Using a 3-dimensional model for navigation and correction of odometry errors in indoor robotics

J.P. Niemantsverdriet
jelle@niemantsverdriet.nl
#0919462

12th April 2004

external advisor:
dr. ir. T.K. ten Kate (TNO TPD Delft)

internal advisors:
prof. dr. L.R.B. Schomaker (Artificial Intelligence, RuG)
drs. G. W. Kootstra (Artificial Intelligence, RuG)

Artificial Intelligence
University of Groningen

Abstract

This thesis deals with the development and implementation of a system for mobile robot navigation using a camera and a 3-dimensional map of the environment. In this project, a method to compare camera images to images from the virtual environment was developed. This makes accurate control of the position of a mobile robot possible, without specialized constructions like stereo-vision.

To compare the camera images to the images from the virtual environment, a distance transform of the camera image is used. Then, a range of possible virtual images is put over this transformed image, and the minimized values under the lines of the virtual image are looked up.

This method seems to perform well, but it still needs some refinement. In experiments with the robot, the performance is not robust enough yet. Some suggestions about how to realise these improvements are given.

Acknowledgement

I would like to thank a few people who helped me fulfill this research and thesis:

dr. ir. Ton ten Kate (TNO TPD) for the supervision of the project at TNO TPD and his useful ideas on the project in general and the image-processing part in particular.

ir. Jan Baan (TNO TPD) for helping me with *Matlab* and image-processing problems.

ir. Michel van Elk (TNO TPD) for helping me with the VRML part of the project.

prof. dr. Lambert Schomaker (R*u*G) for the supervision of the project from the University.

drs. Gert Kootstra (R*u*G) for the supervision of the project as the second advisor from the University.

Besides these people, I would also like to thank the other people at TNO TPD for their enthusiasm and support. Furthermore I would like to thank my family and friends for supporting me throughout this project.

Nynke of course deserves a more than big *thank you* for her patience and support!

Contents

1 Introduction 1

2 Theoretical background 3

2.1 Odometry . . . 3

2.2 Computer vision . . . 4

2.3 Map representation . . . 5

2.4 Absolute and global localization . . . 6

3 Previous work on vision-based localization 7

3.1 Omni-directional vision . . . 7

3.2 Visual landmark learning . . . 8

3.3 FINALE and its successors . . . 8

3.4 Visual-sensor model . . . 9

3.5 VERONA . . . 9

3.6 Overview . . . 10

4 Why it is necessary to use a 3D map of the environment in robot navigation 11

4.1 Introduction . . . 11

4.2 Design issues and choices . . . 12

4.3 Hardware setup . . . 13

4.4 Software setup . . . 14

4.5 Our method . . . 15

4.6 The environment map . . . 22

4.7 Navigation . . . 22

4.8 Self-localization . . . 24

4.9 Modular design . . . 24

4.10 Other tools . . . 25

5 Experiments 28

5.1 Odometry tests . . . 28

5.2 Match threshold . . . 28

5.3 Complete search . . . 29

5.4 Localization experiment I . . . 29

5.5 Localization experiment II . . . 32

5.6 Localization experiment III . . . 32

6 Discussion 35

7 Conclusion and future work 37

7.1 Conclusion 37

7.2 Future work 38

A Results of odometry tests 40

B Results of localization experiment I 41

C Results of localization experiment III 42

D List of implemented robot control commands 43

References 45

List of Figures

1 Picture of the Nomad Scout robot 13
2 Example of an image processing graph in the Argos-SDK. 15
3 Schematic overview of our method. 17
4 Example of edge-detection on a picture of our hallway . . . 19
5 Example of distance-transform of a simple shape. 20
6 Map of our office that was used to create the VRML model. 24
7 Screenshot of the review utility. 26
8 Simulation of match value ($D_{chamfer}$) on several x and y positions. 30
9 Simulation of match value ($D_{chamfer}$) on several x and *angle* position. 30
10 Example of two positions that could (incorrectly) return a similar match value 31
11 Results of localization experiment I. 33
12 Results of localization experiment III. 34

List of Algorithms

1 Pseudo-code example for very simple obstacle avoidance . 16
2 Pseudo-code example of our navigation algorithm 23

1 Introduction

In this thesis my graduation research project, conducted at TNO TPD in Delft, is described. The project's aim was to create software for a mobile wheeled robot. This robot should be able to navigate reliably in an indoor office environment, guided by input from its visual system. To achieve this goal, we wanted to use a 3-dimensional environment map of the office.

The use of vision for robot navigation is an important research field within Artificial Intelligence. Humans and animals are able to use their visual system for navigation, obstacle detection and recognition of objects, places and people. So far, the performance of artificial agents has not yet approached the performance by biological agents. Most systems still require some kind of modelling (creation of a map of the environment for example) or are only able to perform very small tasks.

Nevertheless, the advantages of a visually guided system are great, even when it is not performing at a human level. Visual input could perform well without artificial markers (for instance transponders in the road) that can be costly and sometimes difficult to put on certain places. It is also a very useful addition to other sensors that are used in robotics, for instance odometry (information from the wheels of the robot).

An indoor robot that is able to use visual information for reliable navigation, can be used for various tasks. It could for instance be used as a guard which drives through a building, as a guide to bring people to specific locations or as an aiding device for elderly or disabled people.

The research question that was formulated in the research proposal is:

In what way can a mobile robot use its camera to refine a visual map of its environment, and use this map to navigate in a robust and reliable way?

Other aspects related to this question are:

- measuring in images
- navigating using a map
- refining the map from the images (from 2 dimensions to 3 dimensions)
- data processing in a robust way

During the research project, the focus changed more to navigation and localization with the environment map, instead of updating the map. This was mainly due to lack of time and was done after advice from my advisors, because otherwise the project would become too big.

The outline of this thesis is as following: Chapter 2 gives a theoretical background on robotics and navigation with or without an internal map. Then, chapter 3 describes previous work on this subject. In chapter 4, the model and the hardware and software that is used in this particular project are explained. Chapter 5 describes the experiments conducted with the robot and the software. Finally, chapter 6 discusses the results and chapter 7 lists the conclusions and suggestions for future work.

2 Theoretical background

To operate reliably and successfully in a real-world environment, a robot (or other form of autonomous agent) has to be able to have and gain knowledge about its environment. This knowledge can have the form of some kind of *map*, but it is also possible to create a system which does not use a complete representation of its environment (see 2.3 for more details on this subject).

A very important task for an agent is to *navigate*. Navigation includes aspects as finding clear paths, avoiding obstacles and calculating the agent's current velocity and orientation [1]. We will refer to this last aspect as *localization*.

In order to complete this *navigation* task, a robot has to have good *perception* of the environment. There are numerous types of sensors which can be used for this *perception*. This project deals with the development and testing of the use of visual information as the primary source for the *perception*.

In this chapter, some of the common terms in robotics and computer vision will be explained and the problems associated with them will be described.

2.1 Odometry

Odometry is navigation based on the motion of the wheels of the robot, and is probably the most widely used method for navigation in mobile robotics[2]. The reason for this is, that it is rather accurate in short distances and is easy to implement (many robots even have onboard hardware which keeps track of the odometry). The main problem with odometry is however, that it gets very inaccurate after some time, because for instance the wheels of the robot may slip: the wheel encoder registers rotation of the wheels, but the linear displacement of the robot doesn't correspond with this rotation. Conseqequently, the position where the robot 'thinks' it is, differs from the actual position. It is rather obvious that this is a rather serious problem when the robot has to fulfill various tasks, because it will not be able to navigate reliably or bump into obstacles.

Errors in odometry can be divided into two groups: *systematic* and *non-systematic* errors. *Systematic errors* are errors caused by unequal wheel diameters, misalignment of the wheels, finite resolution of the wheel encoders et cetera. *Non-systematic errors* are caused by uneven floors, unexpected objects on the floor or slippage of the wheels due to various

causes. The obvious difference between these errors, is that systematic errors accumulate constantly, while non-systematic errors can appear unexpectedly. Thus, on clean surfaces, systematic errors have a great influence on odometry, while on more rough terrain the non-systematic errors will have more impact.

In the office environment which is dealt with in this project, systematic errors will probably give the largest problems. In [2] the authors describe a method to quantify these errors, which is used in this project. This method, called Bidirectional Square-Path experiment, requires the robot to drive a square path in two directions. After completing each square, the distance from the starting point is measured. This distance gives a measure for the *systematic error* in the odometry. The test has to be performed in two directions, because otherwise two mutually compensating odometry errors can compensate each other and thus give the incorrect impression that the odometry is performing good.

In 5.1, the results of this experiment are described.

2.2 Computer vision

In [3], the goal of *computer vision* is defined as: to make useful decisions about real physical objects and scenes based on sensed images. Often it is necessary to create a sort of model or representation of the object from the image, to be able to make decisions about these objects.

Computer vision can be divided into the following aspects:

- *sensing:* The process of obtaining images from the real world
- *encoded information:* The information from the 3-dimensional world that is encoded in the images (such as geometry, texture, motion, et cetera)
- *representations:* How should the descriptions of objects and their relationships be represented in the system?
- *algorithms:* The methods used to process the images and to construct the descriptions of the world

This project will deal with all these aspects, but the focus will be on the last three aspects. This will hopefully become more clear in the next chapters.

2.3 Map representation

In [4], indoor mobile robotics with vision-based navigation is divided into three groups:

- *map-based navigation:* these robots rely on user-created models of their environment
- *map-building-based navigation:* these robots use sensors to construct a model of the environment, and afterwards use this model for their navigation
- *mapless navigation:* these robots have no representation of their environment

Although the navigation methods without any knowledge about the environment (for instance behaviour based robotics [5]) have been quite successful, many authors think that some kind of a priori knowledge will be necessary to complete more complex tasks [1], because even a simple task like finding a target and bringing it back to the starting point requires some internal representation of the environment, which is very similar to the map in the so-called 'classical AI'.

