setting.

A good indicator of the dimensions of the color space is to evaluate the dynamic range of the channels;

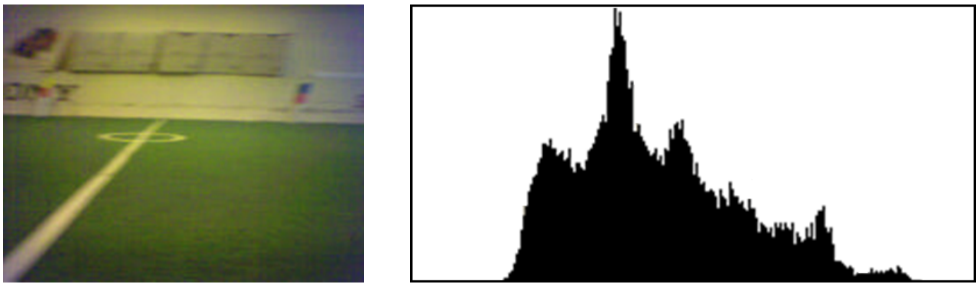


Figure B.1: The camera needs to be set up depending on the illumination conditions. **Left:** Example of a well-illuminated image from the Aibo camera. **Right:** Corresponding histogram (Y-channel).

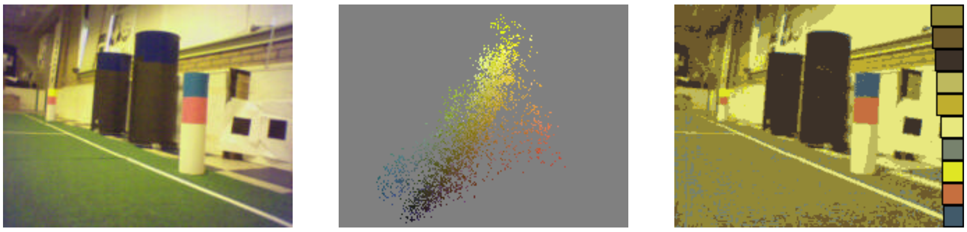


Figure B.2: Before the visual compass can go into work, a color-to-colorclass mapping has to be established. This mapping can be created by the Aibo automatically by using an Expectation-Maximization clustering algorithm. **Left:** Typical raw image as captured by the robot while turning on the spot. The robot extracts in total 100.000 color samples. **Middle:** Visualization of the collected color samples in the color space, prepared for clustering. **Right:** The found clusters are used to segment incoming images.

and both the shutter speed and the camera gain mainly influence the brightness, it seems sufficient to measure the dynamic range of the Y-channel only. The underlying assumption is that the brightness as well as the other two channels have lower noise when the image is well illuminated.

Usually, the dynamic range is defined as the ration between the smallest and the largest possible value of a data source. However, as the camera data contains noise and outliers, it is reasonable to measure the dynamic range as the distance between the 15-percentile and the 85-percentile of the distribution of the Y-channel (see fig. B.1).

When the autoshutter behavior is activated, the robot iterates through all camera settings and computes respectively the dynamic range. As it takes a while before the camera stabilizes on new settings, the autoshutter waits for 15 frames before it captures and analyzes a picture. Therefore the whole autoshutter process takes a few seconds to complete.

B.2 Color clustering

In order to create the color class lookup table (see fig. B.2), the Aibo starts by turning on the spot and collecting a big number of color samples (around 100.000). Only colors above the horizon are taken into account, as panoramic localization later on likewise only learns color class transitions from above the horizon.

The clustering process uses an Expectation-Maximization (EM) algorithm on a Gaussian mixture

Shutter speed	Camera gain	Comments
fast	low	Outdoor sunlight.
fast	medium	Outdoor shadow or indoor sunlight (behind a window).
fast	high	Outdoor cloudy or indoor soccer lighting.
medium	high	Indoor office lighting. Some motion blur.
slow	high	Indoor normal lighting. Strong motion blur.

Table B.1: Expedient camera settings probed by the auto-shutter.

model. Considering the relatively low computational power of the Aibo processor, it was necessary to reduce the sample set significantly but in such a way that the important (and characteristic) color samples do not get lost (by the overwhelming amount of background gray samples).

