Diploma Thesis

Autonomous Navigation and Control of a Mobile Robot in a Cell Culture Laboratory

by

Axel Schneider
(Mat. No. 1414952)

Daniel Westhoff
(Mat. No. 1234134)

June 2002

Supervisors:

PD Dr. Jianwei Zhang

Dipl.-Inform. Torsten Scherer

University of Bielefeld

University of Bielefeld
Faculty of Technology
Technical Computer Science
33594 Bielefeld
Germany

We hereby declare that we have done this work completely on our own and with no other help and resources than mentioned in the acknowledgements and bibliography.

Axel Schneider Daniel Westhoff

Bielefeld, June 2002

Contents

Chapter 1

Introduction

That old sorcerer has vanished
And for once has gone away!
Spirits called by him, now banished,
My commands shall soon obey.
Every step and saying
That he used, I know,
And with spirits obeying
My arts I will show.

First verse of *The Sorcerer's Apprentice*
by
Johann Wolfgang von Goethe (1779)[1].

The wish for good spirits is as old as the history of mankind. Already Goethe's sorcerer's apprentice wanted to make use of his master's spirits. In folktales of many cultures there are stories of little helpers and servile spirits. In German folklore it is the ”Heinzelmann” who together with his comrades is said to finish other people's work during the night when everybody else sleeps.

But not only in dreams these things were conceived. From the abacus of ancient times to the first calculating machines of Wilhelm Schickard (1623) and modern computers, people have always developed technical machines which make life easier. In the 1950's, numerical controls (NC) were introduced in manufacturing.[2] In 1954, Georg Devol filed a patent application for a programmable manipulator. He realized this invention together with Joseph Engelberger in 1956.

[1]Translation by Edwin Zeydel (1955), see appendix B.1
[2](SPITERI, 1990), p. 3.

Two years later in 1958, the robot manufacturer Unimation was incorporated with Engelberger as its founder.[3] This is considered to be the beginning of modern robot applications in industry.

The word robot itself originated in the 1921 science fiction play R.U.R. (*Rossum's Universal Robots*) by the Czech author Karel Čapek. The term is derived from the Czech word robota, meaning ”forced labour”.[4]According to this play, a robot is

> ”[...] any automatically operated machine that replaces human effort, though it may not resemble human beings in appearance or perform functions in a humanlike manner.[...]” [5]

In the field of robotics, a vast amount of attention has been payed to stationary robotic manipulators on the one side and to mobile robots on the other side. For a long time, these two branches developed in parallel but not together. Robotic manipulators can be found in our industrial society in a lot of places. One can come across them for example in the car industry for welding, spray painting and the assembly of modules. They replace humans where heavy, tedious or dangerous work must be performed. But still most of the applications that exist nowadays work with fixed robot arms.

Mobile robotics however is even one step behind in industrial applications. In a lot of cases, mobile robots are used as wheeled transport vehicles in warehouses. These constructions follow paths marked by reflective tape, paint or buried wire. In modern approaches, more autonomous systems are designed which for example permit to build up mobile delivery systems for hospitals. Autonomous systems in these cases may react dynamically to changes in their environment or may be easily able to switch between different tasks. The developers of those systems created different methods like behaviour-based and bio-inspired approaches or more algorithmic solution based on physical models to solve the emerging problems. Nowadays, the algorithmic methods are preferred for industrial solutions because in every case the next step the robot takes is predictable. People are not yet that familiar with robots in their environment to accept an unpredictable move from the robot.

However, if a behaviour architecture is used it is difficult to perform a precise positioning of the vehicle. In a lot of cases, a reasonable trade-off between flexible behaviour and precision is not possible. Let's assume a robot has to perform the task of moving from point 1 to point 2. This is a simple example for a path planning problem. One standard solution is to calculate the exact trajectory and make the robot follow it. If the robot comes across an obstacle on its journey it has to find a path around it, which means it has to leave the original plan at least for a while. It might be reasonable to use an avoid-obstacle behaviour for that. But who can decide when to switch off a precise algorithm and when to use a behaviour or how to mix them? These questions are

[3](SPITERI, 1990), pp. 5-6.

[4]Ibid., p. 1.

[5]Encyclopædia Britannica

focal points of current research interests. Even more problems occur when a mobile robot carries a manipulator arm.

1.1 State of the art

In industry, a lot of different transport systems are established. Besides conveyors and fork lifts, other automated transporters called AGVs (Automated Guided Vehicles) carry parts to different stations in production halls. **Figure 1.1** shows such a system by the manufacturer CORECON. AGVs act automatically but not autonomous. Usually, they follow marked paths to find their goal points with high accuracy. But this precision has to be paid with the lack of autonomy.

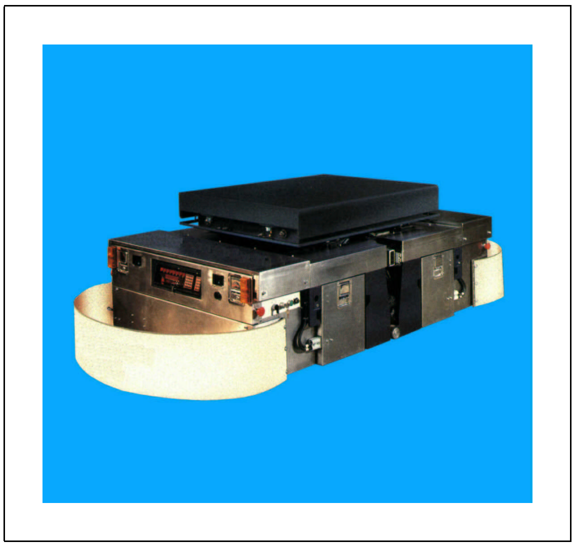


Figure 1.1: *An automated guided vehicle*[6]

If one looks for autonomous systems which can perform transportation tasks, very often, behaviour-based systems can be found.[7] These systems allow a certain level of autonomy but do not obtain a good positioning reliability. **Figure 1.2** shows a service robot by ActivMedia Robotics as an example. Such a service task does no require millimetre precision accuracy.

In fact, these systems need a reactive behaviour in the manner ”*cup not above table yet* $\longrightarrow$ *move a bit forward*”. In this case, results in the shape of numbers do not play a role for this kind of positioning strategy.

[6]This photograph is courtesy of CORECON Automated Guided Vehicles.

[7]See (Arkin, 1998) for detailed information about the behaviour-based approach.

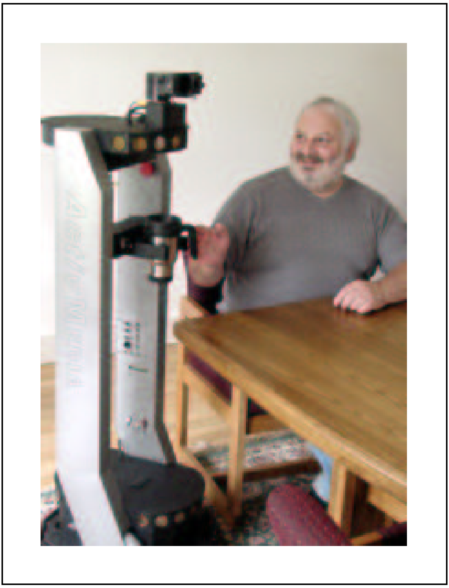


Figure 1.2: *A service robot by the mobile robot manufacturer ActivMedia Robotics delivers a cup.*[8]

As a last example in this section, **figure 1.3** shows another robot by ActivMedia Robotics. This one is a model with manipulator arm and a payload of about 100 kg. Again, this system is controlled with a behaviour-based approach. Especially, mobile robots, which are equipped with a manipulator arm, are very rarely found as a commercial product. The robot by ActivMedia Robotics as mentioned above and the mobile robot *GenBase II* which is used in this work (see chapter 2 for a detailed description) are only produced on order. They are mainly used in research projects at universities. Still, a lot of questions have to be answered until they are suitable to fulfill industrial or service tasks.

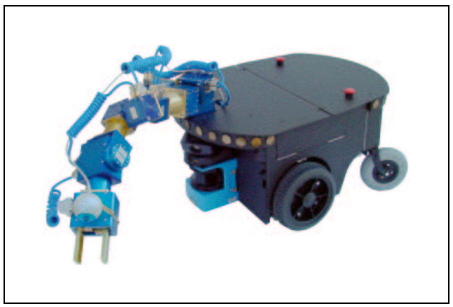


Figure 1.3: *A mobile robot with arm and a high payload of about 100 kg by the mobile robot manufacturer ActivMedia Robotics*[8]

[8]These photographs are courtesy of ActivMedia Robotics.

It should be mentioned that behaviour-based robotics is still at the beginning of its development. Most of the systems are built for scientific purposes only. Last but not least, the above mentioned disadvantage of low position reliability prevent these systems from being used in industry.

1.2 About this work

The diploma thesis deals with the navigation of a mobile robot in a laboratory environment. The mobile platform *GenBase II* has to perform a transportation task in the cell culture lab of the Institute of Cell Culture Technology at the University of Bielefeld. For these purposes, a robust operating control software with high positioning accuracy must be developed.

This task can be separated into seven main parts. Chapter 2 gives an overview on the robot's software and hardware. An introduction to Kalman filtering and extended Kalman filters is given in chapter 3. This is needed to introduce a means for solving the self-localisation problem of the mobile platform. Chapter 4 introduces a model for the robot which serves as part of the Kalman filter's world model. Together with the measurement model and the Kalman filter theory, the self-localisation for the mobile robot is realized. Based on the preceding chapters, chapter 5 deals with the development of a suitable control system which gives the robot a method for an accurate positioning. Chapter 6 describes methods for path planning in consideration of map, robot and path representation. The path planning is based on the local position of the robot and the desired goal point. The behaviour of the developed software is compared to those of the original software in chapter 7. For this purpose, two experiments are carried out. The results are illustrated and discussed. Based on the drawn conclusions, chapter 8 gives an outlook on the work which still has to be done.

Figure 1.4 on the next page shows a photograph of the mobile platform *GenBase II* by the manufacturer *genRob*. A Mitsubishi PA-10 manipulator arm is mounted on top of the platform. This is the hardware basis which is explained in the next chapter.

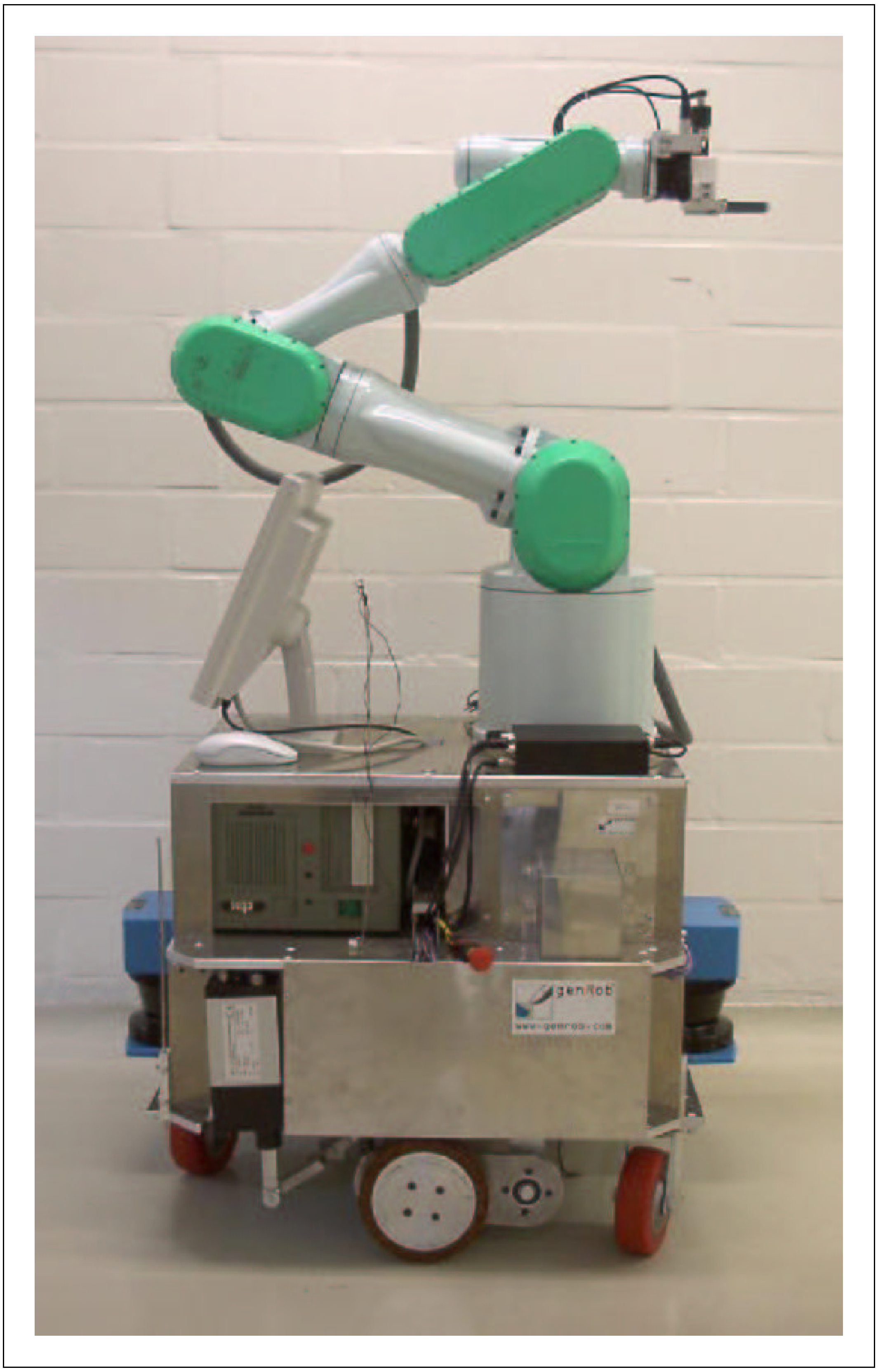


Figure 1.4: *The figure shows a photograph of the mobile platform GenBase II with a PA-10 manipulator arm mounted on top.*

Chapter 2

Setup and Hardware

The present chapter gives an introduction to the scenario in which the mobile robot's task is embedded. The hardware and software setup is described in the corresponding sections. The mobile robot is a *GenBase II* which is manufactured by the company *genRob*.[1] The mobile platform *GenBase II*, equipped with the PA-10 manipulator arm, is to be used in a cell culture laboratory, where it has to perform a complete sample management cycle. This means that the robot grasps test tubes and transports them between different biotechnological devices. In addition, the robot has to operate these devices with its manipulator arm.

A possible sequence might be the following: Firstly, the robot fetches an empty test tube from the tube storage and brings it to the sample device. A sample is taken from the fermenter's content and filled into the test tube. The robot takes the full test tube and transports it to the centrifuge where it is put into the rotor. The lid is closed and the centrifuge is started. After the centrifugation has finished, the supernatant has to be pipetted and frozen in a freezer. Again, the robot provides the transportation to the pipette and to the freezer. The rest of the sample is brought to the cell counter (CEDEX) which determines the number of cells. During the whole process, the robot has to operate all devices with its manipulator. That means that all buttons are pressed and all lids are opened and closed by the robot autonomously.