This project deals with the first category: *map-based navigation*. The map can be represented in several ways. The earliest models used so-called *occupancy maps*, 2-dimensional projections of objects (in the horizontal plane, much like an ordinary map of a room looks like).

Other approaches keep track of uncertainties of the position of objects, to deal with errors in the measurement of their coordinates. In all approaches, the general procedure in vision-based localization consists of the following four steps:

- *acquire sensor information:* acquire the images from the camera
- *detect landmarks:* this usually means filtering actions like edge detection, smoothing, segmenting regions
- *establish matches between observation and expectation:* this step tries to compare the observed landmarks to the map stored in the database
- *calculate position:* if a match is found, the information from the database is used to calculate the actual position of the robotdsf

The third step is - not surprisingly - often considered the most difficult one [4, 6]. This step requires some kind of search algorithm, which hopefully can be limited by a priori knowledge or by other criteria.

2.4 Absolute and global localization

One thing most practical implementations of *navigation* have in common, is that they rely on some form of *localization*: the ability to have or gain knowledge of the agent's current position.

Localization can be divided into three subgroups: *absolute localization, relative localization* and *localization derived from landmark tracking* [4]. In *absolute localization* the initial position of the robot is unknown. The opposite, *relative localization*, assumes that the initial position of the robot is known (approximately). In this way, the robot only has to keep track of the movements from the beginning. This can for instance be done by knowledge about the uncertainty in movement, as is done by the FINALE system [7], which will be described in more detail in the section on previous work.

In *localization derived from landmark tracking,* natural or artificial landmarks (visually striking objects in the environment) in the camera images are used to find the position of the robot. Examples of this method are systems which keep track of the movement of these landmarks in consecutive camera images, and deduce the movements of the robot from these movements.

This project uses *relative localization:* the robot starts (approximately) at a specified position and keeps track of its movements. This approach is chosen, because it greatly limits the search space. The prerequisite that the robot has to start on a known position, is one that is not considered a problem in the tasks the robot has to fullfil.

The project uses also some aspects from *landmark tracking*-based methods, because the robot searches for natural landmarks in the environment (which are compared to the landmarks in the model). This will become more clear in chapter 4.

3 Previous work on vision-based localization

Vision is often considered one of the most challenging subjects in robotics and Artificial Intelligence. The development of a system that can sense its environment by looking at it, just as humans and animals do, is often considered a necessary step towards a real intelligent agent. The hardware used (cameras, computers et cetera) has greatly improved over the last decades, but the results in computer vision show that just adding more processing power or a better camera are not the only way to a good solution.

In this chapter, some other projects on vision-based localization or navigation will be discussed. There are many different approaches to this problem. This chapter shows some projects which use a (partly) similar approach to our approach, but will start with a few that handle the problem differently (both in hardware and software).

In the next chapters our approach will be described. It will become clear, that various aspects of these projects are used in this project.

3.1 Omni-directional vision

In [8], the authors use an omni-directional camera (a vertically oriented camera with a hyperbolic mirror mounted in front of the lens) on their mobile robot. They transform the images from the camera to 360 degrees panoramic images, and then extract features from these images to reduce the dimensions of these images. The images are then mapped to the appropriate locations. To match new images, they use Principal Component Analysis and a supervised projection method. According to the authors, the localization error is about 40 cm if 15 features are used and the environment is represented with 300 training samples. The authors have experimented with data from the MEMORABLE robot database, a database of about 8000 robot positions with associated measurements from sonars, infrared sensors and omni-directional camera images, but they have also used images from their own office environment.

An advantage of this approach is, that the robot doesn't need any a priori knowledge about the environment. On the other hand, it has to be trained when it is in a new environment, which could take quite some time and sometimes gives unexpected results.

3.2 Visual landmark learning

Another approach is to look for *landmarks* in the camera image. Such a project is described in [9], where the authors try to use biologically inspired methods to learn visual landmarks. The robot tries to decide which landmarks are useful and reliable as navigation aids. These landmarks are templates which are uniquely identifiable in their neighbourhoods.

The authors conclude that inspiration from biology can be useful in landmark learning, but that it needs to be well formalized for an efficient implementation in artificial agents. They state that a definition for landmark reliability has to be developed and that a measure for the quality of the learning phase has to be introduced.

The use of a biologically inspired method seems to be a useful approach, but these approaches aren't very scalable to more difficult tasks (as the authors also conclude).

3.3 FINALE and its successors

The FINALE system [7] is an example of a map-based localization system. The system uses *relative localization*, so the robot's starting position is always known. The system uses a geometrical representation of the environment, and a statistical model of the uncertainty in the position of the robot.

The robot starts to move from a known position, and the uncertainty in position is updated by an empirically constructed model. When the robot starts to determine its position, this uncertainty is projected into the camera image. Then the model of the hallway is used to determine which position fits to the actual camera image. The search space is limited by the uncertainty in the robot's position, so the system doesn't have to check all the possibilities. When a match is found, the position of the robot is updated.

To determine the match between the camera image and the model of the environment, the lines are represented in Hough space. Then, the landmarks (lines) are matched with the help of a Kalman filter.

The purpose of a Kalman filter is, to estimate the state of a system from measurements which contain random errors. This is done by minimizing the mean of the squared error.The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown [10].

According to the authors in [4], the system navigates at an average speed of 17m/min on ordinary PC-based hardware (Pentium II 450 MHz) and the self-localization routine checks an image in less than 400 ms.

The same authors revised their methods in [11], which discusses a system that is capable of continous self-localization. This revised system is also capable of obstacle-avoidance by its visual system.

This method requires the development of an a priori map, but can then be used 'out of the box' (so there is no long and sometimes unpredictable training period required). The good results which the authors claim are interesting.

3.4 Visual-sensor model

In [12, 6], Fichtner develops a probabilistic sensor model for camera-pose estimation in office environments. He claims the model is very suitable for use in sensor fusion approaches. He uses a wireframe representation of the hallway and tries to find a suitable error function to determine the best match between the camera image and the model.

He has compared several methods for this error function, for instance centred match count, grid length match and nearest neighbour. These methods are tested on artificially created images of the office and later on real images. Finally, the methods are tested on a real robot setup.

This project gives a good overview of some methods to compare an environment map to the real world.

3.5 VERONA

In [13], the authors describe the Virtual Environment for Robot Navigation (VERONA). This environment consists of a 3D environment model, used to facilitate control of mobile robots. The environment offers a 3-dimensional view on the scenery from any viewpoint necessary (for instance bird's eye view or side views). The system is also capable of generating its own maps. The walls are constructed by ultrasound measuring. Later, snapshots from the camera are used to apply appropiate texture on them.

The map can also be used for vision-based self-localization of the robot, because it is possible to compare the camera view with the virtual view. To compare these images, vertical lines are used. These lines are detected by Hough transformation, the matches are then computed by cross correlation or minimization of absolute differences.

This project is interesting for its use of VRML as a method to control the robot and for its automatic texturing of walls. The approach which is used to compare the model to the camera image is quite similar to the method used in [12, 6].

3.6 Overview

The above articles list some very different approaches to navigation and localization in robotics. A few conclusions can be drawn from them:

- advanced tasks require some kind of modelling of the environment
- there are multiple ways to compare the model to the actual environment
- most approaches integrate or use the results from multiple sensors
- a 3-dimensional model (like VRML, used by VERONA) gives some interesting possibilities

These are a few of the conclusions and thoughts which have been considered in the design process. The next chapter will show how these prerequisites have led to design choices and implementation.

4 Why it is necessary to use a 3D map of the environment in robot navigation

4.1 Introduction

As concluded in chapter 3, advanced navigation by vision requires some kind of knowledge about the environment. An agent needs to have some kind of environment map as a basis for its reasoning about the acquired sensoric data. Systems without a kind of map give promising results on easy tasks, but to make them scalable to more complex tasks, the agent needs to have some kind of representation of the environment [1].

A system that can perform reliably in a real world environment needs to be robust and flexible, because the real world is a very demanding environment. In [1], the five properties of the real world are defined as follows:

- *inaccessible:* the agent has to deal with imperfect sensors and can only perceive stimuli that are near the agent
- *nondeterministic* (from the agent's viewpoint): the wheels of the robot can slip, the batteries run down, et cetera. So it is never sure if an intended action is going to have the desired effect.
- *nonepisodic:* the effects of an action change over time
- *dynamic:* the robot has for example to know when it is useful to act immediately or to wait and calculate other possibilities
- *continuous:* this makes it impossible to enumerate a finite set of possible actions

Considering these conclusions, we came to the following demands for our system:

- the system had to use an environment map for reliable and accurate localization
- the system needed a structure which made fast reactions to changes in the world possible (so not all navigation could be done on the static map, because there will be (moving) objects which are not on the map).
- the system should not require some kind of artificial markers on the environment

The promising results of some of the research projects described in chapter 3, as well as research wishes at TNO TPD, led to the use of a 3-dimensional model of the environment as a basis for our system. A 3-dimensional model gives the possibility to model natural landmarks and to compare the sensoric data (from the real world) directly to the model. A 3-dimensional model also gives some extra advantages, which we will review in the next sections and in the future work section.

4.2 Design issues and choices

In this part some choices we made will be highlighted. The next sections will give more detailed descriptions on the hardware and software we have used and developed.