To achieve this, a color space histogram is created on the fly with the aim to delete color samples from crowded regions. This is achieved with a Monte-Carlo filter. When the corresponding color cube of a color C has the frequency of occurrence $P\left(cube_C\right)$, then the probability that this color is used for clustering is equivalent to $P_{use}(C) = 1/P\left(cube_C\right)$.

The reduced sample set has then still captures the shape of the color space well enough, and can be fed into an Expectation-Maximization algorithm [34] with an underlying Gaussian mixture model. A single color class is therefore is represented in the color space by a Gaussian distribution defined by its weight π, mean μ and covariance Σ.

The algorithm expects a fixed number of clusters, which should be chosen carefully. A too small number might result in only finding the background colors, while a too large number would increase the number of color transitions distributions enormously. Additionally the computation time of the EM algorithm increases with the number of clusters. Experimentally I found ten color clusters leave on one hand enough room also for small clusters to emerge and on the other hand keep the panorama model sufficiently small. However, this number is adjustable and is given further analysis in chapter 5.

The EM algorithm initializes the Gaussian with the mean positioned on existing samples and equal weights and covariance matrices. Then it iteratively estimates the likelihood of all sample points with respect to the current Gaussians in the **expectation step**:

$$q_{ns} \leftarrow p(s|x_n) = \frac{\pi_s p(x;\theta_s)}{\sum\limits_{s'} p(x;\theta_{s'})} = \frac{\pi_s N\left(x;\mu_s,\Sigma_s\right)}{\sum\limits_{s'} N\left(x;\mu_{s'},\Sigma_{s'}\right)} \tag{B.1}$$

and then maximizes these likelihoods by re-estimating the Gaussian parameters in the **maximization step**:

$$\pi_s \leftarrow \frac{1}{N}\sum_{n=1}^{N} q_{ns} \tag{B.2}$$

$$\mu_s \leftarrow \frac{1}{N\pi_s}\sum_{n=1}^{N} q_{ns}x_n \tag{B.3}$$

$$\Sigma_s \leftarrow \frac{1}{N\pi_s}\sum_{n=1}^{N} q_{ns}\left(x_n-\mu_s\right)\left(x_n-\mu_s\right)^T \tag{B.4}$$

These steps have in principle to be iterated until the parameters stabilize – in practice the clusters converge already after a few iterations, and the current implementation simply applies the above iteration 10 times.

With the found parameters, it is possible to determine the color class of a given color by finding the maximum-likelihood cluster. This is however very time consuming, as it involves for each source color the evaluation of 10 (inverse) Gaussians in order to determine its target class. Therefore, after the clustering process, a lookup table for faster access is created. The lookup table is a pre-computed array containing the color classes of all colors. Each of its three indexes (one for each color channel) is six bit wide; therefore the color table has a total size of 262KB.

At each iteration step, the algorithm evaluates recursively the eight corners of the current color cube. When all corners have the same color class, then the complete cube is considered to be uniform and all color cells are assigned to this color class. When however, the color classes of the corners differ, the current cube is split up into 8 sub-cubes (by splitting it in each dimension into two parts), and each sub-cube is evaluated recursively in the same way. To ensure that small clusters are unlikely to be overlooked, only cubes below a certain size are allowed to be filled uniformly (starting from the third iteration turned out to be effective).

This technique proves to be most effective; the resulting color table slightly differs from the fully evaluated version, for example because the round edges of the Gaussians are likely to be pruned when touching a neighboring cube only in the middle. However, experimental comparisons have shown no significant difference, but the advantage in processing time is enormous: the color table based on 10 clusters is usually built within less than 2.500ms, corresponding to a speedup of factor 20. The time needed for color calibration then sums up to 9 seconds (for turning the Aibo a complete round on the spot) and another 4.5 seconds for clustering and building the color table.

The color-to-colorclass mapping is used by the compass sensor (section 3.1), but it has also been shown this approach can be used for automatic color learning with normal landmarks [44]. Together with the autoshutter step, the complete vision system calibrates itself **fully autonomously** within less than 20 seconds.