During that operating sequence, the robot gets the basic commands from another computer. A coarse overview of the software environment in the cell culture lab is shown in **figure 2.1**. The robot is connected to the lab network via WLAN (IEEE 802.11b). The LAN is symbolized with the red lines in **figure 2.1**. A database contains all necessary parameters and the positions of all devices in the laboratory. Furthermore it contains robot scripts which include different production sequences. The database runs on the same computer as the main program. The main program controls the general procedures of the transport processes. Moreover, it operates the sample device. Other devices which are connected via TCP/IP are the cell counter (CEDEX) and

[1]For more information about the company see the web page *http://www.genrob.de*.

the Multi Fermenter Control System (MFCS). The robot itself contains software for the control of the manipulator arm and for the control of the mobile platform. An image server software runs on the mobile robot as an interface to the gripper's micro head camera. The scope of the different computers is indicated by the dashed lines in **figure 2.1**.

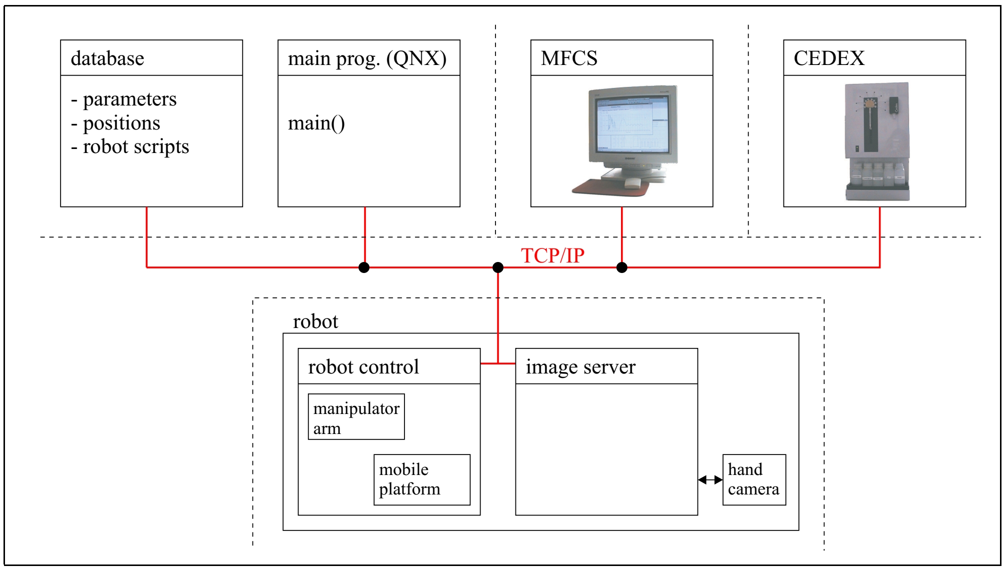


Figure 2.1: *Overview on the software environment of the mobile platform*

2.1 The mobile platform *GenBase II*

The hardware design of the mobile platform *GenBase II* changed during the development process because of further development of the navigation software. **Figure 2.2** shows the hardware configuration of the robot in delivery condition. The notable feature of this configuration is the fact that it is equipped with two control computers. This has the following reasons. Since the software for the PA-10 manipulator arm needs a Linux operating system to run and the software for the mobile platform was programmed under Windows 2000, they can only be used simultaneously when placed on two different computers. In **figure 2.2**, the Linux control PC can be seen on the left-hand side of the diagram. It is connected with the PA-10 arm controller via an ARCNET interface. Furthermore, the Linux PC guarantees the connection with the lab network by means of a WLAN PCMCIA interface card (IEEE802.11b). Therefore, all work instructions from the lab computer run through the Linux PC. The control PC on the right-hand side of the picture is connected with the Linux PC via TCP/IP interface. The operating system of the PC is Windows 2000 and hosts only genRob software for the control of the mobile unit. According to its task, the Windows 2000 PC is linked to the measurement and drive hardware of the mobile platform. This is done via a RS232 interface. A C167 microcontroller controls the

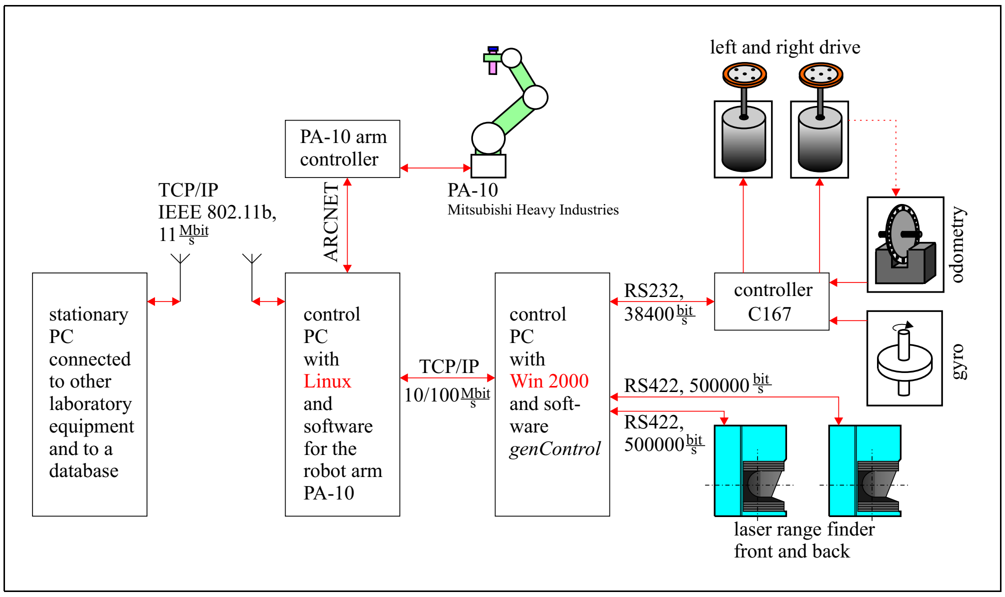

left and right drive
PA-10 arm controller
TCP/IP
IEEE 802.11b,
11 Mbit/s
ARCNET
PA-10
Mitsubishi Heavy Industries
odometry
stationary PC connected to other laboratory equipment and to a database
control PC with Linux and software for the robot arm PA-10
TCP/IP
10/100 Mbit/s
control PC with Win 2000 and software genControl
RS232,
38400 bit/s
controller C167
gyro
RS422, 500000 bit/s
RS422,
500000 bit/s
laser range finder front and back

Figure 2.2: *The hardware setup in delivery condition*

left and right drive unit and receives the odometry's data. In addition, the controller reads out the gyrocompass. These data give information about the rotatory part of the robot's movements. The front and back laser range finder of the mobile platform are connected to the Windows 2000 control PC too and can be addressed via an RS422 interface. The goal of the diploma thesis is the programming of a new precision navigation software. Besides the improvement in accuracy, this development saves the Windows 2000 PC which then becomes redundant since the software was developed in Linux. The current hardware setup is shown in **figure 2.3**.

The changes to the mobile robot's software structure during this work is documented in **figures 2.4** and **2.5**. The first shows the delivery condition supplemented by the first part of the proprietary development, the path planner (marked in red). The software *genControl*, displayed on the right-hand half of the diagram, runs on an independent Windows 2000 PC (see also **figure 2.2**). This software solely takes move commands via TCP/IP interface and provides a self-localisation on the basis of the robot's odometry and its laser range finders. The control software runs on the Linux computer. In the diagram, the control software is represented by the class CROBOT. It can be divided into the software for the control of the manipulator arm (represented by the class CARM) and into the software for the control of the mobile platform (represented by the class CMOBILE). CMOBILE mainly contains two classes. The first one is the class CGENBASE which encapsulates the connection via TCP/IP to the *genControl* software. The second one is the path planner which calculates a path from the robot's current position to any goal position. The basis for that is a map of the robot's environment. A description of the path planner can be found in chapter 6. For each subgoal delivered by the path planner, CGENBASE sends single move commands via its TCP/IP connection to the *genControl* software.

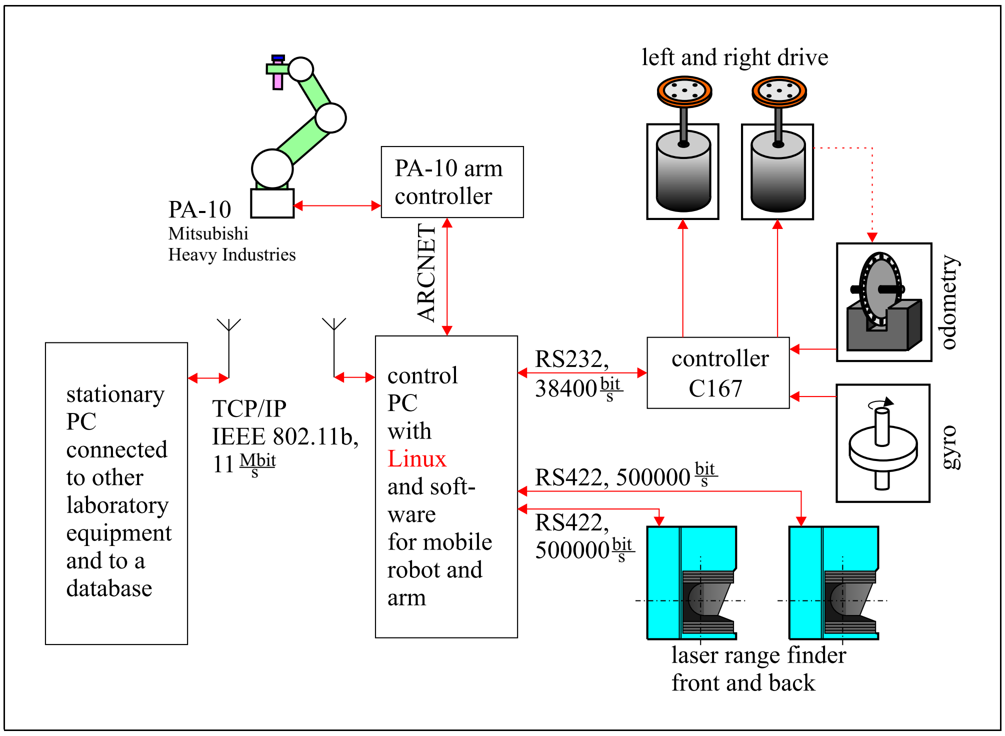


Figure 2.3: *The current hardware setup*

Figure 2.5 shows the current software setup which is associated with the hardware setup in **figure 2.3**. On the left-hand side of the class CROBOT, the software for the control of the manipulator arm (CARM) has remained the same as in **figure 2.2**. The class CMOBILE, for the control of the mobile platform, still contains the path planner. To save the Windows 2000 PC, the class CGENBASE has been changed as far as that it contains the direct communication with the peripheral devices of the mobile platform. CMOTORFEEDER provides the connection with the C167 drive controller. The two classes called CLASERFEEDER operate both laser range finders. Furthermore, CGENBASE includes the localisation which is based on an extended Kalman filer (see chapters 3 and 4) and a trajectory generator which supervises that the desired goal point is reached.

The coarse procedure of a complete movement of the mobile robot to a new goal point looks as follows: CGENBASE contains a move command which gets the new position and orientation of the robot to be taken in. The desired translation and orientation is pushed to a stack. The trajectory generator fetches the new position and orientation estimate from the localisation and calculates a trajectory to the goal point based on that estimate and the current move command from the stack. Based on that new trajectory, new drive commands are transmitted to the left and right drive by CMOTORFEEDER. On the basis of these drive commands, the odometry and gyro data and the data from the laser range finders, the position and orientation estimates are renewed by the localisation. Again based on this, the trajectoy generator updates the trajectory to the goal point and so on. This iterative process is finished when the desired position and orientation has been reached within certain tolerance margins.

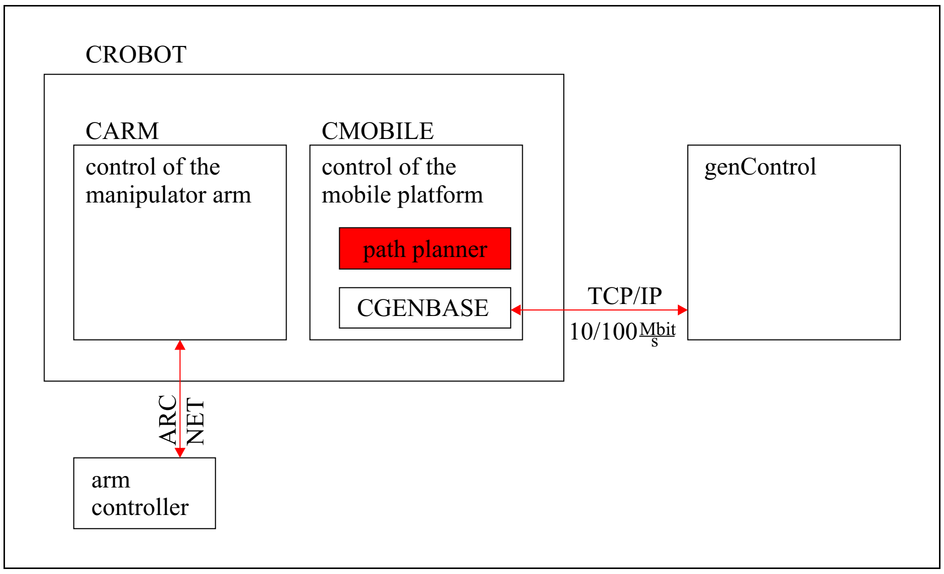

CROBOT
CARM
control of the manipulator arm
CMOBILE
control of the mobile platform
path planner
CGENBASE
genControl
TCP/IP
10/100$\frac{\text{Mbit}}{\text{s}}$
ARC NET
arm controller

Figure 2.4: *General overview of the software of the mobile platform in delivery condition*

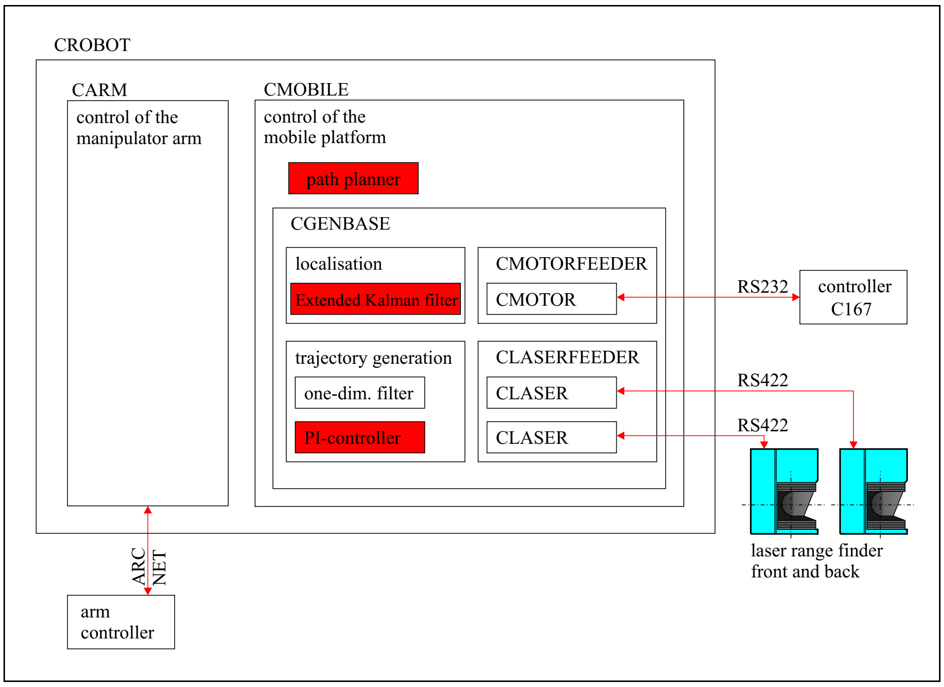


Figure 2.5: *Overview of the current software structure of the mobile platform*

Figure 2.6 includes technical drawings of the top and back views of the mobile platform. Details on the castor and drive wheels are also given. The specified dimensions are used, among other things, to set up a system model of the robot discussed in chapter 4.