4.2.1 Environment model

Using the prerequisites defined in 4.1, the possibilities of 3-dimensional modelling were researched. We decided to use the Virtual Reality Modelling Language the WEB 3D consortium [14] (VRML) files for our environment map. VRML is at this moment the standard for modelling 3-dimensional worlds for all kinds of occasions, for instance building virtual worlds or presenting buildings or devices.

In VRML, it is possible to define geometry, transformations, attributes, lighting, shading and texture of objects. Objects can be defined using simple geometric shapes (cubes, cylinders, et cetera) or by connecting coordinates. These objects can then be transformed and rotated in any direction. It is also possible to set the visible properties of objects, such as colors or textures. And finally, a virtual world can be completed by placing lights on various positions.

Another aspect of VRML is the possibility to 'walk' through a virtual environment with the mouse or keyboard. This really completes the sense of a virtual world where a user can walk and look around. Another possibility is to define a viewpoint on a certain position. This is the option which will be used in this project, as will be explained later.

4.2.2 Comparing model to camera image

The method used for localization in this project, is based on the assumption that it should be possible to compare images from the real RGB-camera on the robot with images taken by the 'virtual' camera in the VRML-model. This approach makes it possible to use the image from

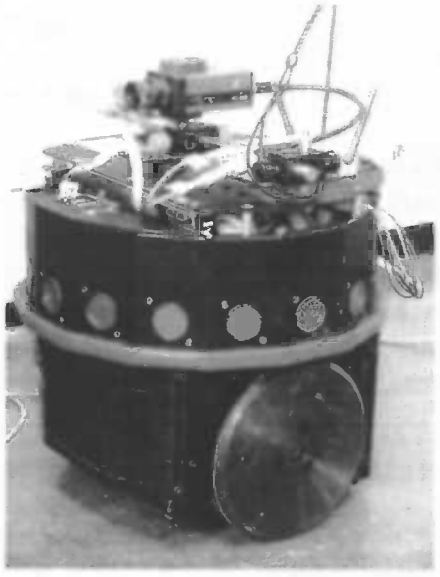


Figure 1: Picture of the Nomad Scout robot

the VRML-model as a kind of template of features, which have to be found in the real image.

4.3 Hardware setup

The robot that is used is a Nomad Scout [15] from Nomadic Technologies (see figure 1). This is a 2 degrees of freedom differential drive robot controlled by a 68332 controller board and an onboard PC. The robot is equipped with a range of 16 ultrasonic sensors and a touch-sensitive bumper. The diameter of the robot is 41 cm and its height is about 35 cm (without camera).

The PC setup is modified to be more compliant with today's standards. The robot now uses a Pentium III processor running on 1,3 GHz and has 512 MB of RAM onboard. It is also equipped with a PCI-framegrabber, which connects the PC to the RGB-camera mounted on top of the robot. The PC is connected to the controller board by the serial port.

To provide a connection to the network, the robot has a wireless network card (10 Mb speed). As it has an onboard PC, the robot can drive autonomously, but the network connection is useful to review results and enter commands from another PC.

4.4 Software setup

4.4.1 Operating system

The operating system used is Microsoft Windows 2000. This operating system was chosen, because the software toolkit used for the image analysis is based on DirectShow [16], which is only available for Windows. By using Windows, some of the possibilities of (Real Time) Linux can't be used, but as our application doesn't need very much computational resources, this is not considered a disadvantage. The Argos-SDK gives some other advantages which will be discussed in 4.4.2.

Because the software from Nomadic Technologies for the Nomad Scout is Linux based, a new interface to the robot had to be written.

4.4.2 Image analysis framework (Argos-SDK)

The image analysis framework (called Argos-SDK) offers a connection between *DirectShow* and *Matlab [17]* (an advanced tool for doing numerical computations with matrices and vectors with its own scripting language). The framework grabs images or movies in a very fast way using *DirectShow.* Filtering can be done by *DirectShow,* but it is also possible to transfer the image to *Matlab* to perform advanced processing on them. While *Matlab* is busy analysing the images or controlling the robot, the grabbing of images in *DirectShow* continues (because this part runs in a separate thread and thus is not influenced by *Matlab*).

The framework allows rapid prototyping and development, because it is very easy to change the image processing graph (see figure 2 for an example of such a graph) and to try some algorithms in *Matlab.* As mentioned in 4.4.1, the combination of platform and operating system is not the best possible in terms of (real time) performance, but this is not considered a problem at this project.

4.4.3 VRML framework (Visserver)

To be able to use VRML-files (which we use as our 3-dimensional environment map), we use *Open Inventor* [18]. This is an object-oriented toolkit for developing interactive, 3D graphics applications. A *Matlab* toolbox to access *Open Inventor* (and thus control VRML files from *Matlab*), called the *Visserver,* has already been developed by TNO TPD.

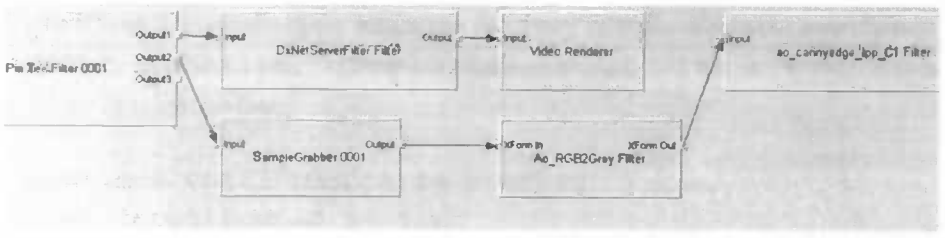


Figure 2: Example of an image processing graph in the Argos-SDK. Each block represents a *DirectShow*-filter.

4.4.4 Robot control

As mentioned in 4.4.1, a new interface to the robot had to be developed, because the original software is Linux based. Because all image analysis had to be done in *Matlab*, the decision was made to control the robot from *Matlab* too. To achieve this, so-called Matlab Executable (or MEX) files are used. These files can contain C-code (or other language, but in this case C was used) and can after compilation be executed from *Matlab* as ordinary *Matlab*-files or functions (see [19] for more information about MEX-files).

Already available Nomad software for Windows [20] was used as a basis for these MEX-files. Using this software, most of the functions where transcribed to *Matlab* functions. Thus the robot's motors and the ultrasonic sensors as well as the touch sensor can be controlled from *Matlab*. All these rather simple functions (like 'move forward with a certain speed') can be combined in a *Matlab* script, to make the robot fulfill a certain task. An example of such a script is listed (in pseudocode) in algorithm 1.

These scripts can also contain functions to control the image-processing toolbox (Argos-SDK) and the VRML toolbox. Thus the entire behaviour of the robot and all sensors can be controlled from *Matlab*.

This architecture gives the possibility to develop certain behaviour in a fast way by putting the desired behaviour in a script. More complex behaviour can be achieved by combining scripts. It is of course also possible to control the robot by typing in the functions at the *Matlab* prompt.

4.5 Our method

The method used for localization in this project, is based on the assumption that it should be possible to compare images from the real RGB-

Algorithm 1 Pseudo-code example for very simple obstacle avoidance

```
% Note: in Matlab, commentlines start with a %
% like this line

% continue until bumper is pressed
while ( rbgetBumper == 0 )
    % read sonar values
    sonars = rbgetSonars;
    if ( sonars > 50 )
        % obstacle at a certain distance: then we
        % move forward
        rbMove(50,50);
    else
        % otherwise move backwards
        rbMove(-20,-20);
    end;
end;
```

camera on the robot with images taken by the 'virtual' camera in the VRML-model. The features in the model are used as landmarks which have to be found in the real image.

The process of our approach is shown in figure 3. The wheel encoders (odometry) of the robot are used as an initial guess for the position of the robot. Then the viewpoint (the 'virtual camera') is placed in the VRML model on that estimated position and angle. If the guess is good enough, the view from the virtual camera will be nearly identical to the view from the real camera on top of the robot.

First, some image-processing steps conducted on the camera images will be discussed. Then the comparison between the virtual and the real image will be explained.

4.5.1 Image processing

The RGB-camera and the framegrabber on the robot are able to deliver 32-bit RGB images at a framerate of 30 frames per second. This is a far higher framerate than is necessary for our purposes. Therefore, the framerate is set to 5 frames per second (which is still a bit higher than needed, though) at a resolution of 320 by 240 pixels.