B.3 Summary

The **auto-shutter** enables the robot to set up its camera in a wide range of illumination circumstances. A robot can use the **automatic color calibration** to create fully autonomously a color-to-colorclass lookup table that is perfectly suited for the current environment. For this purpose, an expectation-Maximization clustering algorithm is applied on large number of color samples taken from the robot's surroundings.

Appendix C

Direct pose triangulation

A totally different (but not implemented) approach would be to exploit the projective distortions that occur naturally in particular in small rooms. Imagine that a robot has two compass sensors that have been trained on two different spots with a fixed distance that is known to the robot (see fig. C.1(left)). Imagine further that the robot is then placed in the center after learning is complete, and looks into a certain direction – of which it evaluates both visual compasses by matching the incoming camera frame with both compass maps.

As the wall and the furniture do not have infinite distance from the robot, the two rotational estimates $R^{(1)}$ and $R^{(2)}$ will be slightly different, and the angular disparity $\Delta R = R^{(1)} - R^{(2)}$ can be used just like in stereo vision to estimate the depth of the room in a particular direction. Once a depth model (or map) of a room is available, a sensor model $p(R^{(1)}, R^{(2)} | x, m^{(1)}, m^{(2)})$ can be formulated that assigns to each reading $z =< R^{(1)}, R^{(2)} >$ a likelihood (given the robot pose x, and the two maps $m^{(1)}, m^{(2)}$)).

This approach directly builds upon the primary output of the visual compass which is in a way conceptually better than relying upon the variance estimates – as multi-spot localization does. The results of compass experiments where the robot has been moved (see fig. C.1(right)) strongly suggest that deriving the distance to the wall or equivalently estimating the robot's position should be possible. To translate this approach in a working implementation is certainly an interesting trajectory for future research.

C.1 Approach

The idea is to exploit the angular disparity between the estimates of two compass sensors, trained at two different spots $T^{(1)}$ and $T^{(2)}$ respectively:

$$\Delta \hat{R}_t = \hat{R}_t^{(1)} - \hat{R}_t^{(2)} \tag{C.1}$$

As the distance $d = \left\| T_t^{(1)} - T_t^{(2)} \right\|$ between the training spots is known, the distances $d^{(1)}$ and $d^{(2)}$ from training spots $T^{(1)}$ and $T^{(2)}$ respectively to intersection point A on the wall can be estimated. The distance from the training spots towards the intersection point on the wall are denoted by $d^{(1)} = \left\| T_t^{(1)} - A \right\|$ and $d^{(2)} = \left\| T_t^{(2)} - A \right\|$ respectively.

To start with, the following trigonometric relation holds concerning the height h of the triangle

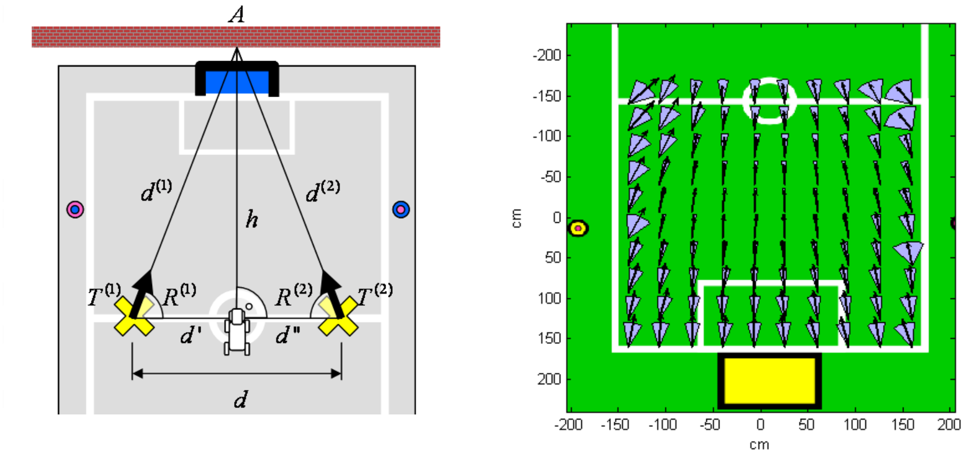


Figure C.1: Motivation of a direct triangulation filter, by using two compass sensors. **Left:** Sketch how two visual compasses could be combined for direct triangulation. When the distance d between two training spots is known, then the robot can triangulate its distance to the wall (from the angular disparity of the two compasses). **Right:** Heading and variance estimates of a visual sensor as reported by the compass filter in an experiment indicate strongly that such an approach could work.