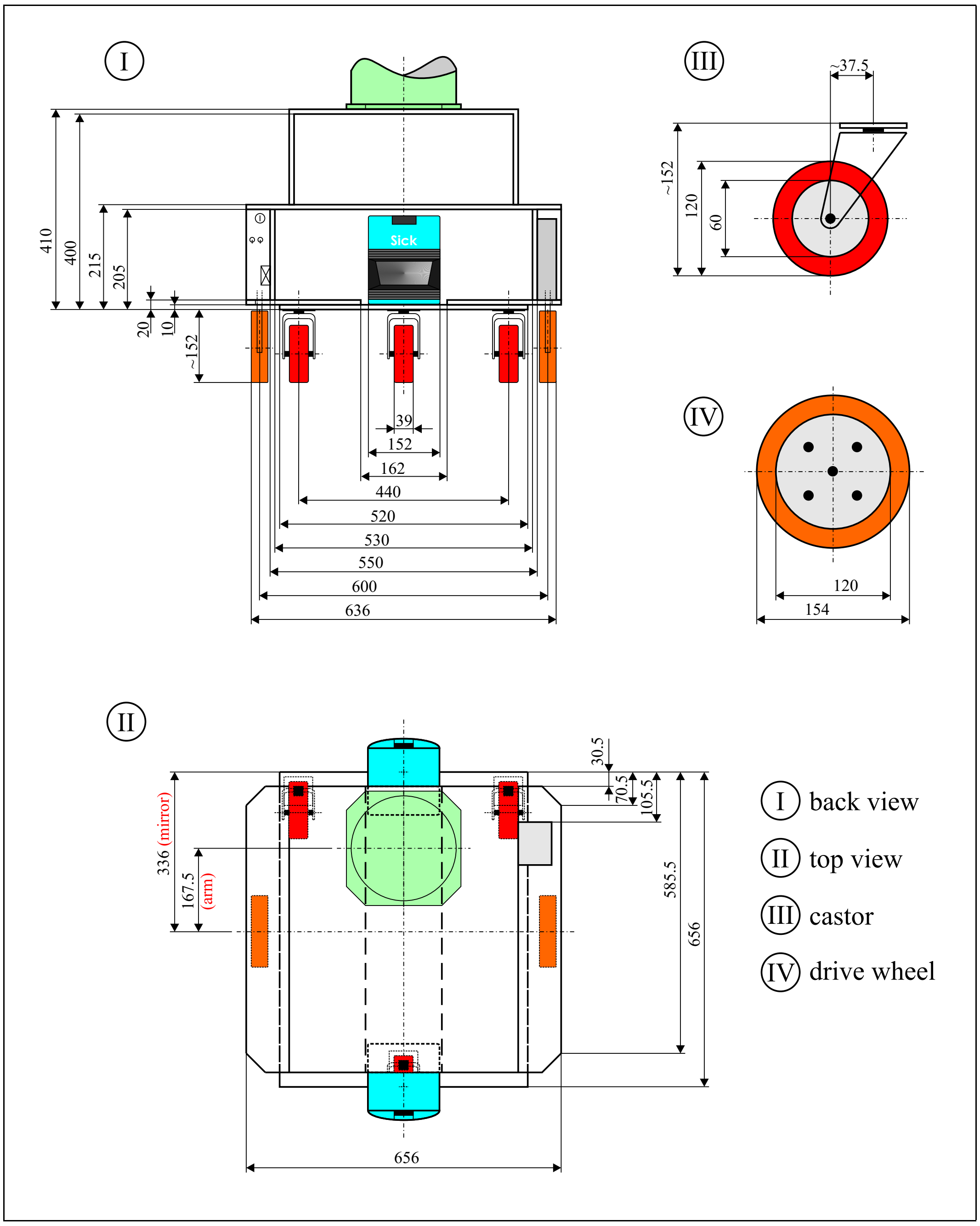


Figure 2.6: *Technical drawings of the back and top view and details of the castor and drive wheels*

Figure 2.6 shows a technical drawing of the mobile platform in front, left side and right side view. The positions of the laser range finders, the gyro, the manipulator arm and the drive lifters are given.

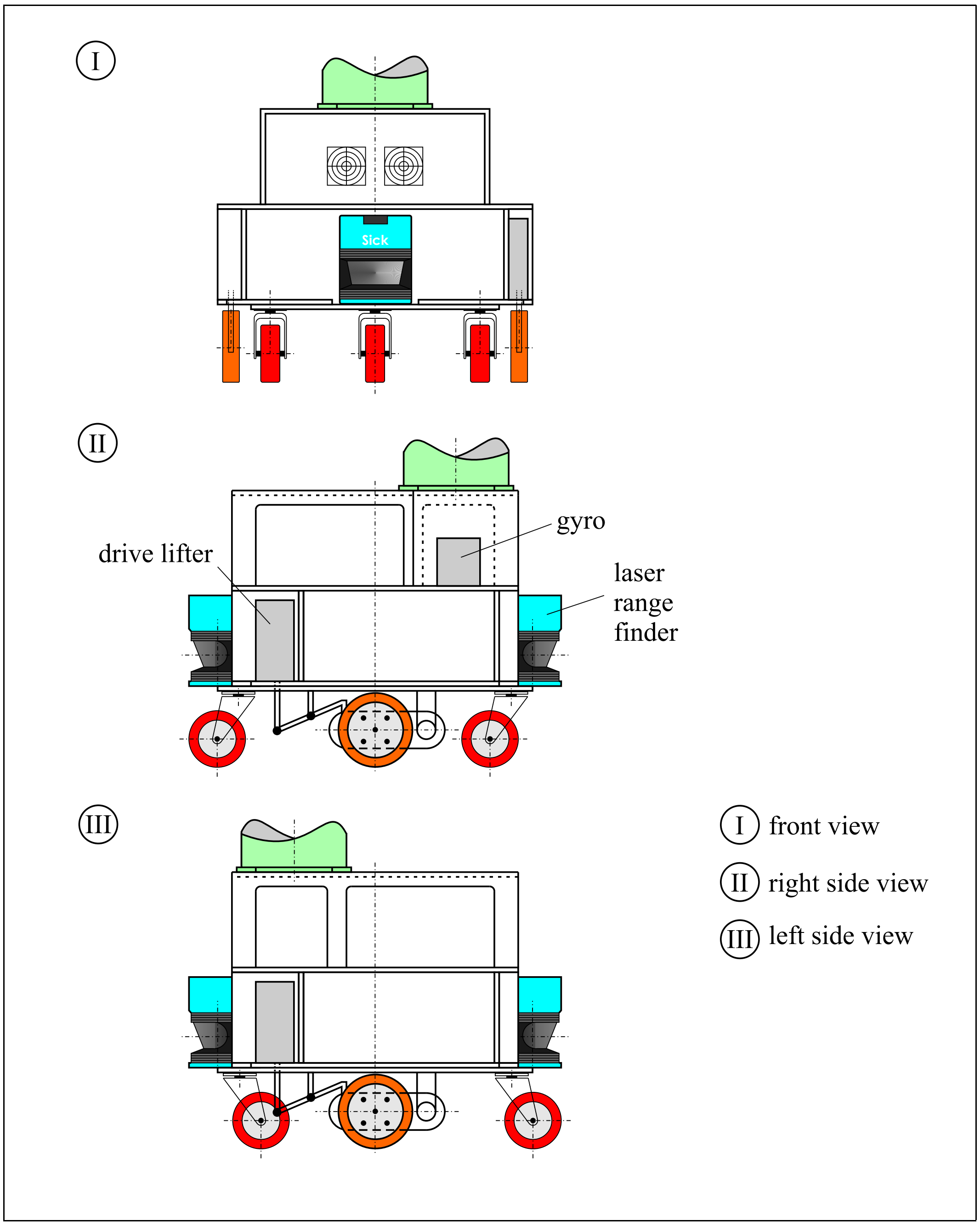


Figure 2.7: *Technical drawings of the front, right and left side view*

2.2 The laser range finder on the mobile platform *GenBase II*

The mobile robot platform *GenBase II* uses two laser range finders for contact-free scans of the robot's environment. SICK laser range finders use infrared laser beams to measure the distance to surrounding objects.

The principle of function is shown in **figure 2.8**. The sensor operates on the principle of reflex light time measurement or time of flight. A laser light source S emits short light pulses. These pulses are reflected by objects and thrown back to the scanner where the pulses are detected by a receiver E. The time Δt between emitting a light pulse and receiving it is proportional to the distance s between the object and the laser range finder. In the laser range finder there is a rotating mirror which redirects the light pulses so that a semicircular area in front of the scanner is covered. The mirror angle is detected. Thus, the directions and the distances of objects in front of the laser range finder are determined.

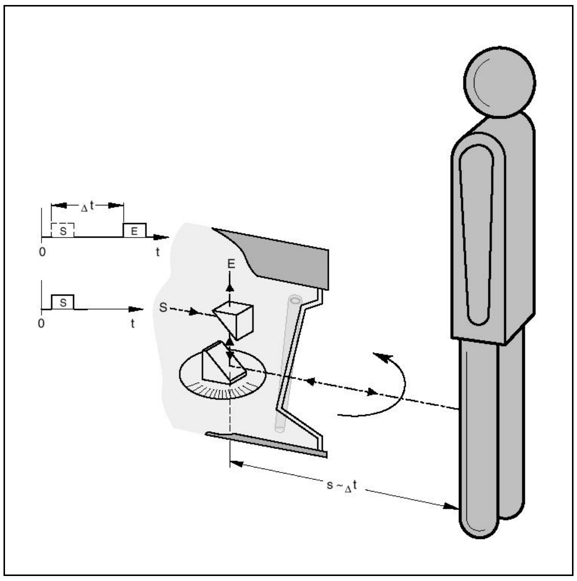


Figure 2.8: *The SICK laser range finder's principle of function*

The light spot, which appears where the emitted light pulses encounter the surface of an obstacle, has a certain diameter. This diameter depends on the range. The further the object is away from the scanner the bigger the diameter of the light spot is. The following relation can be found in the SICK manual

$$d_{spot} \approx 13 + 0.0046 \cdot r \quad \text{[mm]}$$

The light spot's diameter for a range of 0 m therefore is approximately 13 mm. For a range of 10 m the diameter is approximately 59 mm.

Depending on the mode of operation, a light spot will be emitted every $0.25°$, $0.5°$ or resp. $1°$. In the mode that is used on the mobile platform the scanning angle is $0.5°$. This means that 361 measurements are executed for a scanning range of $180°$. While the laser range finder's mirror rotates $180°$ all 361 values have to be taken. This can be done in approximately $26\ ms$. The mirror's direction of rotation can be seen in **figure 2.9**.

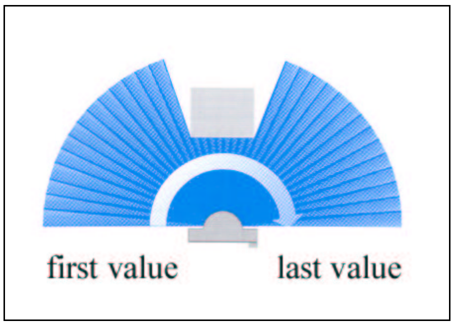


Figure 2.9: *The SICK laser range finder's direction of rotation*

Another property is the precision of measurements. The manufacturer specifies this as follows: At a scanning angle of $180°$ and a resolution of $10\ mm$ the typical precision lies in the range of $\pm 30\ mm$. This value is very important for the use of the laser range finder's measurements in a Kalman filter. The connection becomes clear in chapter 4. The physical dimensions of the SICK laser range finder LMS200 are shown in **figure 2.10**.

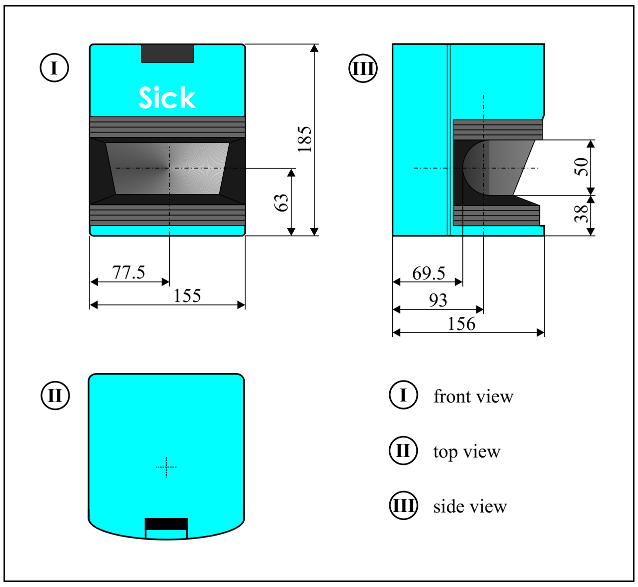
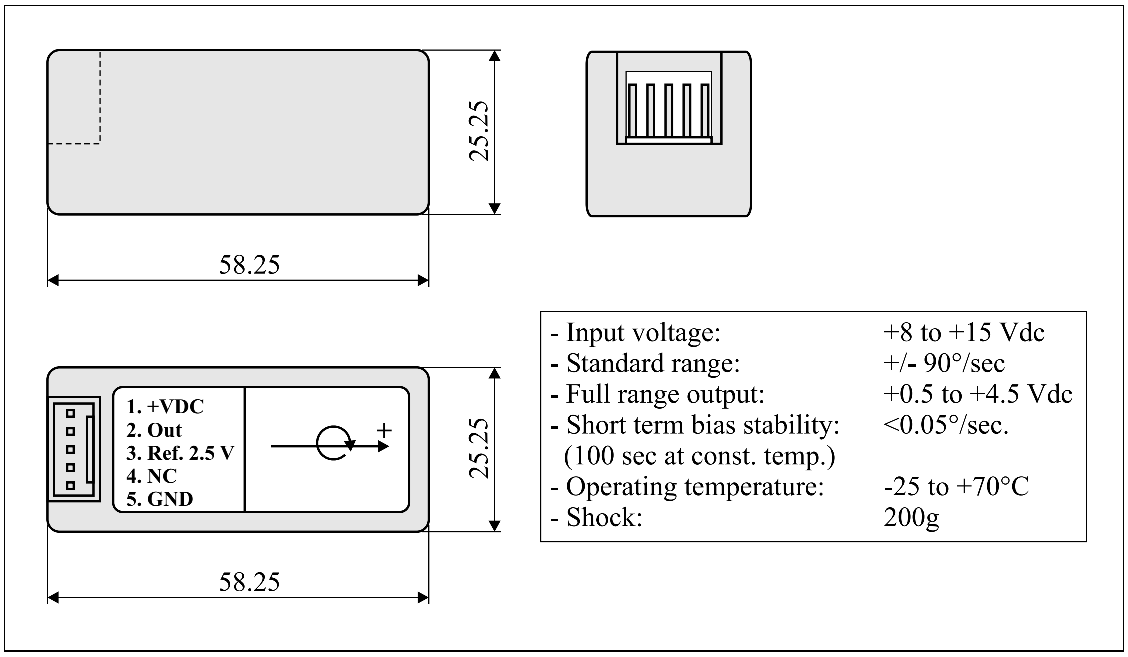


Figure 2.10: *Technical drawing of the Sick laser range finder LMS200*

From **figures 2.6** and **2.10**, the installation height of the laser range finder can be determined. This height is $225\ mm$. These measuring instruments therefore provide only a two-dimensional picture of the world in reference to this installation position.