Most image-processing is done in a Directshow graph (see figure 2 for an

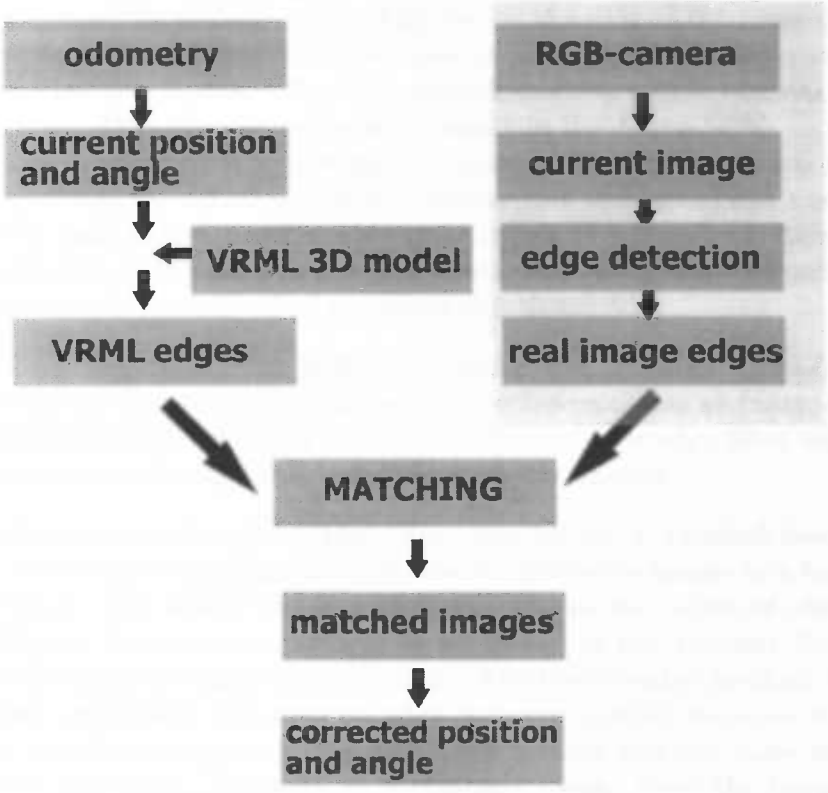


Figure 3: Schematic overview of our method. The boxes inside the greater box represent the parts that are done by *DirectShow*

example) and thus keeps on running in a thread separate from *Matlab*. The following transformations in Directshow on the images from the camera are used:

- *Camera calibration:* This first step is required to remove any distortion caused by the camera lens. The virtual camera in the VRML model is 'ideal', which means that for example all straight lines really look straight. In our real camera, such lines are often distorted (especially when they are on the side of the image). To compensate for this effect, we have to calculate a so-called *camera-calibration matrix*. This is a transformation which corrects this effect. This filter was already present in the Argos-SDK, we only had to calibrate it for our specific camera. This calibration is done by moving a chessboard with a known size in front of the camera. The calibration filter is able to calculate the distortion (because the size of the chessboard is known), and the calculated calibration matrix can be used to correct this distortion.

- *Transformation from RGB to grayscale image*: This is the conversion of a Red-Green-Blue image (so a color-image) to an image with only levels of grey. This is necessary because the edge filter we use (see the next step) only handles grayscale images.

- *Canny-edge detection*: This filter (see [3] for a detailed description of this algorithm) transforms the grayscale image to a binary (black and white) image which only shows the edges of objects. Figure 4 shows an example of an image of our hallway that is transformed using this algorithm. The Canny-edge method is often considered the optimal edge detector, mainly because it has a very low error rate (few edges are missed and few false edges are detected). It works in a multiple steps: First the image is smoothed by Gaussian convolution. Then gradient magnitude and direction are computed for each pixel. The direction of the gradient is used to suppress pixel responses that are not higher than the two neighboring pixels on either side of it (*non-maximal suppression)*. The two neighbours of a pixel that have to be compared are found by rounding off the computed gradient direction to yield one neighbour on each side of the center pixel. Then, the high magnitude contours are tracked. The tracking process is controlled by two thresholds (T1 and T2): Tracking can only begin at a point on a ridge higher than T1. Tracking then continues in both directions out from that point until the height of the ridge falls below T2. This procedure helps to ensure that noisy edges are not broken up into multiple edge fragments.

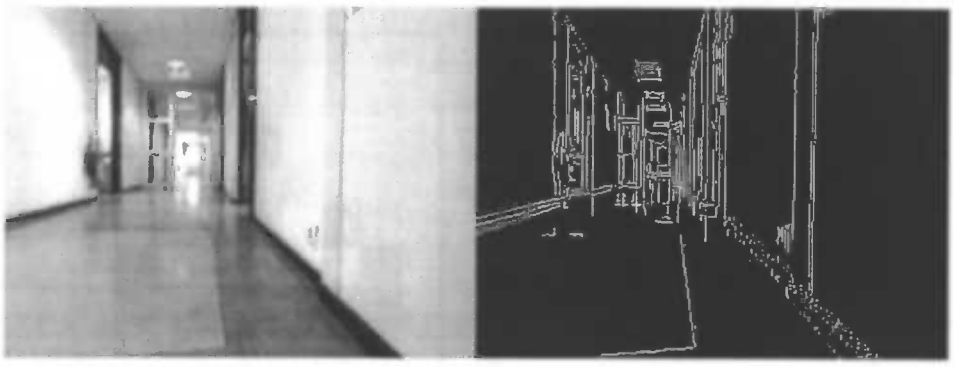


Figure 4: Example of edge-detection on a picture of our hallway. The left picture shows the normal camera image. The right picture shows the same picture, after the Canny edge detection algorithm was performed on it.

After this step, the image is transferred to *Matlab*. In *Matlab*, another transformation is done:

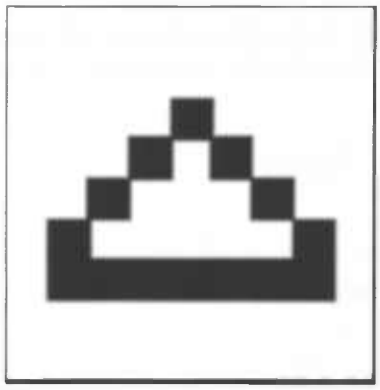
- *Distance transform:* This transformation calculates for each position the distance to the nearest pixel that is 'on' (it is normally only applied to binary images). So the positions where lines are found get value 0, the pixels next to the lines get value 1, et cetera. In figure 5, an example of the distance-transform of a simple shape is shown. In our application, the Euclidean distance is used. The Euclidean distance transform between two points (*x1,y1*) and (*x2,y2*) is defined as follows:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (1)$$

  Using the standard *Matlab* implementation (*bwdist)*, this value is calculated for each point to the nearest nonzero pixel. The calculation for each point is not really necessary (because only the pixels where we expect to find edges are used), but this is done due to implementation issues (a partial distance transform is not easily implemented in *Matlab*).

The steps mentioned above are all processed on the image from the RGB-camera, but some transformations are also done on the image from the virtual camera.

- *Rendering of image:* This step is the actual 'taking' of the picture from the VRML model. When the virtual camera is on the desired position, a snapshot of the view is taken and this image is



4.2	3.6	2.8	2.2	2.0	2.2	2.8	3.6	4.2
3.6	2.8	2.2	1.4	1.0	1.4	2.2	2.8	3.6
2.8	2.2	1.4	1.0	0.0	1.0	1.4	2.2	2.8
2.2	1.4	1.0	0.0	1.0	0.0	1.0	1.4	2.2
1.4	1.0	0.0	1.0	1.4	1.0	0.0	1.0	1.4
1.0	0.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1.4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.4
2.2	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.2

Figure 5: Example of distance-transform of a simple shape. The left picture shows a binary image (black pixels have a value of 1, white pixels have a value of 0). The right picture shows the distance-transform of the image: each position shows the distance to the nearest black pixel.

transferred to *Matlab*. This is done using the *Visserver* application, which we have mentioned earlier.

- *Transformation from RGB to grayscale image:* Same step as done with the real camera image. Necessary for the edge-detection.
- *Canny-edge detection:* The edges of our virtual image have to be used, so the Canny edge algorithm is applied to this image as well.

At this point, two images are available: one binary image of the edges of the VRML model, and one distance-transformed image of the edges of the real world. These two images now have to be compared and a measure representing their match has to be found.

4.5.2 Image matching

To define the match between the two images, the following approach is used. The virtual image is used as a template, because this view is the expected view on the robot's current position. So for each pixel that is part of a line (each pixel that has the value of 1, because we are using a binary image), the value on that same location in the distance-transformed real image is checked.

If the two images are equal, these values should be low (because the distance to the nearest pixels is low on these locations, which means there

are pixels on approximately the same locations). In [21], this distance between the template image T and the original image I is referred to as the *chamfer* distance, which is defined in :

$$D_{chamfer}(T, I) \equiv \frac{1}{\mid T \mid} \sum_{t \epsilon T} d_I(t) \tag{2}$$

where $\mid T \mid$ denotes the number of features (in our approach: pixels) in T and $d_I(t)$ denotes the distance between feature t and the nearest feature in I.

To get an usable measure for these match, the mean of all pixels (so of the whole image) is used. If this match-value is below a certain threshold, the two images are considered to be good matches:

$$D(T, I) < \theta \tag{3}$$

The exact value of this threshold θ has to be found experimentally. We will explain these experiments in 5.2.

When two images are being compared, both images are also saved to disk. The computed match-value and the current location are also logged to a file. These files can be used to review the performance of the system afterwards.

4.5.3 Searching through the virtual environment

If the calculated match value on a certain position doesn't exceed the specified threshold θ, the robot is apparantly not on the location where it thinks it is. A search through our virtual environment then has to be conducted (so the virtual camera has to be moved), until a location where the images do match is found. Because the robot's position is checked every 1,5m and the starting position is known (see the part on *incremental localization)*, this search space hopefully doesn't have to be very large. The search is in fact a minimalisation of the function described in 4.5.2, because we want the result of this function to be as low as possible.

The search is based on the heuristic that our hallway is a rather simple geometric environment. In our first experiments, the model was only used in one direction (the width of the hallway). To find the best match in that direction, the virtual camera only had to be moved from right to left in the width of the hallway. When this method was later extended to the other direction and to the angle, these directions were simply

added. So the camera first moves from left to right to find the best match in $y$direction, then forward to backward to find the best match in x direction, and finally the virtual camera is rotated to check the robot's *angle* (see figure 6 to get a better understanding of the x and y directions). This method seemed to get quite accurate results and is a rather fast way to search through our 3-dimensional search space (x, y, and *angle*). As will be described in the sections on the experiments and the conclusion, sometimes some problems in the x direction occur due to repetitive elements in the hallway.