$\left(T^{(1)}, T^{(2)}, A\right)$:

$$\cos R^{(1)} = \frac{h}{d^{(1)}} \tag{C.2}$$

$$\cos R^{(2)} = \frac{h}{d^{(2)}} \tag{C.3}$$

$$\Rightarrow d^{(1)} \cos R^{(1)} = d^{(2)} \cos R^{(2)} \tag{C.4}$$

The triangle $\left(T^{(1)}, T^{(2)}, A\right)$ can be split into two rectangular triangles (by splitting at the height) into two parts $d = \left(T^{(1)}B\right) + \left(BT^{(2)}\right) = d' + d''$. Then for the new sides d' and d'' it holds by geometric inference:

$$\sin R^{(1)} = \frac{d'}{d^{(1)}} \tag{C.5}$$

$$\sin R^{(2)} = \frac{d''}{d^{(2)}} \tag{C.6}$$

(C.7)

From these equations, it becomes possible to compute the distances $d^{(1)}$ and $d^{(2)}$ from the training spot to the intersection point on the wall:

$$\begin{aligned}
d &= d' + d'' && \text{(C.8)}\\
d &= d^{(1)} \sin R^{(1)} + d^{(2)} \sin R^{(2)} && \text{(C.9)}\\
d &= d^{(1)} \sin R^{(1)} + \left(\frac{d^{(1)} \cos R^{(1)}}{\cos R^{(2)}} \right) \sin R^{(2)} && \text{(C.10)}\\
d &= d^{(1)} \sin R^{(1)} + d^{(1)} \frac{\cos R^{(1)} \sin R^{(2)}}{\cos R^{(2)}} && \text{(C.11)}\\
\Rightarrow d^{(1)} &= \frac{d}{\sin R^{(1)} + \frac{\cos R^{(1)} \sin R^{(2)}}{\cos R^{(2)}}} && \text{(C.12)}\\
\Rightarrow d^{(2)} &= \frac{d}{\sin R^{(2)} + \frac{\cos R^{(2)} \sin R^{(1)}}{\cos R^{(1)}}} && \text{(C.13)}
\end{aligned}$$

C.2 Map learning

From such distance estimates $\hat{d}^{(i)}$, the robot could easily create corresponding depth maps $\hat{m}^{(i)}_{depth}$. This could be done for example by collecting a few samples $\left(\hat{d}^{(i)}_1(\phi), \hat{d}^{(i)}_2(\phi), \ldots\right)$ in each direction ϕ, and then computing the mean or median to a depth value for $\hat{m}^{(i)}_{depth}(\phi)$.

C.3 Sensor model

If the robot has now a distance map $m^(i)_{depth}$ at its disposal (for example after it has been learned $\hat{m}^{(i)}_{depth}$ or it has been supplied from outside), the robot can reason about the expected angular disparity $\Delta R(x)$ given a particular pose $x =< \phi, \tau >$: it can evaluate the distance map in a certain direction in order to determine the expected intersection point with the wall A. Then the expected angle from a training spot $T^{(i)}$ to this intersection point A can be computed:

$$\begin{aligned}
R^{(i)} &= arctan\left(T^{(i)} - A\right) && \text{(C.14)}\\
& && \text{(C.15)}
\end{aligned}$$

The expected angles $R^{(i)}$ can now be compared with the measured (estimated) angles $\hat{R}^{(i)}$. By assuming Gaussian noise in the measurements (C.16), a sensor model is constructed in (C.17):

$$\begin{aligned}
\hat{R}^{(i)} = R^{(i)} + \epsilon, \text{ with } \epsilon &\sim \mathcal{N}(0, \sigma) && \text{(C.16)}\\
p\left(R^{(i)} | x, m^{(i)}_{depth}\right) &= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(- \frac{(R^{(i)} - \hat{R}^{(i)})^2}{2\sigma^2} \right) && \text{(C.17)}
\end{aligned}$$

which subsequently can be used by a Monte-Carlo localization method (like a particle filter) to evaluate the likelihood of a pose sample $x =< R, T >$.