2.3 The gyrocompass on the mobile platform *GenBase II*

The featured mobile system uses a gyrocompass which utilizes a one piece, micromachined, vibrating quartz tuning fork sensing element. A rotation about the gyrocompass' rotational axis leads to the appearance of a Coriolis force. This force causes a displacement of electric charges resulting in a voltage which is proportional to the rate of rotation. The technical data and drawings are provided in **figure 2.11**.



- Input voltage:	+8 to +15 Vdc
- Standard range:	+/- 90°/sec
- Full range output:	+0.5 to +4.5 Vdc
- Short term bias stability: (100 sec at const. temp.)	<0.05°/sec.
- Operating temperature:	-25 to +70°C
- Shock:	200g

Figure 2.11: *Technical drawing of the BEI GyroChip Horizon*

Chapter 3

State Estimation with Kalman Filters

The Navigation of a mobile robot platform is described in Bar-Shalom and Li (1993) as an estimation problem.[1] The state of the platform on which diverse sensors may be located is to be estimated. Here, the state means the values of different parameters of the mobile robot, for example the position or the velocity. Then, estimation is to infer the state from indirect, inaccurate and uncertain observations.

In the first section of this chapter, the *Kalman filter (KF)* which is an *optimal estimator* is introduced. ”An *optimal estimator* is a computational algorithm that processes observations (measurements) to yield an estimate of a variable of interest, that optimizes a certain criterion.”[2] The Kalman filter is an estimator for the *linear-quadratic-Gaussian problem* which is the problem of estimating the actual state of a linear system perturbed by Gaussian noise with the help of measurements linearly related to the state, but corrupted by Gaussian white noise. ”The Kalman filter is statistically optimal with respect to any quadratic function of estimation error.”[3]

Optimal estimations has its advantages and disadvantages which are described in Bar-Shalom and Li (1993).[4] The utilization of the observations, the knowledge about the system and the appropriate disturbances is an advantage, but the Kalman filter is sensitive to modeling errors like any optimal technique. In view of this, a clear understanding of the assumptions is important and, therefore, they are presented in detail in this chapter.

In the second section, the *extended Kalman filter (EKF)* is discussed. The extended Kalman filter enhances the Kalman filter to non-linear systems and measurements that are non-linearly related to the state. This technique provides a better fitting of the system model which is used by the Kalman filter for the actual robot system.

[1](BAR-SHALOM AND LI, 1993), p. 2.

[2]Ibid.

[3](MOHINDER AND ANDREWS, 1993), p. 1.

[4](BAR-SHALOM AND LI, 1993), p. 3.

3.1 Kalman filters (KF)

In 1960, Rudolph E. Kalman published an article in which he described a mathematical method for the linear filtering of discrete data.[5] This set of equations, which is known as the Kalman filter, provides an efficient recursive solution for the least-squares method. It serves to estimate the state of a system on the basis of measurements with superimposed random errors. Hence, the Kalman filter provides an estimate for past, present and future system-states even if the precise internal structure of the system in question is unknown.

3.1.1 Introduction - examples of three simple Kalman filters in comparison

Kalman filters are used in a lot of different areas of science and technology such as tracking of objects in image processing or localisation of vehicles in mobile robotics. Based on three simple examples from the field of robotics, it is shown how this filter is used and which results can be expected from its application. The following examples deal solely with the self-localisation of a robot.

In the self-localisation of a mobile robot, particular attention has to be paid to the following problem: No measuring device is free of errors, i.e. the measurements float or are disturbed by noise. Moreover, sensors with different levels of resolution and precision are used. The fusion and processing of sensory data can be done by the Kalman filter. The examples are restricted to the *MonoRob* scenario (see below) to make the introduction to Kalman filtering as clear as possible. For the sake of simplicity, the Kalman filter will be considered as a ”black box” in this introduction. A more complex system will be discussed in later sections.

The *MonoRob* (see **figure 3.1**) is a one-dimensional robot which is able to measure not only its own position but also the distances to different features in its environment. A feature can be a characteristic or an object of the real world that can be obtained from the sensory data, e.g. the corner of a wall or a reflective label for a laser range finder.

In the experimental set-up of this work, laser range finders with centimetre resolutions are used for distance measurements. Position measurements, e.g. those gained by GPS (**G**lobal **P**ositioning **S**ystem), have rather imprecise resolutions of roughly ten metres. As can be seen, there are extremely different accuracies that can turn up in the same system.

For the simulation with the *MonoRob* a variance of 10 units is assumed for the accuracy of the position measurement. The measurement of the distance varies by one unit. The robot moves with a constant velocity for a time period of 100 units from position zero to position 100. The position, velocity and measurements of the simulation are shown in **figure 3.2**. The features are

[5](KALMAN, 1960), pp. 35-45.

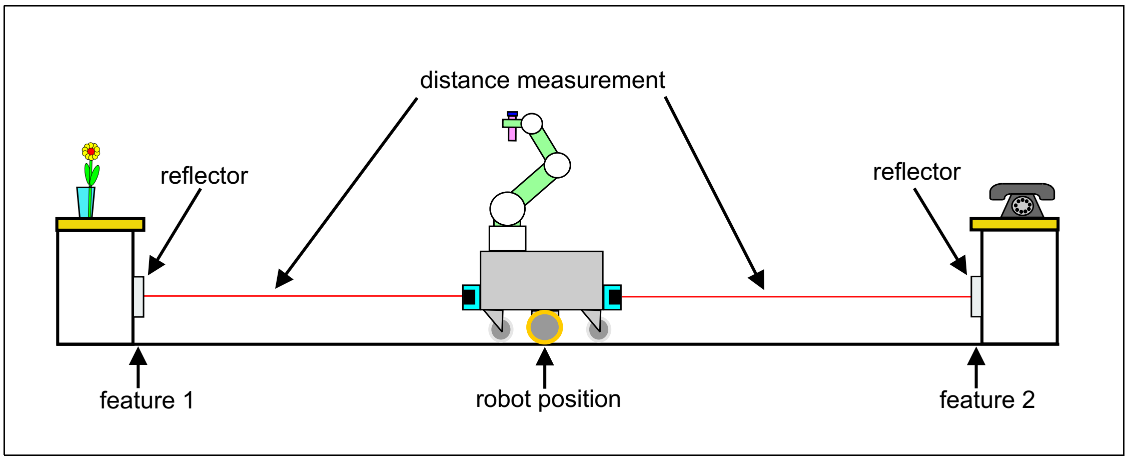


Figure 3.1: *MonoRob - One dimensional robot with distance measurement to known features*

situated at positions 20, 40, 60 and 80. The robot moves on a predetermined trajectory. Effects like slippage are neglected in this simulation for the sake of simplicity.

It needs to be discussed what a Kalman filter does with these data. At first, a model for the whole system must be created as a prerequisite so that the Kalman filter can operate.

The first model does not know any features. This system is, like all others that follow, a linear one. This means that the next state of the system $\vec{x}_{t+1}$ can be calculated by a linear transformation from the present state $\vec{x}_t$.

$$\vec{x}_{t+1} = F\vec{x}_t + \vec{w}_t$$

F is a transformation matrix and $\vec{w}_t$ is the noise at time t. In the first simple model, the system-state vector is two-dimensional, i.e it is based on the present position and the velocity of the robot.

$$\vec{x}_t = \begin{pmatrix} position(t) \\ velocity(t) \end{pmatrix}$$

The transformation into the next state results from the following equation:

$$position(t+1) = position(t) + \Delta t \cdot velocity(t)$$

Applied to the system-state, the equation looks as follows:

$$\vec{x}_{t+1} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \cdot \vec{x}_t + \vec{w}$$

The vector $\vec{w}$ stands for the noise of the process. The measured values can be described by the following equation:

$$\vec{z}_t = \begin{pmatrix} 1 & 0 \end{pmatrix} \cdot \vec{x}_t + \vec{v}_t$$

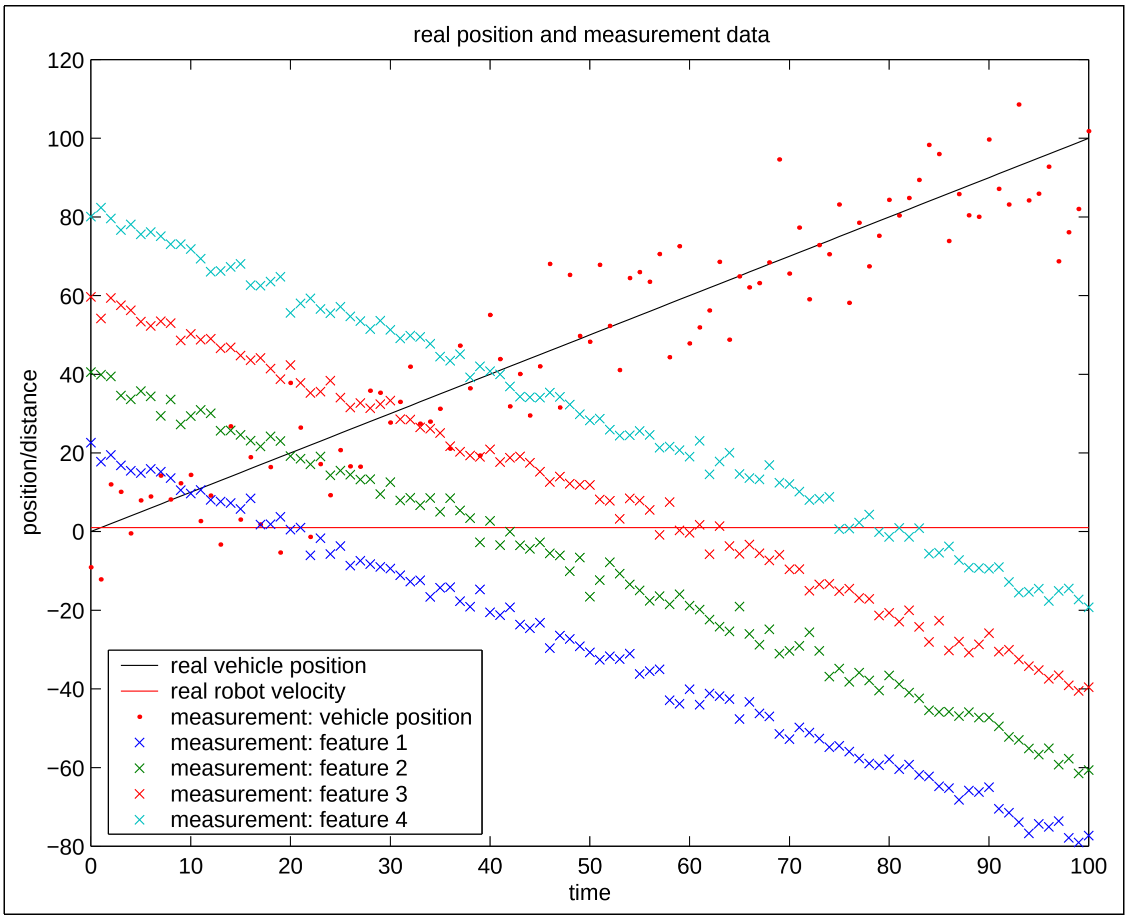


Figure 3.2: *MonoRob - Simulation of the movement and measurement data*

As mentioned above, the vector $\vec{v}_t$ is a noise term, which in this case affects the measurement of the position and therefore results in the corresponding inaccuracy. In this case, $\vec{z}_t$ and $\vec{v}_t$ are still one-dimensional. Starting from the initial state, this description of the system is utilized by the Kalman filter for the estimation of the state vector $\vec{x}_t$. The output of the Kalman filter, given the above system, is shown in **figure 3.3**.

The result, however, is still not satisfactory and seems to be similar to a moving average. Because of that, the original model is extended by the distance measurement of the robot in relation to the features. Firstly, the state vector must be supplemented by additional data.

Now two features are added to the experimental design so that the distances between the robot and these features can be determined.