Something that has to be noted, is that we conduct a 1-dimensional search for 3 times, instead of a true 3-dimensional search (x, y and *angle* all simultaneously). While doing the minimisation search, the robot could get stuck in a local minimum and thus propose a less optimal position.

4.6 The environment map

As a basis for the model, an ordinary map from our office is used (see figure 6). This map already gives 2 dimensions of the model, so after modelling this map only the walls have to be 'raised' to complete the model. The final result can be seen in the screenshot of the review utility in figuroe 7 . Because only the edges will be used, realistic textures on the walls aren't necessary. They would only decrease the performance and are of course useless when only edges are used.

The only aspects that are modelled, are the squares on the floor and the doors. These objects are modelled using standard colors and shapes, and thus don't have any realistic textures on them.

4.7 Navigation

The robot keeps track of its own position (x and y) and angle. A simple algorithm to use coordinates from this coordinate system as a target to drive to was developed.

The algorithm is listed in pseudocode in algorithm 2.

This is of course a very simple algorith, but it is useful so far in the current environment. Another algorithm can be easily implemented, because this only involves replacing one *Matlab* script. Not mentioned in this example is the detection of obstacles and the use of the camera and the VRML model. Something about the cooperation between all parts will be explained in the next section.

Algorithm 2 Pseudo-code example of our navigation algorithm

```
while ( distance > 5 )
    if ( distance > 100 )
        % move pretty fast
        rbmove(90,90);
    else
        % otherwise: move slower (we are getting near)
        rbmove(30,30);
    end;

    [curX,curY] = rbgetpos; % get current position

    % check distance to goal
    distance = sqrt((targetX-curX)^2 + (targetY-curY)^2);

    if ( ~exist( 'tempdistance' ) )
        % help variable
        tempdistance = distance;
    end;
    if ( distance > 50 )
        % we aren't near our goal yet
        if ( abs(tempdistance - distance) > 150 )
            % if we have driven 150 cm's, we check
            % our angle

            % turn to target again
            rbturnto(targetX, targetY);

            % reset distance
            tempafstand = afstand;
        end;
     else
        % we are getting close
        if ( abs(tempdistance - distance) > 10 )
            % if we are really close: correct angle
            % more often (every 10 cm's)
            rbturnto(targetX, targetY);
            tempdistance = distance;
        end;
    end;
end;
```

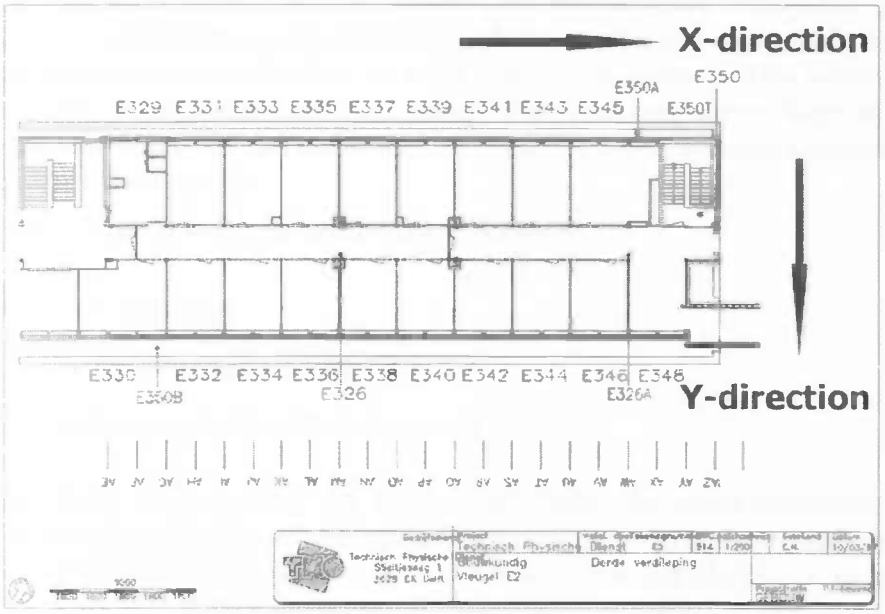


Figure 6: Map of our office that was used to create the VRML model. The x and y directions are marked.

4.8 Self-localization

When the robot has driven about 150 cm, the navigation algorithm checks if the robot is still facing the right direction (see also algorithm 2). On this position, the current camera image is also checked against the VRML model. If this match is not satisfying, the virtual camera is moved and the system tries to find a better match. If multiple matches which are below the threshold θ are found, the minimum is chosen.

After this process, the location of the virtual camera on the best match's position is used as the new location for the robot. If no suitable matches are found at all, the robot continues using the old values from the odometry. This can sometimes happen when for instance someone is blocking the camera view. When the robot has driven another 150 cm's it will start another check and hopefully get a match by then.

4.9 Modular design

Because all parts of robot control are in separate *Matlab* files, it is possible to combine these basic elements into more complex behaviour. A somewhat hierarchical structure in priority , which is inspired by the

subsumption architecture [5] is used. This architecture breaks the processing of information into several modules, which act more or less independently from eachother on both input and output. The low-level behaviour is implemented in a rather easy and very direct way; more intelligent functions are performed in a higher layer and don't interfere with the other layers.

The following priority in behaviour is defined:

1. avoid obstacles
2. navigate towards a goal
3. localise yourself with the camera

Thus, if for some reason the localization fails, the robot continues to drive to its goal (although the errors in odometry will accumulate). And if navigation is not possible, the robot will still avoid obstacles, as this is the most low-level behaviour.

Because of this architecture, it is also possible to implement new behaviour rather easy. A new *Matlab* script could easily be written and placed somewhere in this hierarchy.

4.10 Other tools

Some other tools for the robot, some of which are useful for analysing the results, some of them for other purposes, have been developed.

4.10.1 Review utility

To be able to review the results of the robot, a script with which it is possible to review the results of the localization module was developed. A single snapshot can be watched, but it is also possible to generate a movie from all snapshots. In a single window all relevant images are shown: the real camera image, the VRML camera image, the VRML model viewed from above (to see where the robot is localised), the combined view of the real and VRML image and the match value at that position (see figure 7).

4.10.2 Webserver

Some of the scripts are integrated with a webserver. There is not much functionality yet, but this can be easily extended. The robot's webserver

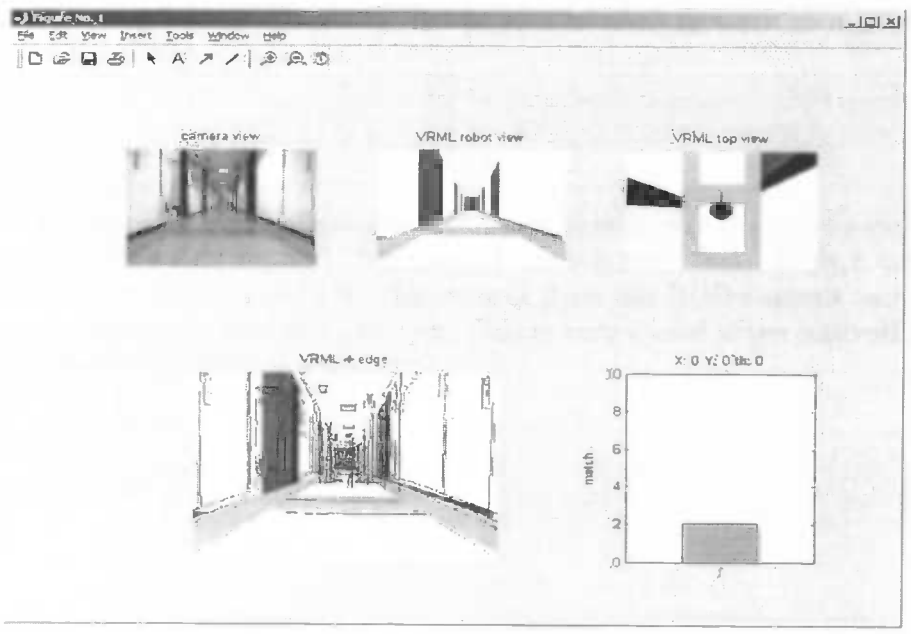


Figure 7: Screenshot of the review utility. Upper left is the real image, the upper middle image shows the view from the virtual camera. Upper right is the view of the model from above. Below left is the combined view of the real and VRML image and the graph on the lower right shows the match value for this position.

shows the last snapshot that was taken, and allows viewers to take a new snapshot. In the future it could be possible to give more specific commands by this web interface.

4.10.3 Interaction with environment

The robot is also able to have some (very basic) interaction with the environment, by using its onboard speakers. A few sentences and words have been sampled. The robot uses these to announce its location and the direction where it is going. It can also be used to warn people who are in the way to step aside.

4.10.4 Net viewer

The webserver already gives the possibility to look over the robot's shoulder while it's driving, but this doesn't refresh very fast. With a component from the Argos-SDK, the images from the RGB-camera can be streamed directly over the network. This is very useful when controlling the robot from another computer.

5 Experiments

5.1 Odometry tests

To get some grip on the errors in odometry, the Bidirectional Square-Path experiment, described in [2] has been conducted. This test requires the robot to drive in a square path in two directions (both clockwise and counter-clockwise). After each square, the distance from the starting point is measured. This distance gives a measure for the systematic error in odometry. The test is performed in two directions, because otherwise two mutually compensating odometry errors can compensate each other.

After a number of repeats of the test, the error in odometry is expressed in a single numeric value. This value is computed by first calculating the center of gravity of the errors in both directions, and finally taking the largest value of these 2 numbers. The test doesn't use the mean of both directions, because in practical applications one has to deal with the largest possible error, and using a lower value would thus give undesirable results.