Please note that in this form the sensor model only addresses a single visual compass (augmented by the corresponding distance map $m^{(i)}_{depth}$). A single-spot sensor model $p\left(R^{(i)} | x, m^(i)_{depth}\right)$ therefore contains intrinsically ambiguities. These ambiguities can be removed when two or more such sensor models are jointly evaluated:

$$p\left(R^{(1)}, R^{(2)}, \ldots | x, m^{(i)}_{depth}, m^{(i)}_{depth}, \ldots\right) = \prod_i p\left(R^{(i)} | x, m^{(i)}_{depth}\right) \tag{C.18}$$

$$= \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(R^{(i)} - \hat{R}^{(i)})^2}{2\sigma^2}\right) \tag{C.19}$$

If two (or more) underlying sensor models are used, the resulting likelihood distribution is free of ambiguity.

C.4 Discussion

The approach shows interesting parallels to approaches known from stereo vision: there, robots are typically equipped with two (or more) cameras. The images are correlated, and the disparity of a correlation can be used to estimate the depth in this direction.

Here, a stereo map is created by the robot by learning individual maps $\hat{m}^{(i)}$ at different training spots $T^{(i)}$. Two of these maps are sufficient (corresponding to the stereo vision case) to estimate the depth of a room – for example in order to learn a depth map $\hat{m}^{(i)}_{depth}$.

C.5 Summary

In this chapter, a promising approach to pose estimation is elaborated that can be used with two (or more) underlying compass sensors. It exploits the natural projective distortions occurring especially in small rooms. The derived sensor model (C.18) assigns likelihoods to a pose given the rotational estimates of the underlying compass sensors. These likelihoods can then in turn directly be used to update the belief distributions of a localization filter.

Appendix D

RoboCup: UvA achievements

Several teams from the Universiteit van Amsterdam participated over the years in different RoboCup leagues (see table D.1). Emiel Corten participated with his Windmill Wanderers at the second RoboCup competitions in 1998 and won the third prize. Two years later, Matthijs Spaans and Bas Terwijn were members of the national Dutch Midsize team called Clockwork Orange. Here Nomad robots were used. In 2003, Jelle Kok [13] won the Simulation league competitions during the second year of his PhD research. Arnoud Visser supervised several RoboCup soccer tournaments [45] and rescue [6, 22] projects. In 2006, two third places were won: one in the newly established RoboCub Rescue 3D simulation league (by Bayu Slamet en Max Pfingsthorn) and another one in the Technical Challenges of the 4-Legged League (with 27 competing teams). Here the Visual Compass as presented in this thesis was demonstrated.

Outside of the university context, Peter Lith[1] since 2005, has been organizing the Dutch RoboCup Junior competitions (with the leagues Dance, Rescue and Soccer) for kids and teenagers.

---

[1] entrepreneur and guest lecturer at the Universiteit van Amsterdam since 1999

Year	Team	League	Place	Prize
1998 Paris, France				
	Windmill Wanderers	Simulation	3rd	3rd prize
2001 Seatle, USA				
	Clockwork Orange	Midsize	7th	
	UvA Trilearn	Soccer simulation	4th	
2002 Fukuoka, Japan				
	UvA Trilearn	Soccer simulation	4th	
2003 Padova, Italy				
	UvA Trilearn	Soccer simulation	1st	1st prize
	UvA Zeppelin	Rescue robot		
	C2003	Rescue simulation	16th	
2004 Lisboa, Portugal				
	UvA Trilearn	Soccer simulation	7th	
	Dutch Aibo Team	Four-legged	15th	
2005 Osaka, Japan				
	UvA Trilearn	Soccer simulation	10th	
	Dutch Aibo Team	Four-legged	9th	
2006 Bremen, Germany				
	Dutch Aibo Team	Four-legged	6th	3rd place (Technical Challenge)
	UvA ResQ	Rescue 3D simulation	3rd	3rd place

Table D.1: Past and current RoboCup activities of the IAS group at the UvA in RoboCup World Championships