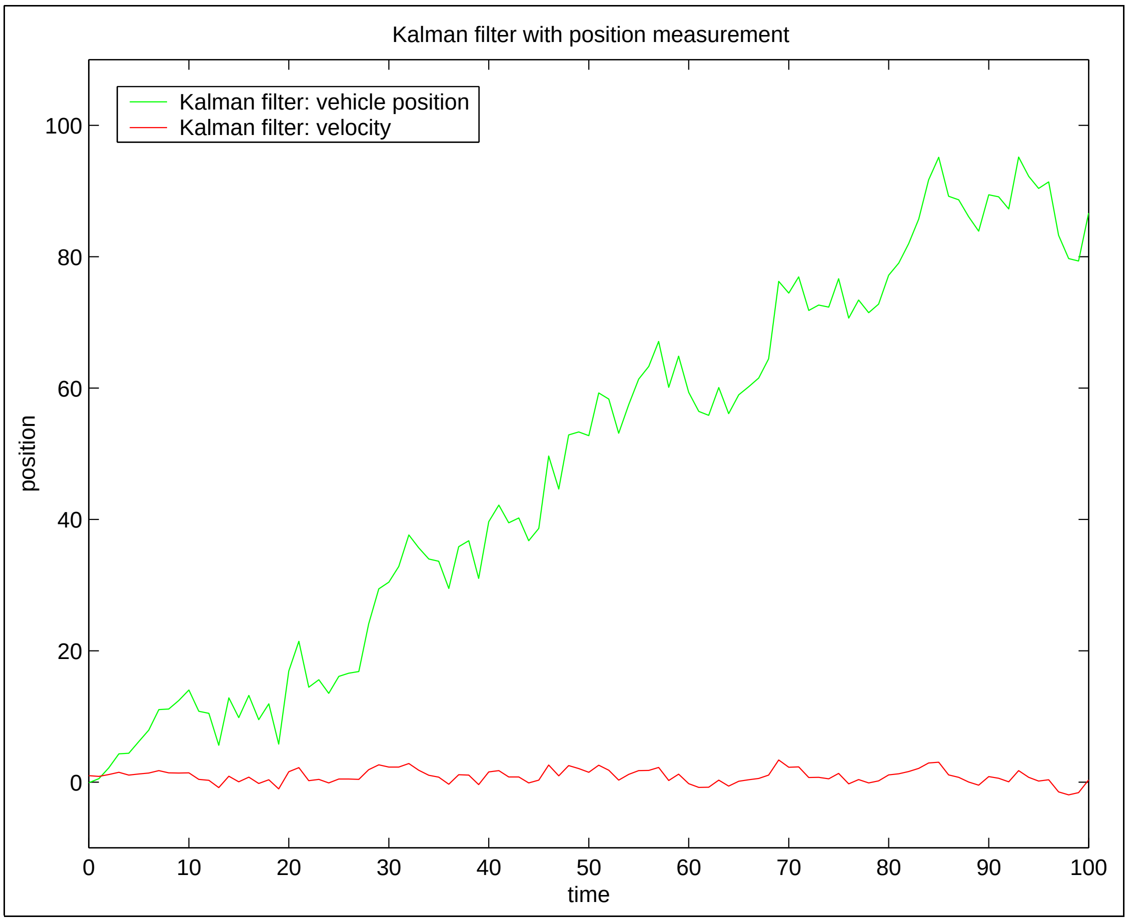


Figure 3.3: *MonoRob - Result of the Kalman filter with measurement of the position*

The new state vector looks as follows:

$$\vec{x}_t = \begin{pmatrix} position(t) \\ velocity(t) \\ positionFeature1 \\ positionFeature2 \end{pmatrix}$$

Considering the features are fixed, i.e. they do not change their positions, the equation for the transformation turns into:

$$\vec{x}_{t+1} = \begin{pmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \vec{x}_t + \vec{w}$$

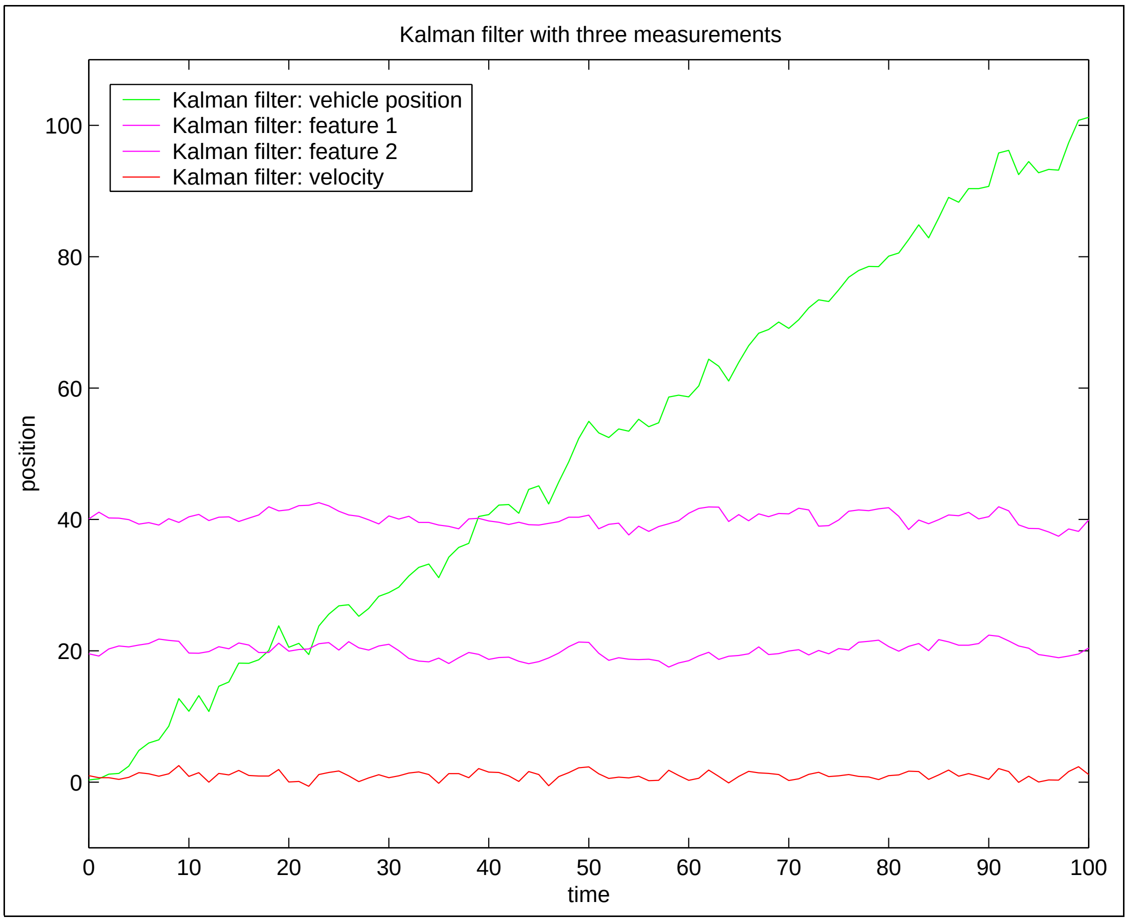


Figure 3.4: *MonoRob - Result of the Kalman filter with measurements of the robot's position and its distances from the two features*

The new measurement vector $\vec{z}_t$ thus becomes:

$$\vec{z}_t = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix} \cdot \vec{x}_t + \vec{v}_t$$

Figure 3.4 shows the output of the Kalman filter, which works with the data from the first example but is extended by data measuring the robot's distance to the features that are situated at positions 20 and 40.

One can clearly see that in this case the estimate of the position and velocity correspond much better to the simulated movement. If the number of distance measurements is increased by another two features, the result is further improved (**figure 3.5**). It is easier to perceive the improvement in **figure 3.6**.

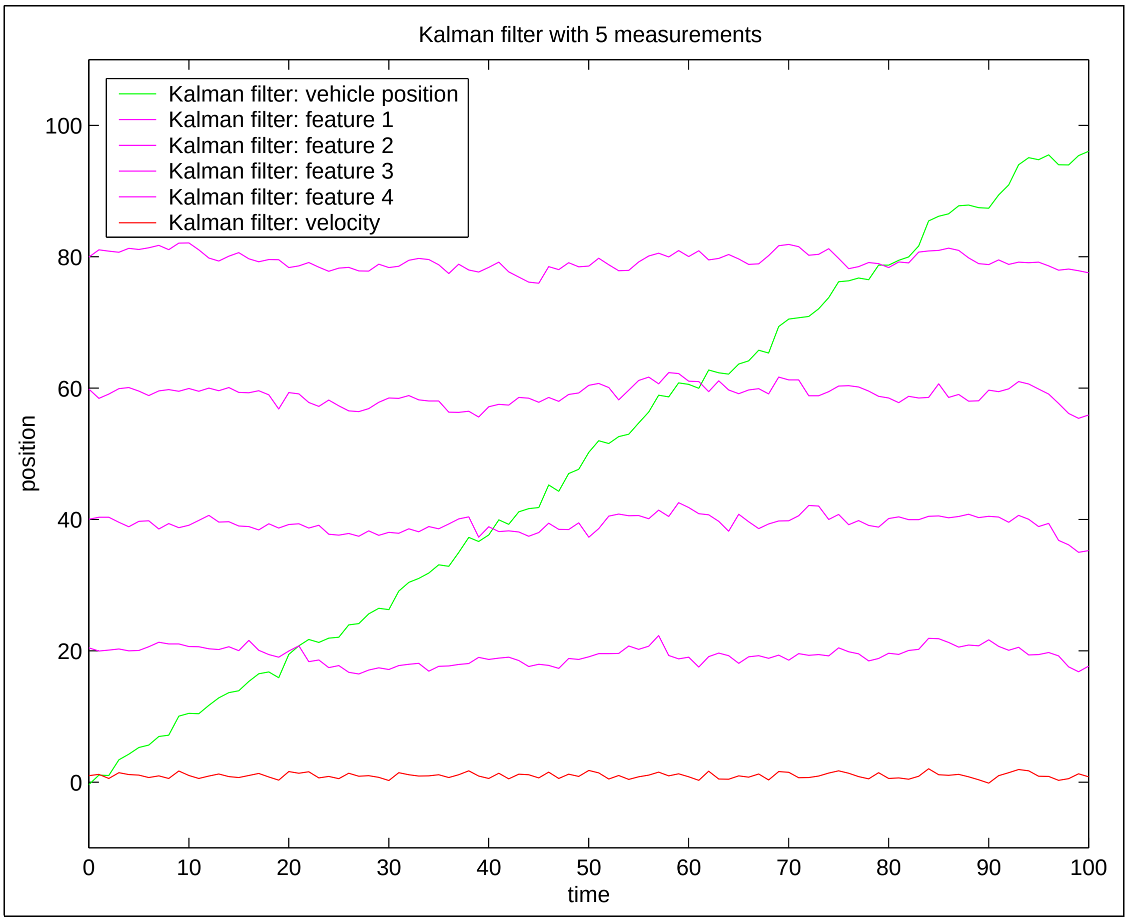


Figure 3.5: *MonoRob - Result of the Kalman filter with measurements of the robot's position and its distances from the four features*

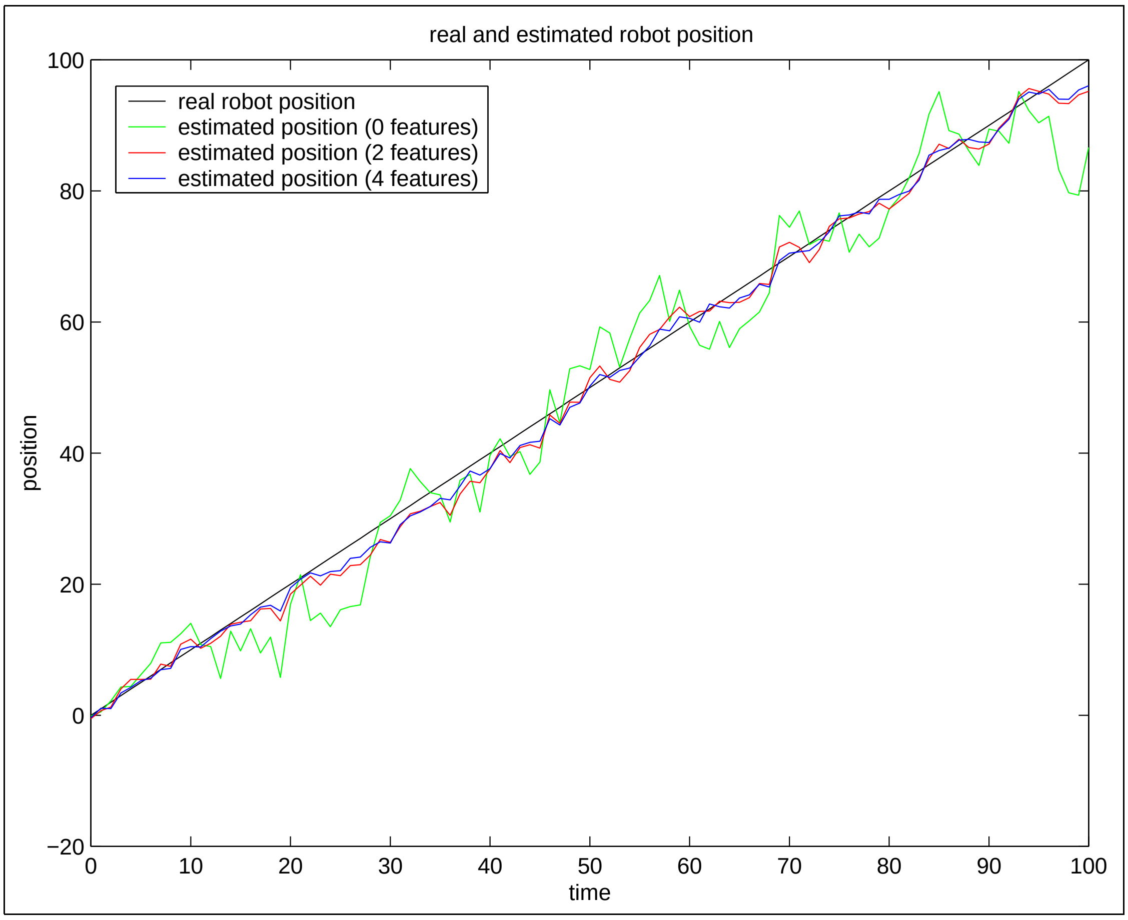


Figure 3.6: *MonoRob - Comparison between the real position and three different Kalman filter estimates*

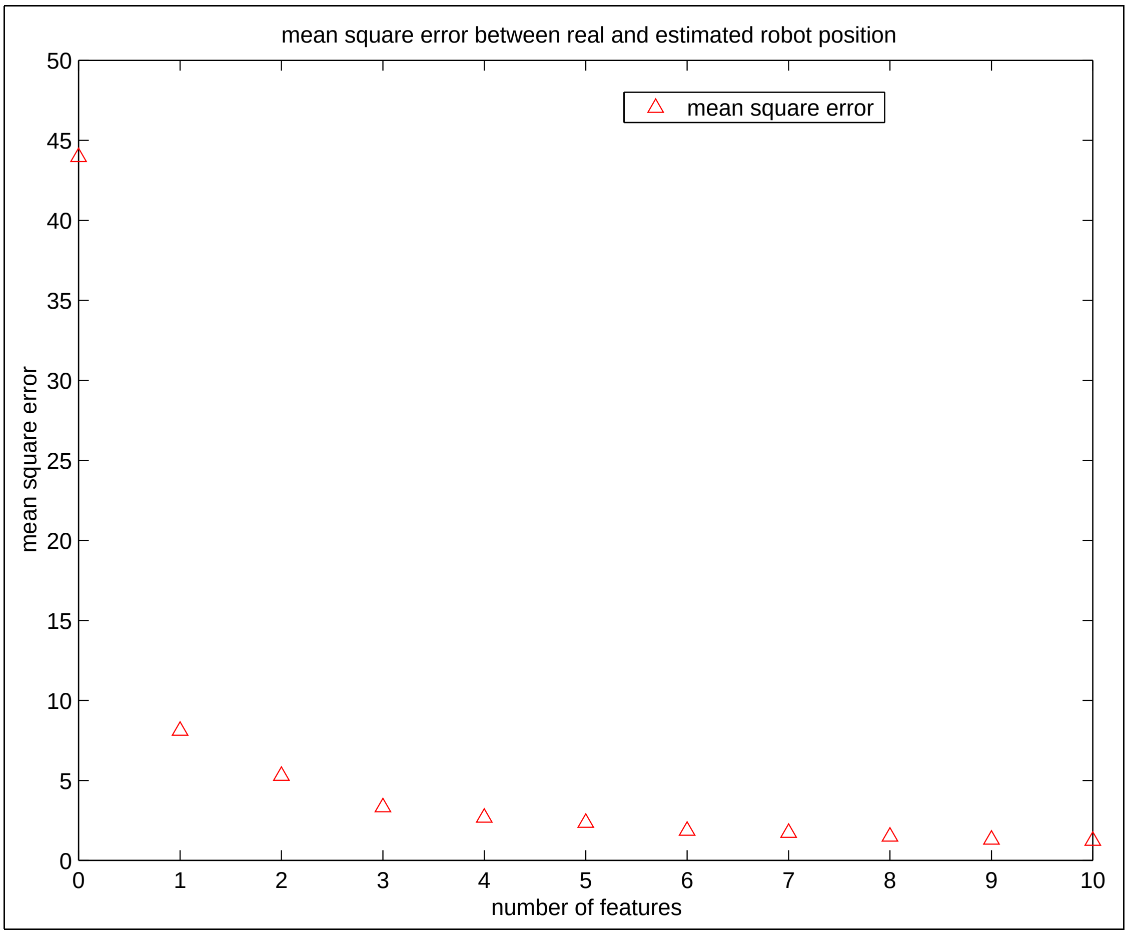


Figure 3.7: *MonoRob - Mean square error between the estimate of the position and the real position*

Increasing the number of features reduces the mean square error between the estimation and the real system-state. This is exemplary shown for the robot position in **figure 3.7**.