In the experiments, this test was conducted by driving a 1m by 1m square for 5 times in both directions. These tests return an error value of 3 cm. The mean systematic error in odometry is thus about 0,75%.

In A, all results of these experiments can be found. The systematic error that was found, has been used for defining the size of the movements of the virtual camera, when it is searching through the model.

5.2 Match threshold

In 4.5.2 the method to get a match value ($D_{chamfer}$) between the real and virtual image is explained. To find an accurate threshold (θ) for the match value, some tests were performed.

This threshold is mostly useful for rejecting images that do not match the templates from the virtual image. The search always returns the minimal distance found (and thus the image that most closely matches the current template), but the threshold defines if this match is considered useful or not.

To gain a thorough knowledge of this value, we have conducted some experiments with images taken on known positions. By moving the virtual camera around this known positions, the change of the match value was found. After these experiments, images with a match value lower than 3 were considered to be equal (and thus taken on the same position

in both the virtual and the real world). The goal was, to be able to match images with a maximum error of 2 cm. As mentioned, the threshold will probably not influence this accuracy very much, because it is only used for rejecting images.

Match values higher than 3 resulted in too much false positive matches (the match value indicates the template and image represent the same location, but in fact they do not). The change in match value when the virtual camera is moved (in another experiment) can be seen in figure 8. This figure shows, that a threshold of 3 will reject the images taken at a greater distance than 2 cm from the actual position.

5.3 Complete search

In the initial stage of the project, some exhaustive searches on fixed positions have been performed. The robot used a snapshot from some location and then put the virtual camera on all possible positions around this position (so not by moving first in x-direction, then in y-direction and then the *angle*, but by putting the virtual camera on *every* possible position). This is an obviously very expensive search (in terms of computational costs), but it was very useful to get a better understanding of the variations in match values on several positions.

This experiment clearly showed the possibility of getting in a local minimum, when searching in 3 directions consecutively. Nevertheless, it is impossible to do such a search every time, so we tried to minimise this problem by choosing the best performing search order. This order was determined by varying the possible search order and reviewing the results. Our initial guess, that the best option would be to start with a search in the y-direction, turned out to be correct.

A (simplified) graph showing the results of this experiment is in figures 8 and 9. Our original figure showed both x, y and the *angle*, but this is impossible to show in a meaningful way on paper. Especially figure 9 showed, that the match value can be (incorrectly) low when the virtual camera is moved to the left or the right, and looking under a certain angle. Figure 10 shows an (exaggerated) example of such positions which could incorrectly return a similar match value (which was not expected, because the viewing angle is quite different). As mentioned, by choosing a specific search order, this problem is hopefully minimised.

5.4 Localization experiment I

Tests with and without the use of our model have been conducted (regarding the design described in 4.9: level 3 has simply been left out.

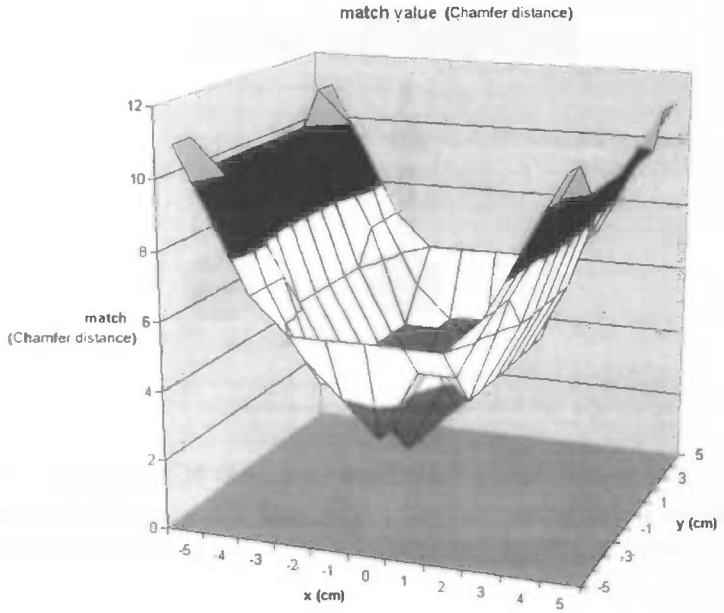


Figure 8: Simulation of match value ($D_{chamfer}$) on several x and y positions. X and y are in cm.

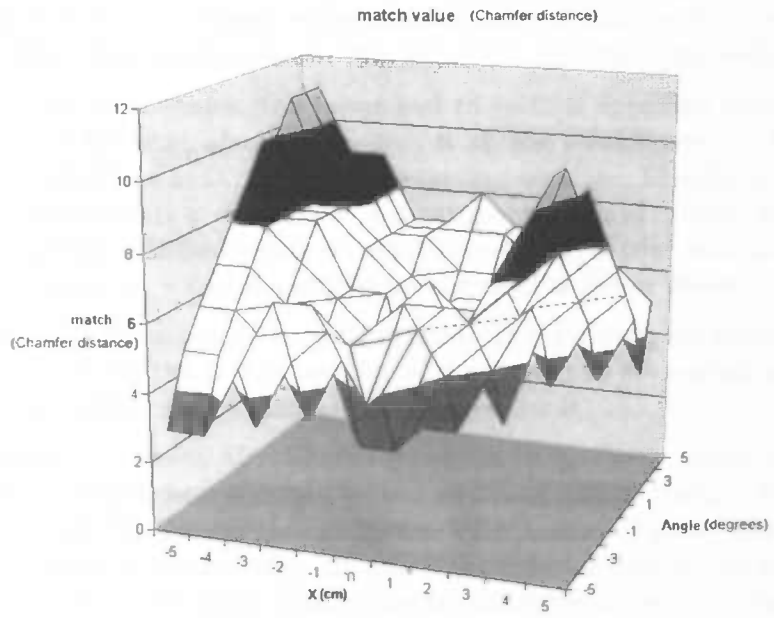


Figure 9: Simulation of match value ($D_{chamfer}$) on several x and *angle* position. X is in cm, *angle* in degrees.

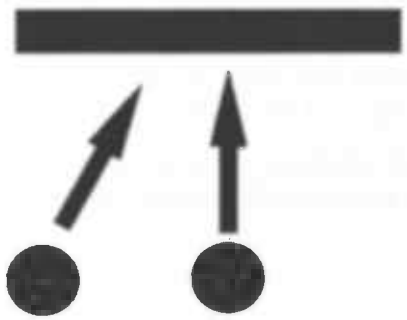


Figure 10: Example of two positions that could (incorrectly) return a similar match ($D_{chamfer}$) value. The robot is represented by the circles, the angle the camera is facing is represented by the arrows.

This doesn't influence total behaviour except that localization is not done). In the first experiments, the VRML-model was only used to position the robot in the width of the hallway (which corresponds to the y direction). This gave some promising results: the robot was able to position itself very accurately in this direction (the camera image was matched with an average error of about 2 cm). This is of course due to the rather clear landmarks in this direction, namely both walls.

In a latter experiment, the robot had to go to a specified location (the end of the hallway, about 11 metres from the starting point) for a few times, and afterwards return to its starting position. Ideally, it would go to the same location each time. Unfortunately, these results where not as satisfying as hoped (at these experiments, the model was used for all 3 dimensions: x, y and the *angle*), as the results below show.

In figure 11 these results are displayed. The distance between the target position and the actual position of the robot was measured or derived from the logfiles and is also listed in appendix B.

As figure 11 shows, the odometry seems to perform worse in the y-direction, which corresponds to the width of the hallway. The localization module (the correction by the VRML-model) gives more errors in the length of the hallway (x), when the robot drives to the end of the hallway. The mean error without model is 1,4 cm (x) and 4,6 cm (y) with standard deviations of 1,9 cm (x) and 5,8 cm (y). When the model is used, the mean error is 11,8 cm (x) and 3,1 cm (y), with standard deviations 11,4 cm (x) and 4,1 cm (y).

When the robot drives back to its starting point, the odometry performs

better than the VRML-model in both directions. The mean error without model is 4,5 cm (x) and 7,8 cm (y) with standard deviations of 4,4 cm (x) and 4,6 cm (y). When the model is used, the mean error is 8,9 cm (x) and 1,6 cm (y), with standard deviations 36,6 cm (x) and 14,6 cm (y).

Generally, the system is less capable to reach its target point with the VRML-model than without the model. In the next chapter, we will explain this behaviour.

5.5 Localization experiment II

To check if the match threshold θ is maybe still too high (and see if the system has to reject possible matches more often), the data from experiments described in 5.4 was analyzed with a lower threshold. This was done on the movies and log files from experiment I, so it was not possible to see the complete effect of this parameter change (because it is of course not possible to change the location of the robot afterwards).

This experiment shows that the errors in the results are not caused by a threshold that is too high. When the threshold is for example set at 2 (original value was 3), the incorrect results at experiment I will still occur (at all runs). In other words: all errors that were made with a threshold of 3, would also occur when the threshold is 2, and the robot would not perform better or worse.

Due to the repetitive elements in the hallway, some images return a match value below 2; even when their position differs more than 10 cm from the actual position of the robot.

5.6 Localization experiment III

A final small experiment that has been done, is another analysis of the log files and videos created by the robot during its tasks. Seventeen runs have been compared at specified distances from the starting point. At these points, we checked if the system was searching the correct area of the model (so the area that covers the actual position of the robot). Correct runs have been marked with a *1*, runs were the system was searching the wrong area (the failures) were marked a *0*. The results are listed in figure 12 and also in appendix C. The search algorithm searches 20 cm around the estimated position (in steps of 2 cm), so the estimated position is more than 10 cm different from the actual position if a *0* is listed in the table.

As the figure shows, the robot succeeds in 15 out of 17 runs, when we compare the results after 3 m. Nevertheless, this success rate drops

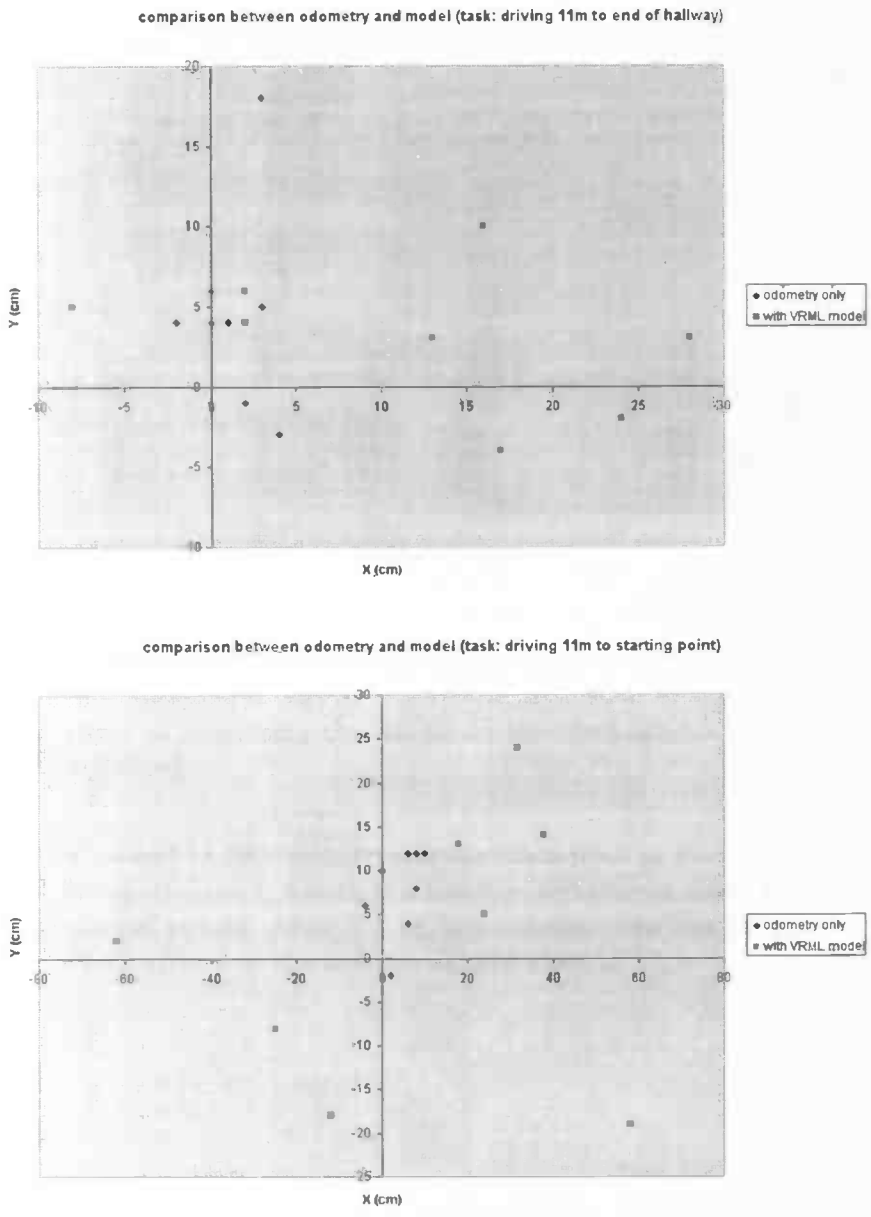


Figure 11: Results of localization experiment I. The target position is on *(0,0)*, the marks show the positions where the robot actually went. The upper graph shows the results of the robot driving to the end of the hallway. The lower graph shows the results of the robot returning to its starting position.

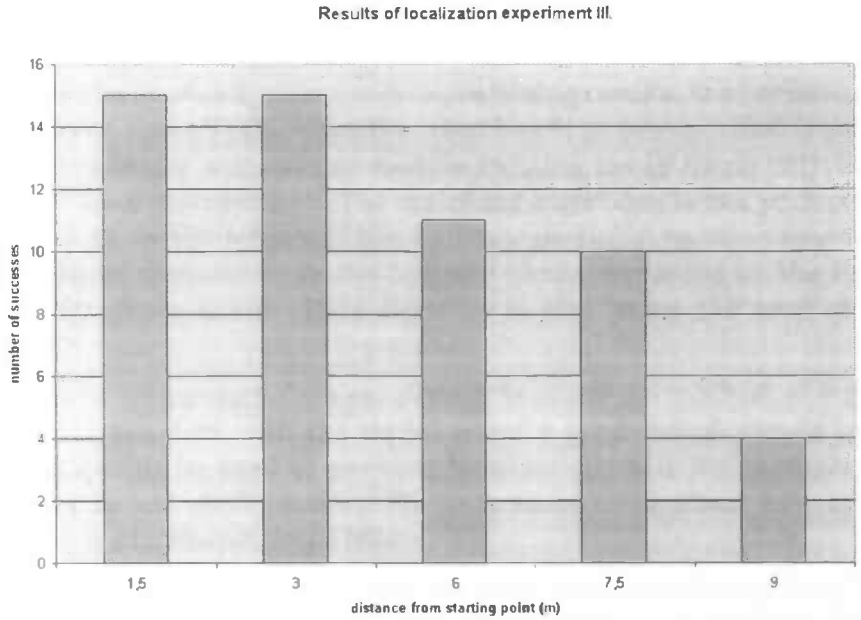


Figure 12: Results of the analysis of the videos after several distances from the starting point. The bars denote the number of successes (meaning the robot is searching the model in the area where it can find its current position).

when the robot has driven a greater distance (this is also shown by the results of experiment I, where the robot is sometimes more than 30 cm from its target point). After 7,5 m, the success rate has dropped to 10 out of 17 and after 9 m the success rate is 4 out of 17.

6 Discussion

Although the approach seems to give promising results, the performance is not robust enough yet. When the robot has to go to a specified location, it performs better without our module running (so by using only information from the odometry). The matching algorithm is not performing good enough in the length of the hallway, probably because some elements repeat themselves in the hallway (doors, elements on the floor). The performance in the other direction is also below the level of the odometry.

The results are nevertheless promising, because the method of matching the camera data with the model seems a good one. A simple hardware setup can be used to perform localization, it is for example not necessary to use stereo vision. We have some ideas about how to improve the performance listed below.

A very obvious improvement that we have overlooked is to do a less strict update of the position. In the current approach, when a match between model and camera image is found, the current position of the robot is overwritten with the position derived from the model. A better approach would be to integrate these values: Both the initial guess from the odometry as well as the suggested improvement from the model are available. The system then uses a weighed mean between these values as the new position. The use of a Kalman filter is a very common approach for this problem.

Experiment II shows, that tuning the threshold parameter θ does not give any improvement. The errors in the system are not caused by false matches that could have been rejected by using a lower threshold value. The choice of the minimal value seems to be a good approach, but when the starting point for the search is not accurate (due to the lack of a Kalman filter or similar module), the system is unable to find a good match.

Something that could be tried is to make a slightly more detailed model. In the current setup, some errors are also caused because the door at the end of the hallway is not modelled very accurately (it is a glass door, but it is modelled as an ordinary door with no glass in it). The analysis of the videos and log files done in experiment III seem to point to this conclusion: after a relatively small distance, the robot still performs reasonable, but when it has driven more (and thus reached the end of the hallway), more errors rise. This also points to the need to use a better method to update the current position (for example by using a Kalman filter), because the matching algorithm is able to detect good matches, but is sometimes searching the wrong area of the model due

to incorrect position updating.

An obvious disadvantage of this approach is the need to develop a VRML model first. We think, that these models will be developed more and more these days, because it is a very useful way to show a new building. Thus, there are already models available before buildings are even built (because the architect uses VRML to show what the building will look like). When such a model is not available, a skilled person could probably create one in less than a day (when a normal map like the one in figure 6 is present).

Something that has to be tested, is the performance of this approach in other buildings or rooms. If, for example, the walls of a room are not totally visible (because of furniture or plants), the robot will probably not find any matches. The system will continue (as described in 4.9), but will only use the information from the odometry. Other aspects, such as a building with less square forms, will probably not have any negative influence on the behaviour, because not the complete lines, but only the individual pixels of edges are detected and compared. Other methods, which are for example based on Hough transformation, rely more on straight lines.

This method is probably not very useful in outdoor conditions: the environment outside is very likely to be too difficult to model and changes too often. For such an environment, a system without a map is probably more suitable.

7 Conclusion and future work

This final chapter lists the conclusions from the theory and the experiments, and lists suggestions for further research.

7.1 Conclusion

The approach has showed some promising results. This use of VRML gives a rather useful way of localization with a very simple hardware setup. Without stereo-vision it is possible to check if the robot is on a certain position. If a model is available, the robot can be used without any training sessions which sometimes give very unexpected results. The localization by direct comparision between the camera image and the image from the model, seems useful.

The choice of using VRML also offers some extra advantages and possibilities, as described in the section on future work.

Unfortunately, the system doesn't work as robust as expected yet. The matching algorithm is not performing good enough in the length of the hallway, probably because some elements repeat themselves in the hallway (doors, elements on the floor). Besides this, the correction of the robot's position and angle by the system could be done in a more advanced way, for example by using a Kalman filter. Such a filter would give a better prediction of the location from which the model has to be searched. When such a filter is used, we expect the system to perform much better.