These short examples show that a Kalman filter can combine measurements of different kinds and quality and that it is able to estimate the current system-state. Until now, the Kalman filter has just been treated as a black box which receives data and outputs the system-state. In the following sections, it will be explained what the internal design of the Kalman filter looks like and how precise its results can be.

3.1.2 Modelling of discrete linear systems

Regarding the Kalman filter as a black box is based on the assumption of a linear world model. It is possible to have a time-continuous or a time-discrete system. Now the latter case is looked at more closely.

A world model described by a time-discrete linear system is given by a state vector $\vec{x}_t$,

$$\vec{x}_t = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}_t$$

representing all the required internal parameters, and a system equation

$$\boxed{\vec{x}_{t+1} = F\vec{x}_t + B\vec{u}_t + \vec{w}_t} \tag{3.1}$$

realizing the projection of the system-state from time step t to time step $t+1$. The matrix F is called the system matrix. $\vec{u}_t$ is the control input of the system with matrix B transforming it to influence the system-state. In the following, the addend $B\vec{u}_t$ is dropped since this does not result in a loss of generality. The noise that affects the system at each time step is represented by the random variable $\vec{w}_t$.

All measurements of the system are combined in the measurement vector $\vec{z}_t$:

$$\vec{z}_t = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix}_t$$

This vector is given as the result of the equation of the measurement

$$\boxed{\vec{z}_t = H\vec{x}_t + \vec{v}_t} \tag{3.2}$$

where each value of the measurement vector is based on a linear combination of the current system-state and superimposed noise.

For a better understanding, some assumptions on the noise variables $\vec{w}_t$ and $\vec{v}_t$, used in the Kalman filter, are needed. $\vec{w}_t$ represents the statistical part of the system and $\vec{v}_t$ the statistical part of the measurement. They are assumed to be white uncorrelated noise and can therefore be described by Gaussian distributions with zero-mean $\mathcal{N}(0, Q)$ where $Q = E\{\vec{w}_t\vec{w}_t^T\} =$

diag$(q_1 \dots q_n)$ and $\mathcal{N}(0, R)$ where $R = E\{\vec{v}_t \vec{v}_t^T\} = $ diag$(r_1 \dots r_m)$. Q and R are positive semi-definite. The system noise and measuring error are not correlated, which means

$$E\{\vec{w}_t \vec{v}_t^T\} = \mathbf{0}$$

3.1.3 Kalman equation

The iterative cycle of the Kalman filtering procedure consists of two main steps. The first step consists of predicting the current state from the estimation of the last state. This prediction is called the *a priori* estimation $\hat{x}_t^-$ of the system-state and is calculated according to the system equation (3.1), in which the last estimate $\hat{x}_{t-1}^+$ substitutes for $\vec{x}_{t-1}$:

$$\boxed{\hat{x}_t^- = F\hat{x}_{t-1}^+} \tag{3.3}$$

The noise term is set to zero. Because of its zero-mean characteristic, this is the most likely assumption for the prediction above.

The second main step in the Kalman filtering procedure is the *a posteriori* assumption of the system-state. This output of the filter is the correction of the predicted system-state due to knowledge of the current measurement. It derives from the weighted sum of the system's *a priori* estimation and the measurement vector:

$$\hat{x}_t^+ = K_t'\hat{x}_t^- + K_t\vec{z}_t \tag{3.4}$$

K_t' and K_t are weighting matrices which are unknown at this point of the derivation. With equation (3.2) substituted into equation (3.4), this leads to

$$\begin{aligned} \hat{x}_t^+ &= K_t'\hat{x}_t^- + K_t(H\vec{x}_t + \vec{v}_t) \\ &= K_t'\hat{x}_t^- + K_tH\vec{x}_t + K_t\vec{v}_t \end{aligned} \tag{3.5}$$

Now two estimation errors (*a priori* and *a posteriori*) can be defined:

$$\tilde{x}_t^- = \hat{x}_t^- - \vec{x}_t \tag{3.6}$$

$$\tilde{x}_t^+ = \hat{x}_t^+ - \vec{x}_t \tag{3.7}$$

Using equations (3.6) and (3.7) in the weighted sum for the a posteriori estimation, equation (3.5) delivers:

$$\begin{aligned} \vec{x}_t + \tilde{x}_t^+ &= K_t'(\vec{x}_t + \tilde{x}_t^-) + K_tH\vec{x}_t + K_t\vec{v}_t \\ &= K_t'\vec{x}_t + K_t'\tilde{x}_t^- + K_tH\vec{x}_t + K_t\vec{v}_t \\ \tilde{x}_t^+ &= K_t'\vec{x}_t + K_tH\vec{x}_t - \vec{x}_t + K_t'\tilde{x}_t^- + K_t\vec{v}_t \\ &= \left[K_t' + K_tH - \mathbb{1}\right]\vec{x}_t + K_t'\tilde{x}_t^- + K_t\vec{v}_t \end{aligned}$$

The Kalman filter should decrease the *a posteriori* error $\tilde{x}_t^+$ between the real and estimated state. In order to achieve this, the expected value is set to zero

$$\begin{aligned} E\left\{\tilde{x}_t^+\right\} \overset{!}{=} 0 &= E\left\{\hat{x}_t^+ - \vec{x}_t\right\} \\ &= E\left\{\left[K_t' + K_t H - \mathbb{1}\right]\vec{x}_t\right\} + \underbrace{E\left\{K_t'\tilde{x}_t^-\right\}}_{=0 \text{ assumed}} + \underbrace{E\left\{K_t\vec{v}_t\right\}}_{=0} \end{aligned} \tag{3.8}$$

Consequently, the first term of the sum of equation (3.8) must be set to zero

$$\begin{aligned} \left[K_t' + K_t H - \mathbb{1}\right]\vec{x}_t &= 0 \qquad |\forall \vec{x}_t \\ \Rightarrow K_t' &= \mathbb{1} - K_t H \end{aligned} \tag{3.9}$$

Simple insertion of (3.9) into the original *a posteriori* estimation leads to

$$\hat{x}_t^+ = \left(\mathbb{1} - K_t H\right)\hat{x}_t^- + K_t\vec{z}_t$$

and the **linear recursive estimation function (Kalman equation)**

$$\boxed{\hat{x}_t^+ = \hat{x}_t^- + K_t\underbrace{\left[\vec{z}_t - H\hat{x}_t^-\right]}_{\text{innovation}}} \tag{3.10}$$

With equations (3.3) and (3.10) *a priori* and *a posteriori* estimations of the system-state are possible on condition that K_t is known. K_t is called the Kalman matrix or Kalman gain. Because $H\hat{x}_t^-$ is a prediction of the measurement, equation (3.10) interprets $\hat{x}_t^+$ as the weighted sum of the state prediction plus the error between the measurement and its estimate.

3.1.4 Kalman matrix

In this section, the Kalman matrix K_t is determined.[6] This matrix provides the best weight for the innovation (see equation (3.10)).

The Kalman matrix should minimize the mean square error between the real and the estimated system-state

$$\epsilon = E\left\{\left(\vec{x}_t - \hat{x}_t^+\right)^T\left(\vec{x}_t - \hat{x}_t^+\right)\right\}$$

The minimization of ϵ with the help of K_t is equivalent to the minimization of the trace of P_t^+, i.e. the covariance matrix of the *a posteriori* estimation error:

$$P_t^+ = E\left\{\left(\vec{x}_t - \hat{x}_t^+\right)\left(\vec{x}_t - \hat{x}_t^+\right)^T\right\} \tag{3.11}$$

[6]Compare with (KUMMERT, 2001).

With the Kalman equation (3.10), equation (3.11) needs to be rewritten as follows:

$$P_t^+ = E\left\{\left(\vec{x}_t - \hat{x}_t^- - K_t\left[\vec{z}_t - H\hat{x}_t^-\right]\right)\left(\vec{x}_t - \hat{x}_t^- - K_t\left[\vec{z}_t - H\hat{x}_t^-\right]\right)^T\right\}$$

Inserting the measurement model equation (3.2) leads to

$$\begin{aligned}
P_t^+ &= E\left\{\left(\vec{x}_t - \hat{x}_t^- - K_t\left[H\vec{x}_t + \vec{v}_t - H\hat{x}_t^-\right]\right)\left(\vec{x}_t - \hat{x}_t^- - K_t\left[H\vec{x}_t + \vec{v}_t - H\hat{x}_t^-\right]\right)^T\right\} \\
&= E\left\{\left(\vec{x}_t - \hat{x}_t^- - K_tH\vec{x}_t - K_t\vec{v}_t + K_tH\hat{x}_t^-\right)\left(\vec{x}_t - \hat{x}_t^- - K_tH\vec{x}_t - K_t\vec{v}_t + K_tH\hat{x}_t^-\right)^T\right\} \\
&= E\left\{\left((\mathbb{1} - K_tH)\vec{x}_t - (\mathbb{1} - K_tH)\hat{x}_t^- - K_t\vec{v}_t\right)\left((\mathbb{1} - K_tH)\vec{x}_t - (\mathbb{1} - K_tH)\hat{x}_t^- - K_t\vec{v}_t\right)^T\right\} \\
&= E\left\{\left((\mathbb{1} - K_tH)\left(\vec{x}_t - \hat{x}_t^-\right) - K_t\vec{v}_t\right)\left((\mathbb{1} - K_tH)\left(\vec{x}_t - \hat{x}_t^-\right) - K_t\vec{v}_t\right)^T\right\} \\
&= E\left\{\left((\mathbb{1} - K_tH)\left(\vec{x}_t - \hat{x}_t^-\right) - K_t\vec{v}_t\right)\left(\left(\vec{x}_t - \hat{x}_t^-\right)^T(\mathbb{1} - K_tH)^T - (K_t\vec{v}_t)^T\right)\right\} \\
&= E\Big\{(\mathbb{1} - K_tH)\left(\vec{x}_t - \hat{x}_t^-\right)\left(\vec{x}_t - \hat{x}_t^-\right)^T(\mathbb{1} - K_tH)^T - (\mathbb{1} - K_tH)\left(\vec{x}_t - \hat{x}_t^-\right)(K_t\vec{v}_t)^T \\
&\qquad - (K_t\vec{v}_t)\left(\vec{x}_t - \hat{x}_t^-\right)^T(\mathbb{1} - K_tH)^T + (K_t\vec{v}_t)(K_t\vec{v}_t)^T\Big\} \\
&= E\left\{(\mathbb{1} - K_tH)\left(\vec{x}_t - \hat{x}_t^-\right)\left(\vec{x}_t - \hat{x}_t^-\right)^T(\mathbb{1} - K_tH)^T\right\} - E\left\{(\mathbb{1} - K_tH)\left(\vec{x}_t - \hat{x}_t^-\right)(K_t\vec{v}_t)^T\right\} \\
&\qquad - E\left\{(K_t\vec{v}_t)\left(\vec{x}_t - \hat{x}_t^-\right)^T(\mathbb{1} - K_tH)^T\right\} + E\left\{(K_t\vec{v}_t)(K_t\vec{v}_t)^T\right\}
\end{aligned}$$

With the covariance matrix of the *a priori* estimation error being

$$P_t^- = E\left\{\left(\vec{x}_t - \hat{x}_t^-\right)\left(\vec{x}_t - \hat{x}_t^-\right)^T\right\} \tag{3.12}$$

further simplification is possible:

$$\begin{aligned}
P_t^+ &= (\mathbb{1} - K_tH)P_t^-(\mathbb{1} - K_tH)^T - \underbrace{(\mathbb{1} - K_tH)E\left\{\left(\vec{x}_t - \hat{x}_t^-\right)\vec{v}_t^T\right\}K_t^T}_{\vec{v}_t \text{ is white noise} \Rightarrow E\{\cdot\}=0} \\
&\qquad - \underbrace{K_tE\left\{\vec{v}_t\left(\vec{x}_t - \hat{x}_t^-\right)^T\right\}(\mathbb{1} - K_tH)^T}_{\vec{v}_t \text{ is white noise} \Rightarrow E\{\cdot\}=0} + K_t\underbrace{E\left\{\vec{v}_t\vec{v}_t^T\right\}}_{R}K_t^T \\
&= (\mathbb{1} - K_tH)P_t^-(\mathbb{1} - K_tH)^T + K_tRK_t^T
\end{aligned} \tag{3.13}$$

Now the solution for the minimizing problem $min(trace(P_t^+))$ must be found. An approach by means of calculus of variations can be applied which results in the assumption: The optimal matrix K_t is known and therefore $K_t \leq K_t + \delta K_t$.[7]

$$\begin{aligned}
&\text{trace}\left[(\mathbb{1} - K_t H)\, P_t^- \,(\mathbb{1} - K_t H)^T + K_t R K_t^T\right] \\
&\quad \leq \quad \text{trace}\left[(\mathbb{1} - (K_t + \delta K_t)\, H)\, P_t^- \,(\mathbb{1} - (K_t + \delta K_t)\, H)^T + (K_t + \delta K_t)\, R\, (K_t + \delta K_t)^T\right]
\end{aligned}$$

$$\begin{aligned}
&\text{trace}\left[(\mathbb{1} - K_t H)\, P_t^- \,(\mathbb{1} - K_t H)^T + \underbrace{K_t R K_t^T}_{\text{drops out}}\right] \\
&\quad \leq \quad \text{trace}\Big[(\mathbb{1} - K_t H - \delta K_t H)\, P_t^- \,(\mathbb{1} - K_t H - \delta K_t H)^T + \underbrace{K_t R K_t^T}_{\text{drops out}} \\
&\qquad\qquad + K_t R \delta K_t^T + \delta K_t R K_t^T + \delta K_t R \delta K_t^T\Big]
\end{aligned}$$

$$\begin{aligned}
&\text{trace}\left[\underbrace{(\mathbb{1} - K_t H)\, P_t^- \,(\mathbb{1} - K_t H)^T}_{\text{drops out}}\right] \\
&\quad \leq \quad \text{trace}\Big[\underbrace{(\mathbb{1} - K_t H)\, P_t^- \,(\mathbb{1} - K_t H)^T}_{\text{drops out}} - (\mathbb{1} - K_t H)\, P_t^- \,(\delta K_t H)^T - (\delta K_t H)\, P_t^- \,(\mathbb{1} - K_t H)^T \\
&\qquad\qquad + (\delta K_t H)\, P_t^- \,(\delta K_t H)^T + K_t R \delta K_t^T + \delta K_t R K_t^T + \delta K_t R \delta K_t^T\Big]
\end{aligned}$$

$$\begin{aligned}
0 \quad &\leq \quad \text{trace}\Big[-\underbrace{(\mathbb{1} - K_t H)\, P_t^- H^T \delta K_t^T}_{\text{see appendix A.1, equation (A.1)}} - \delta K_t H P_t^- \,(\mathbb{1} - K_t H)^T + \delta K_t H P_t^- H^T \delta K_t^T \\
&\qquad + \underbrace{K_t R \delta K_t^T}_{\text{see appendix A.1, equation (A.2)}} + \delta K_t R K_t^T + \delta K_t R \delta K_t^T\Big]
\end{aligned}$$

$$\begin{aligned}
0 \quad &\leq \quad \text{trace}\Big[-\delta K_t H P_t^- \,(\mathbb{1} - K_t H)^T - \delta K_t H P_t^- \,(\mathbb{1} - K_t H)^T + \delta K_t R K_t^T \\
&\qquad + \delta K_t R K_t^T + \delta K_t H P_t^- H^T \delta K_t^T + \delta K_t R \delta K_t^T\Big]
\end{aligned}$$

$$0 \quad \leq \quad \text{trace}\left[-2\delta K_t H P_t^- \,(\mathbb{1} - K_t H)^T + 2\delta K_t R K_t^T + \delta K_t H P_t^- H^T \delta K_t^T + \delta K_t R \delta K_t^T\right] \qquad (3.14)$$

The last two terms of (3.14) are quadratic expressions and thus greater than zero. I follows from

[7] For transformation rules, see appendix A.1.

this that the trace of the first two summands must be greater than zero as well.

$$\begin{aligned} 0 &= \text{trace}\left[-2\delta K_t H P_t^- \left(\mathbb{1} - K_t H\right)^T + 2\delta K_t R K_t^T\right] \\ &= -2\,\text{trace}\left[\delta K_t \left(H P_t^- \left(\mathbb{1} - K_t H\right)^T - R K_t^T\right)\right] \end{aligned}$$

As assumed in the calculus of variations, $\delta K_t \geq 0$ is valid. If the trace is expected to be zero, the second factor in the square brackets has to be zero as well.

$$\begin{aligned} \mathbb{0} &= H P_t^- \left(\mathbb{1} - K_t H\right)^T - R K_t^T \\ &= H P_t^- - H P_t^- H^T K_t^T - R K_t^T \\ H P_t^- H^T K_t^T + R K_t^T &= H P_t^- \\ \left(H P_t^- H^T + R\right) K_t^T &= H P_t^- \\ K_t \left(H P_t^- H^T + R\right)^T &= H P_t^- \end{aligned} \tag{3.15}$$

It follows from $\left(H P_t^- H^T + R\right)^T = \left(H P_t^- H^T + R\right)$ that

$$K_t = H P_t^- \left(H P_t^- H^T + R\right)^{-1}$$

This is known as the **gain equation of the Kalman filter**:

$$\boxed{K_t = P_t^- H^T \left(H P_t^- H^T + R\right)^{-1}} \tag{3.16}$$

3.1.5 Covariance matrix of the *a posteriori* estimation

As seen in the derivation of the Kalman filter's gain equation, the covariance of the estimation error equation (3.11) can be calculated on the basis of the filter gain, the measurement model, the measurement noise covariance matrix and the covariance matrix of the *a priori* estimation. Equation (3.13) can be used for this, but it can be simplified if it is rewritten as

$$P_t^+ = P_t^- - P_t^- \left(K_t H\right)^T - K_t H P_t^- + K_t H P_t^- \left(K_t H\right)^T + K_t R K_t^T \tag{3.17}$$

and the transformation of (3.15) modified as follows:

$$\begin{aligned} \mathbb{0} &= H P_t^- - H P_t^- H^T K_t^T - R K_t^T \\ H P_t^- &= H P_t^- H^T K_t^T + R K_t^T && \mid \text{pre-multiplication with } K_t \\ K_t H P_t^- &= K_t H P_t^- H^T K_t^T + K_t R K_t^T && (3.18) \end{aligned}$$

The right-hand side of equation (3.18) can be found in (3.17), where it is replaced by the left-hand side of equation (3.18). Equation (3.17) then collapses, and this leads to:

$$\begin{aligned} P_t^+ &= P_t^- - P_t^- \left(K_t H\right)^T \\ &= P_t^- \left(\mathbb{1} - \left(K_t H\right)^T\right) \end{aligned}$$

This equation is called the **updated state covariance**

$$\boxed{P_t^+ = \left(\mathbb{1} - K_t H\right) P_t^-} \tag{3.19}$$

Bar-Shalom and Li (1993) state ”that the *covariance of the state* is the same as the *covariance of the estimation error*... [as] a consequence of the fact that the estimate is the conditional mean”[8].

3.1.6 Covariance matrix of the *a priori* estimation

The covariance matrix P_t^- of the *a priori* estimation $\hat{x}_t^-$ is still not known. The system equation (equation (3.1)) and the *a priori* estimation for the system-state (equation (3.3)) are inserted into the general formulation of the *a priori* estimation for the covariance of the state vector (equation (3.12)):

$$\begin{aligned} P_{t+1}^- &= E\left\{\left(F\vec{x}_t + \vec{w}_t - F\hat{x}_t^+\right)\left(F\vec{x}_t + \vec{w}_t - F\hat{x}_t^+\right)^T\right\} \\ &= E\left\{\left(F\left(\vec{x}_t - \hat{x}_t^+\right) + \vec{w}_t\right)\left(F\left(\vec{x}_t - \hat{x}_t^+\right) + \vec{w}_t\right)^T\right\} \\ &= E\left\{F\left(\vec{x}_t - \hat{x}_t^+\right)\left(\vec{x}_t - \hat{x}_t^+\right)^T F^T + F\left(\vec{x}_t - \hat{x}_t^+\right)\vec{w}_t^T + \vec{w}_t\left(\vec{x}_t - \hat{x}_t^+\right)^T F^T + \vec{w}_t\vec{w}_t^T\right\} \\ &= \underbrace{E\left\{F\left(\vec{x}_t - \hat{x}_t^+\right)\left(\vec{x}_t - \hat{x}_t^+\right)^T F^T\right\}}_{FP_t^+F^T} \\ &\quad + \underbrace{E\left\{F\left(\vec{x}_t - \hat{x}_t^+\right)\vec{w}_t^T\right\}}_{\vec{w}_t \text{ is white noise} \Rightarrow E\{\cdot\}=0} + \underbrace{E\left\{\vec{w}_t\left(\vec{x}_t - \hat{x}_t^+\right)^T F^T\right\}}_{\text{see left}} \\ &\quad + \underbrace{E\left\{\vec{w}_t\vec{w}_t^T\right\}}_{Q} \end{aligned}$$

Since white noise is uncorrelated by definition, both summands in the middle of the equation above can be dropped. This leads to the following description of the *a priori* estimation for the covariance matrix:

$$\boxed{P_{t+1}^- = FP_t^+F^T + Q} \tag{3.20}$$

[8](Bar-Shalom and Li, 1993), p. 210.

This covariance matrix is often called **state prediction covariance** or **predicted state co-variance**.

3.1.7 Block diagram of the Kalman filter

The elements of the Kalman filter that have been derived can be represented in a block diagram (**figure 3.8**).

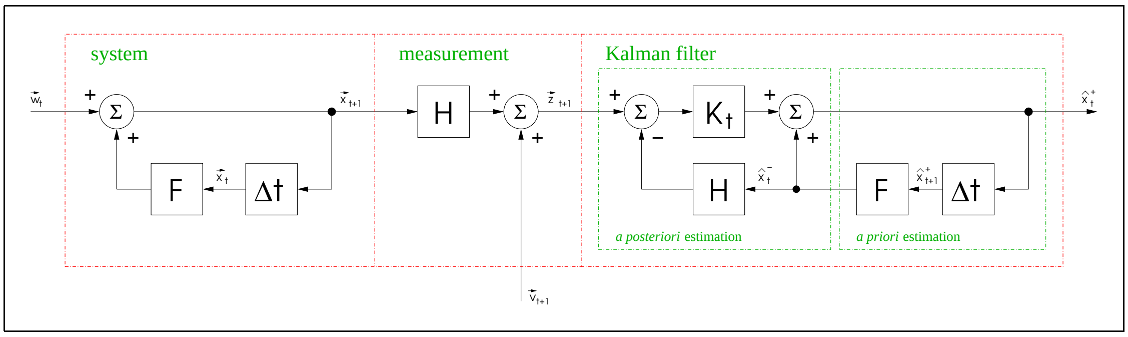


Figure 3.8: *Block diagram of the Kalman filter*

3.1.8 How to use the Kalman filter

The procedure of applying the Kalman filter consists of the following steps:

1. Definition of the world model $\Rightarrow F$
2. Definition of the measurement model $\Rightarrow H$
3. The system and the measurement contain statistical parts $\Rightarrow \vec{w}_t$ and $\vec{v}_t$ can be defined as white noise, i.e. the covariance matrices Q and R can have the following shape:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

   With the knowledge of the system and measurement variances, these can be set as the diagonal elements of Q and R since these are appropriate choices. This leads to a better performance of the Kalman filter. [9]

[9]See web page of (WELCH AND BISHOP, 2000), chapter 1, section *Filter Parameters and Tuning* for a short discussion and chapter 3 section *The Simulations* for examples.

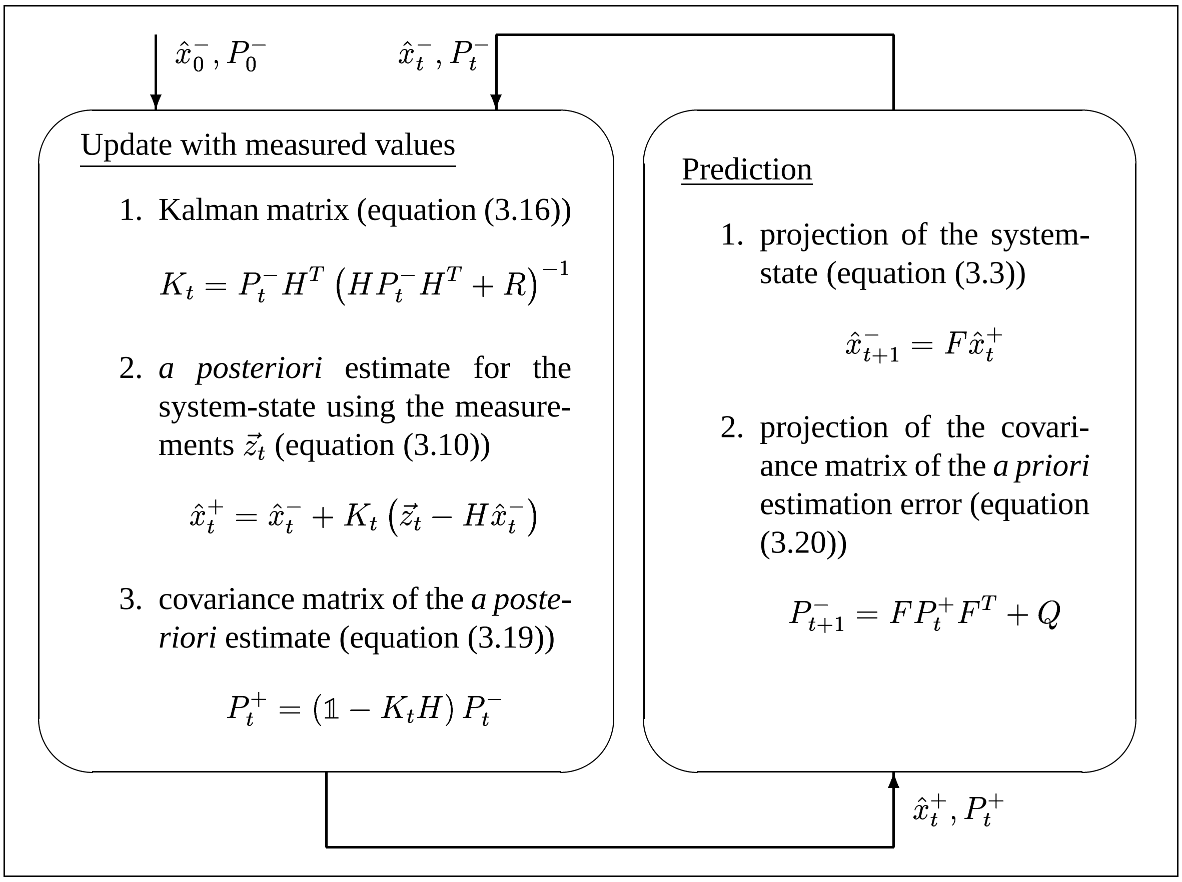

$\hat{x}_0^-, P_0^-$
$\hat{x}_t^-, P_t^-$
Update with measured values
1. Kalman matrix (equation (3.16))
$K_t = P_t^- H^T \left(H P_t^- H^T + R\right)^{-1}$
2. *a posteriori* estimate for the system-state using the measurements $\vec{z}_t$ (equation (3.10))
$\hat{x}_t^+ = \hat{x}_t^- + K_t \left(\vec{z}_t - H\hat{x}_t^-\right)$
3. covariance matrix of the *a posteriori* estimate (equation (3.19))
$P_t^+ = \left(\mathbb{1} - K_t H\right) P_t^-$
Prediction
1. projection of the system-state (equation (3.3))
$\hat{x}_{t+1}^- = F\hat{x}_t^+$
2. projection of the covariance matrix of the *a priori* estimation error (equation (3.20))
$P_{t+1}^- = F P_t^+ F^T + Q$
$\hat{x}_t^+, P_t^+$

Figure 3.9: *Kalman filter - sequence of operations*

4. An initial estimate for $\hat{x}_0^-$ and P_0^- has to be made

5. Iteration as shown in **figure 3.9**

3.2 Extended Kalman Filter (EKF)

As pointed out in section 3.1, the use of a Kalman filter for state estimation is appropriate for linear systems. Non-linear systems have to be linearized, or an extended version of the Kalman filter has to be applied. The fundamental idea of extending the standard Kalman filter in order to deal with non-linear systems was first proposed by Stanley F. Schmidt.[10] In the following sections, it is presented how this extended version is derived from the original Kalman filter.

[10](SCHMIDT, 1970), (SCHMIDT, 1976).

3.2.1 Modelling of discrete non-linear systems

The foundation of the standard Kalman filter is the modeling of a linear system and its measurements with the equations (3.1) and (3.2):

$$\vec{x}_{t+1} = F\vec{x}_t + C\vec{u}_t + \vec{w}_t$$
$$\vec{z}_t = H\vec{x}_t + \vec{v}_t$$

As in section 3.1, $\vec{x}_t$ denotes the current system-state and $\vec{z}_t$ the measurement at time t.