Another thing that could be tried is to make a slightly more detailed model. The current model is quite simple (which saves of course time when creating it), but more detail could add some extra reliability.

More experiments are also needed, to get a better understanding of the (errors in the) results. Also, tests in other environments have to be done.

An obvious disadvantage of a map-based approach is the need to develop a model first. We think that VRML models (or in the future maybe other, comparable models) will be developed more and more these days, because it is a very useful way to show a new building. Thus, there are already models available before buildings are even built (because the architect uses VRML to show what the building will look like).

Overall, the system seems to be a very usable start in experimenting with 3D-map based navigation. The chosen method definitely needs to be improved, but the global setup and the software can be used as a basis for new research. In the next section, some suggestions for future work are given.

7.2 Future work

Some other things that we think can be useful in the future are the following.

7.2.1 VRML and sensor fusion

It can be useful to use more information from the VRML model. In the *Visserver* toolbox, it is possible to find the distance to objects, just like the robot does with its sonars. If this information is used and compared to the real sonars on the robot, input from another sensor is available to check the location of the robot. In this approach, sensor-fusion could be used to gain more reliability.

7.2.2 VRML and staircases

Modelling the staircases in the VRML model will also give some advantages. In the described experiments, the robot was kept away from the staircases, to prevent it from falling down. With the current sensors (sonars, camera) it is hard to detect a 'missing' floor before the robot. When these missing floors are modelled in the VRML model, a script which checks if there is a floor on the location where the robot is going to could be used.

7.2.3 VRML and locations

Another improvement could be to read the locations of targets (for instance of a room to go to) directly from the VRML file. In the current scripts, the translation between a coordinate and for instance a room number is made by hand. It would probably be possible to make this translation automatically when for instance someone clicks on a location in the model. This would also give the possibility to develop a very intuitive way of controlling the robot, like in [13].

7.2.4 Navigation algorithm

As already mentioned, the algorithm used to drive to location is rather simple. If this approach is going to be tested in a more complex environment, an algorithm which can also handle corners and doorways is definitely needed. Fortunately, such a change can probably be implemented rather easily.

7.2.5 Ambient intelligence

The possibility to move the virtual camera in a VRML model to every location also opens some doors to *ambient intelligence*. If there is for instance a camera in the hallway (for guarding purposes), the system could compare the image from this camera with the model as well (because the virtual camera in the VRML model can easily switch from the robot to the position of the guarding camera).

A Results of odometry tests

Below are the results of the bidirectional square-path experiment. All values are in *cm*.

Task: driving a 1m by 1m square in two directions. The distance from the starting point is listed in the table.

Clockwise	difference X	difference Y
#1	2	3
#2	2	2
#3	1	3
#4	2	2
#5	1	2
μ	1.6	2.4
r	2.88	
Counterclockwise	difference X	difference Y
#1	1	2
#2	4	2
#3	0	3
#4	1	3
#5	3	3
μ	1.8	2.4
r	3	

B Results of localization experiment I

Below are the results of the comparison between the robot driving with only odometry and the robot driving with both odometry and the VRML model. All values are in *cm*.

Task: driving to end of hallway (11m from starting point). The distance from the destination point is listed in the table:

#	difference X	difference Y	#	difference X	difference Y
without model			*with model*		
1	0	4	*1*	28	3
2	3	18	*2*	2	4
3	2	-1	*3*	2	6
4	0	6	*4*	17	-4
5	4	-3	*5*	16	10
6	3	5	*6*	-8	5
7	-2	4	*7*	13	3
8	1	4	*8*	24	-2
μ	1,4	4,6	μ	11,8	3,1
σ	1,9	5,8	σ	11,4	4,1

Task: returning to starting point (11m again):

#	difference X	difference Y	#	difference X	difference Y
without model			*with model*		
1	10	12	*1*	58	-19
2	8	12	*2*	-12	-18
3	6	12	*3*	-62	2
4	0	10	*4*	38	14
5	-4	6	*5*	24	5
6	6	4	*6*	18	13
7	2	-2	*7*	-25	-8
8	8	8	*8*	32	24
μ	4,5	7,8	μ	8,9	1,6
σ	4,4	4,6	σ	36,6	14,6

C Results of localization experiment III

Below are the results of the analysis of the logfiles and videos after the robot has driven certain distances from the starting point. A *1* denotes that the robot is still searching the model in the correct area (so the area that covers the actual robot position), a *0* denotes a failure: the robot has somehow lost track of the correct position and is searching an area where it will not find its actual position.

Task: driving to various target points. All tasks are compared after several distances.

run #	1,5m	3m	6m	7,5m	9m
1	1	1	1	1	0
2	1	1	0	0	0
3	1	1	1	1	0
4	1	1	0	0	0
5	1	1	0	0	0
6	1	1	1	0	0
7	1	1	0	0	0
8	1	1	1	1	0
9	1	1	1	1	0
10	1	1	1	1	0
11	1	1	1	1	1
12	0	0	0	0	0
13	1	1	1	1	1
14	1	1	1	1	1
15	1	1	1	1	1
16	0	0	0	0	0
17	1	1	1	1	0

D List of implemented robot control commands

Below is a list of commands that are implemented in *Matlab*

Robot programmes

rbavoid - start the obstacle avoid programme

rbkeyboard - move the robot by keyboard control

rbmouse - move the robot by mouse (from another pc)

rbfollow - let the robot follow moving objects (using the sonars)

Utilities

rbdisplaysonars - show the sonar values

rbshowmap - show the map created by RBMAP

rbshownewmap - create the VRML map created by RBNEWMAP

rbreviewpath - review the results of the matching process (see figure 7 for a screenshot of this utility)

rbmaakavi - make a AVI-file (video) of the matching process

rbnetview - view the robot's camera image (from another pc)

rbsay - make the robot 'talk'

rbremote - start the server to control the robot from another pc

Scripts for virtual and real camera

rbinitcam - initialise the VRML-world

rbdxstart - start the *DirectShow* graph for the camera

rbmovecam - move the virtual camera to a specified position

rbmovetopcam - move the virtual camera to a specified position (top view)

rbcompare - compare the camera image with the VRML model (partly replaced by rbreviewpath)

rbfindmatch - try to match the camera image against the VRML model (by moving the virtual camera)

Robot low-level functions

rbmove - move the robot

rbgetbumper - get the bumper status

rbgetsonars - get the sonar values

rbgetpos - get the robot position from the odometry

rbgetangle - get the angle from the odometry

rbreset - reset the robot's odometry to zero or to another position

rbstartsonars - start the sonars

rbstopsonars - stop the sonars

rbgetobstacle - check if there is an obstacle near the robot

rbmap - create a pixel map of the robot path

rbnewmap - improved version of RBMAP which creates a VRML map

References

[1] S.J. Russel and P. Norvig. *Articial Intelligence - a modern approach*. Prentice Hall, Inc., 1995.

[2] J. Borenstein, H.R.Everett, and L.Feng. Where am I? - sensors and methods for mobile robot positioning. Technical report, The University of Michigan, 1996.

[3] L.G. Shapiro and G.C. Stockman. *Computer Vision*. Prentice-Hall Inc., 2001.

[4] G.N. DeSouza and A.C. Kak. Vision for mobile robot navigation: A survey. *IEEE Transactions on pattern analysis and machine intelligence*, 24:237–267, February 2002.

[5] R. Brooks. New approaches to robotics. *Science*, (253):1227–1232, 1991.

[6] M. Fichtner. A camera sensor model for sensor fusion. Master's thesis, Technische Universität Dresden,, October 2002.

[7] A. Kosaka and A.C. Kak. Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. *Computer Vision, Graphics and Image Processing - Image Understanding*, 56:271–329, 1992.

[8] B.J.A. Kröse, N. Vlassis, and R. Bunschoten. Omnidirectional vision for appearance-based robot localization. In G.D. Hagar, H.I. Cristensen, H. Bunke, and R. Klein, editors, *Sensor Based Intelligent Robots: International Workshop*, pages 39–50. Springer, October 2000.

[9] G. Bianco, A. Zelinsky, and M. Lehrer. Visual landmark learning. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2000)*. Takamatsu, Japan, October 2000.

[10] Greg Welch and Gary Bishop. An introduction to the Kalman filter. March 2004.

[11] Akihisa Ohya, Akio Kosaka, and Avinash Kak. Vision-based navigation by a mobile robot with obstacle avoidance using single-camera vision and ultrasonic sensing. 14:969–978, December 1998.

[12] M. Fichtner and A. Großmann. A visual-sensor model for mobile robot localisation. In *Proc. of the 18th Int. Joint Conf. on Artificial Intelligence*, August 2003.

[13] M. Schmitt. Verona - virtual environment for robot navigation. pages 85–90, 1999.

[14] the WEB 3D consortium. *VRML: Virtual Reality Modeling Language*. http://www.web3d.org/.

[15] *Nomad Scout user's manual*. Nomadic Technologies Inc., 1999.

[16] *DirectShow website*. http://www.microsoft.com/Developer/PRODINFO/directx/dxm/help/ds/default.htm.

[17] *Matlab website*. http://www.mathworks.com/products/matlab.

[18] *Open Inventor*. http://www.sgi.com/software/inventor.

[19] *Information about MEX-files from Mathworks*. http://www.mathworks.com/support/tech-notes/1600/1605.shtml.

[20] I.R. Nourbakhsh. *Free Robot Programming Code*. http://www-2.cs.cmu.edu/ illah/ROBOCODE/.

[21] D.M. Gavrila. Multi-feature hierarchical template matching using distance transforms. In *Proc. IEEE International Conferene on Pattern Recognition*, 1998.