Hence a non-linear system and measurement model can be described in a similar fashion:

$$\vec{x}_{t+1} = f\left(\vec{x}_t, \vec{u}_t, \vec{w}_t\right) \tag{3.21}$$
$$\vec{z}_t = h\left(\vec{x}_t, \vec{v}_t\right) \tag{3.22}$$

In this case, the non-linear function f in equation (3.21) relates the system-state of the previous time step to the current time step. $\vec{u}_t$ is the controller input and $\vec{w}_t$ is a random zero-mean variable representing the process noise. In the measurement equation (3.22) the non-linear function h relates the current system-state $\vec{x}_t$ to the measurement $\vec{z}_t$. The random variable $\vec{v}_t$ represents the noise of the measurement process. It also has zero mean.

3.2.2 Kalman equation

In order to derive the Kalman equation for the standard Kalman filter, the first step is to get an *a priori* estimation of the system state. In analogy to equation (3.3)

$$\hat{x}_t^- = F\hat{x}_{t-1}^+$$

the system-state estimation $\hat{x}_t^-$ for the non-linear system is:

$$\hat{x}_t^- = f\left(\hat{x}_{t-1}^+, \vec{u}_t, 0\right) \tag{3.23}$$

As in the linear case, the value of the system noise $\vec{w}_t$ is unknown at time t and therefore is approximated with zero because of its zero mean.

For the extended Kalman filter, another estimation of the approximation of the measurement at time step t is made. In the standard Kalman filter, this estimation is the second summand in the innovation term (see equation (3.10)). The estimation in a linear system is

$$\hat{z}_t = H\hat{x}_{t-1}^-$$

and becomes

$$\hat{z}_t = h\left(\hat{x}_{t-1}^-, 0\right) \tag{3.24}$$

for the non-linear system. Similarly to the *a priori* system-state estimation, a zero-mean characteristic of the noise of the non-linear system is assumed.

With the help of equation (3.23) a Taylor series for the non-linear system equation (3.21) is derivable for the evaluation point $\vec{x} = \hat{x}_t^+$, $\vec{u} = \vec{u}_t$ and $\vec{w} = 0$:

$$\begin{aligned}\vec{x}_{t+1} &= f\left(\hat{x}_{t-1}^+, \vec{u}_t, 0\right)\\ &+ \left.\frac{\partial f}{\partial \vec{x}}\right|_{\vec{x}=\hat{x}_t^+} \cdot \left(\vec{x}_t - \hat{x}_t^+\right) + \left.\frac{\partial f}{\partial \vec{u}}\right|_{\vec{u}=\vec{u}_t} \cdot \left(\vec{u}_t - \vec{u}_t\right) + \left.\frac{\partial f}{\partial \vec{w}}\right|_{\vec{w}=0} \cdot \left(\vec{w}_t - 0\right)\\ &+ \ldots \text{(higher order terms)}\end{aligned}$$

If equation (3.23) is applied to the first summand of the Taylor series and the higher order terms are left out, the value for $\vec{x}_{t+1}$ can be approximated as follows:

$$\vec{x}_{t+1} \approx \hat{x}_{t+1}^- + F_t^{\text{Jacobian}} \cdot \left(\vec{x}_t - \hat{x}_t^+\right) + W_t^{\text{Jacobian}} \cdot \vec{w}_t \tag{3.25}$$

with

$$F_t^{\text{Jacobian}} = \left.\frac{\partial f}{\partial \vec{x}}\right|_{\vec{x}=\hat{x}_t^+} = \left.\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}\right|_{\vec{x}=\hat{x}_t^+}$$

and

$$W_t^{\text{Jacobian}} = \left.\frac{\partial f}{\partial \vec{w}}\right|_{\vec{w}=0} = \left.\begin{bmatrix} \frac{\partial f_1}{\partial w_1} & \frac{\partial f_1}{\partial w_2} & \cdots & \frac{\partial f_1}{\partial w_n} \\ \frac{\partial f_2}{\partial w_1} & \frac{\partial f_2}{\partial w_2} & \cdots & \frac{\partial f_2}{\partial w_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial w_1} & \frac{\partial f_n}{\partial w_2} & \cdots & \frac{\partial f_n}{\partial w_n} \end{bmatrix}\right|_{\vec{w}=0}$$

In the same way, a Taylor series for the measurement equation (3.22) is generated for the evaluation point $\vec{x} = \hat{x}_t^-$ and $\vec{v} = 0$:

$$\vec{z}_t = h\left(\hat{x}_t^-, 0\right) + \left.\frac{\partial h}{\partial \vec{x}}\right|_{\vec{x}=\hat{x}_t^-} \cdot \left(\vec{x}_t - \hat{x}_t^-\right) + \left.\frac{\partial h}{\partial \vec{v}}\right|_{\vec{v}=0} \cdot \left(\vec{v} - 0\right) + \ldots \text{(higher order terms)}$$

Using equation (3.24) and again leaving out the higher order terms leads to an approximation of $\vec{z}_t$

$$\vec{z}_t \approx \hat{z}_t + H_t^{\text{Jacobian}} \cdot \left(\vec{x}_t - \hat{x}_t^-\right) + V_t^{\text{Jacobian}} \cdot \vec{v}_t \tag{3.26}$$

with

$$H_t^{\text{Jacobian}} = \left.\frac{\partial h}{\partial \vec{x}}\right|_{\vec{x}=\hat{x}_t^-} = \left.\begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial_2}{\partial x_2} & \cdots & \frac{\partial h_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_m}{\partial x_1} & \frac{\partial h_m}{\partial x_2} & \cdots & \frac{\partial h_m}{\partial x_n} \end{bmatrix}\right|_{\vec{x}=\hat{x}_t^-}$$

and

$$V_t^{\text{Jacobian}} = \left.\frac{\partial h}{\partial \vec{v}}\right|_{\vec{v}=0} = \left.\begin{bmatrix} \frac{\partial h_1}{\partial v_1} & \frac{\partial h_1}{\partial v_2} & \cdots & \frac{\partial h_1}{\partial v_m} \\ \frac{\partial h_2}{\partial v_1} & \frac{\partial h_2}{\partial v_2} & \cdots & \frac{\partial h_2}{\partial v_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_m}{\partial v_1} & \frac{\partial h_m}{\partial v_2} & \cdots & \frac{\partial h_m}{\partial v_m} \end{bmatrix}\right|_{\vec{v}=0}$$

With the help of equations (3.25) and (3.26), both the prediction error

$$\epsilon_{\vec{x}_t} = \vec{x}_t - \hat{x}_t^- \tag{3.27}$$

and the innovation

$$\epsilon_{\vec{z}_t} = \vec{z}_t - \hat{z}_t \tag{3.28}$$

can be formulated. After re-indexing equation (3.25) and inserting it into the prediction error equation (3.27), it is possible to give the approximate value

$$\begin{aligned} \epsilon_{\vec{x}_t} &\approx \hat{x}_t^- + F_{t-1}^{\text{Jacobian}} \cdot \left(\vec{x}_{t-1} - \hat{x}_{t-1}^+\right) + W_{t-1}^{\text{Jacobian}} \cdot \vec{w}_{t-1} - \hat{x}_t^- \\ &= F_{t-1}^{\text{Jacobian}} \cdot \left(\vec{x}_{t-1} - \hat{x}_{t-1}^+\right) + W_{t-1}^{\text{Jacobian}} \cdot \vec{w}_{t-1} \\ &\equiv \hat{\epsilon}_{\vec{x}_t} \end{aligned}$$

It follows from inserting equation (3.26) into the measurement residue equation (3.28) that

$$\begin{aligned} \epsilon_{\vec{z}_t} &\approx \hat{z}_t + H_t^{\text{Jacobian}} \cdot \left(\vec{x}_t - \hat{x}_t^-\right) + V_t^{\text{Jacobian}} \cdot \vec{v}_t - \hat{z}_t \\ &= H_t^{\text{Jacobian}} \cdot \left(\vec{x}_t - \hat{x}_t^-\right) + V_t^{\text{Jacobian}} \cdot \vec{v}_t \\ &= H_t^{\text{Jacobian}} \cdot \epsilon_{\vec{x}_t} + V_t^{\text{Jacobian}} \cdot \vec{v}_t \\ &\equiv \hat{\epsilon}_{\vec{z}_t} \end{aligned}$$

If the two noise terms are re-defined as

$$\begin{aligned} \vec{\eta}_t &= W_t^{\text{Jacobian}} \cdot \vec{w}_t \\ \vec{\gamma}_t &= H_t^{\text{Jacobian}} \cdot \vec{v}_t \end{aligned}$$

the error estimations are re-written as

$$\hat{\epsilon}_{\vec{x}_t} = F_{t-1}^{\text{Jacobian}} \cdot \left(\vec{x}_{t-1} - \hat{x}_{t-1}^+\right) + \vec{\eta}_t \tag{3.29}$$

$$\hat{\epsilon}_{\vec{z}_t} = H_t^{\text{Jacobian}} \cdot \hat{\epsilon}_{\vec{x}_t} + \vec{\gamma}_t \tag{3.30}$$

Looking at the comparison in **table 3.1**, Welch' and Bishop's (2000) idea of a "[...] (hypothetical) Kalman filter to estimate the prediction error [...]"[11] $\hat{\epsilon}_{\vec{x}_t}$ is applied to equations (3.29) and (3.30).

[11]See web page of (WELCH AND BISHOP, 2000), chapter 2, section *The Computational Origins of the Filter*.

system description of the Kalman filter	prediction error
$\vec{x}_{t+1} = F_t \vec{x}_t + \vec{w}_t$	$\hat{\epsilon}_{\vec{x}_t} = F_{t-1}^{\text{Jacobian}} \cdot \left(\vec{x}_{t-1} - \hat{x}_{t-1}^{+}\right) + \vec{\eta}_t$
measurement description of the Kalman filter	measurement residue
$\vec{z}_t = H_t \vec{x}_t + \vec{v}_t$	$\hat{\epsilon}_{\vec{z}_t} = H_t^{\text{Jacobian}} \cdot \hat{\epsilon}_{\vec{x}_t} + \vec{\gamma}_t$

Table 3.1: *Comparison between the Kalman filter system and measurement descriptions and the prediction error and innovation estimations*

Such an estimate is used ”[...] to obtain the *a posteriori* state estimate for the original non-linear process [...]”[12]. The *a posteriori* state estimate which is gained in this process is

$$\hat{x}_t^{+} = \hat{x}_t^{-} + \hat{\epsilon}_{\vec{x}_t} \tag{3.31}$$

At this point, a closer look at the (hypothetical) Kalman filter is sensible. This Kalman filter requires the following assumption about the probability distributions. The random variables $\hat{\epsilon}_{\vec{x}_t}$, η_t and γ_t are approximated as normally distributed. Yet in practice they are not because of the respective non-linear transformations of the original random variables. ”The EKF is simply an ad hoc state estimator that only approximates the optimality of Bayes’ rule by linearization.”[13] Hence the assumptions’ formal descriptions are:

$$\begin{aligned} p\left(\hat{\epsilon}_{\vec{x}_t}\right) &\sim \mathcal{N}\left(0, E\left\{\hat{\epsilon}_{\vec{x}_t}\hat{\epsilon}_{\vec{x}_t}^T\right\}\right) \\ p\left(\eta_t\right) &\sim \mathcal{N}\left(E\left\{\eta_t\right\}, E\left\{\eta_t\eta_t^T\right\}\right) \\ p\left(\gamma_t\right) &\sim \mathcal{N}\left(E\left\{\gamma_t\right\}, E\left\{\gamma_t\gamma_t^T\right\}\right) \end{aligned}$$

where

$$\begin{aligned} E\left\{\eta_t\right\} &= 0 \\ E\left\{\eta_t\eta_t^T\right\} &= E\left\{W_t^{\text{Jacobian}} \cdot \vec{w}_t \cdot \left(W_t^{\text{Jacobian}} \cdot \vec{w}_t\right)^T\right\} \\ &= E\left\{W_t^{\text{Jacobian}} \cdot \vec{w}_t \cdot \vec{w}_t^T \cdot \left(W_t^{\text{Jacobian}}\right)^T\right\} \\ &= W_t^{\text{Jacobian}} \cdot E\left\{\vec{w}_t \cdot \vec{w}_t^T\right\} \cdot \left(W_t^{\text{Jacobian}}\right)^T \\ &= W_t^{\text{Jacobian}} Q \left(W_t^{\text{Jacobian}}\right)^T \end{aligned}$$

[12]See web page of (WELCH AND BISHOP, 2000), chapter 2, section *The Computational Origins of the Filter*.
[13]Ibid., chapter 2, section *The Process to be Estimated*.

and

$$
\begin{aligned}
E\{\gamma_t\} &= 0 \\
E\{\gamma_t\gamma_t^T\} &= E\left\{V_t^{\text{Jacobian}} \cdot \vec{v}_t \cdot \left(V_t^{\text{Jacobian}} \cdot \vec{v}_t\right)^T\right\} \\
&= E\left\{V_t^{\text{Jacobian}} \cdot \vec{v}_t \cdot \vec{v}_t^T \cdot \left(V_t^{\text{Jacobian}}\right)^T\right\} \\
&= V_t^{\text{Jacobian}} \cdot E\left\{\vec{v}_t \cdot \vec{v}_t^T\right\} \cdot \left(V_t^{\text{Jacobian}}\right)^T \\
&= V_t^{\text{Jacobian}} R \left(V_t^{\text{Jacobian}}\right)^T
\end{aligned}
$$

Some variations of the extended Kalman filter have been developed by Julier and Uhlmann (1996), preserving the normal distributions throughout the non-linear transformations.[14]

To sum up, the Kalman equation for the (hypothetical) Kalman filter is

$$
\begin{aligned}
\hat{\epsilon}_{\vec{x}_t} &= 0 + K_t\left(\epsilon_{\vec{z}_t} - 0\right) \\
&= K_t\epsilon_{\vec{z}_t}
\end{aligned}
$$

In conjunction with equation (3.31), this leads to the following Kalman equation for the non-linear system:

$$
\begin{aligned}
\hat{x}_t^+ &= \hat{x}_t^- + K_t\epsilon_{\vec{z}_t} \\
&= \hat{x}_t^- + K_t\left(\vec{z}_t - \hat{z}_t\right) \\
&= \hat{x}_t^- + K_t\left(\vec{z}_t - h\left(\hat{x}_t^-, 0\right)\right) \qquad (3.32)
\end{aligned}
$$

With this result, the (hypothetical) Kalman filter is not really needed, and equation (3.32), which is similar to the Kalman equation for the linear system (3.10), is utilized for state estimation.

3.2.3 How to use the extended Kalman filter

As in section 3.1.8, the same sequence for filtering is used for the extended Kalman filter. In **figure 3.10** this procedure is shown with the appropriate substitutions for the error covariance matrices. The raised index *Jacobian* is abbreviated to J in order to make the figure easier to read.

Bar-Shalom and Li (1993)[15] present a flowchart which explains the operation sequence of one cycle of the extended Kalman filter in a more detailed way. **Figure 3.11** shows this flowchart adapted to the notation used so far. There are time-variant system and measurement functions f_t and h_t used, which require that the Jacobians need to be calculated for each time step. Some previously introduced calculations are fragmented to facilitate the understanding of the flowchart.

[14](JULIER AND UHLMANN, 1996).

[15](BAR-SHALOM AND LI, 1993), p. 387.

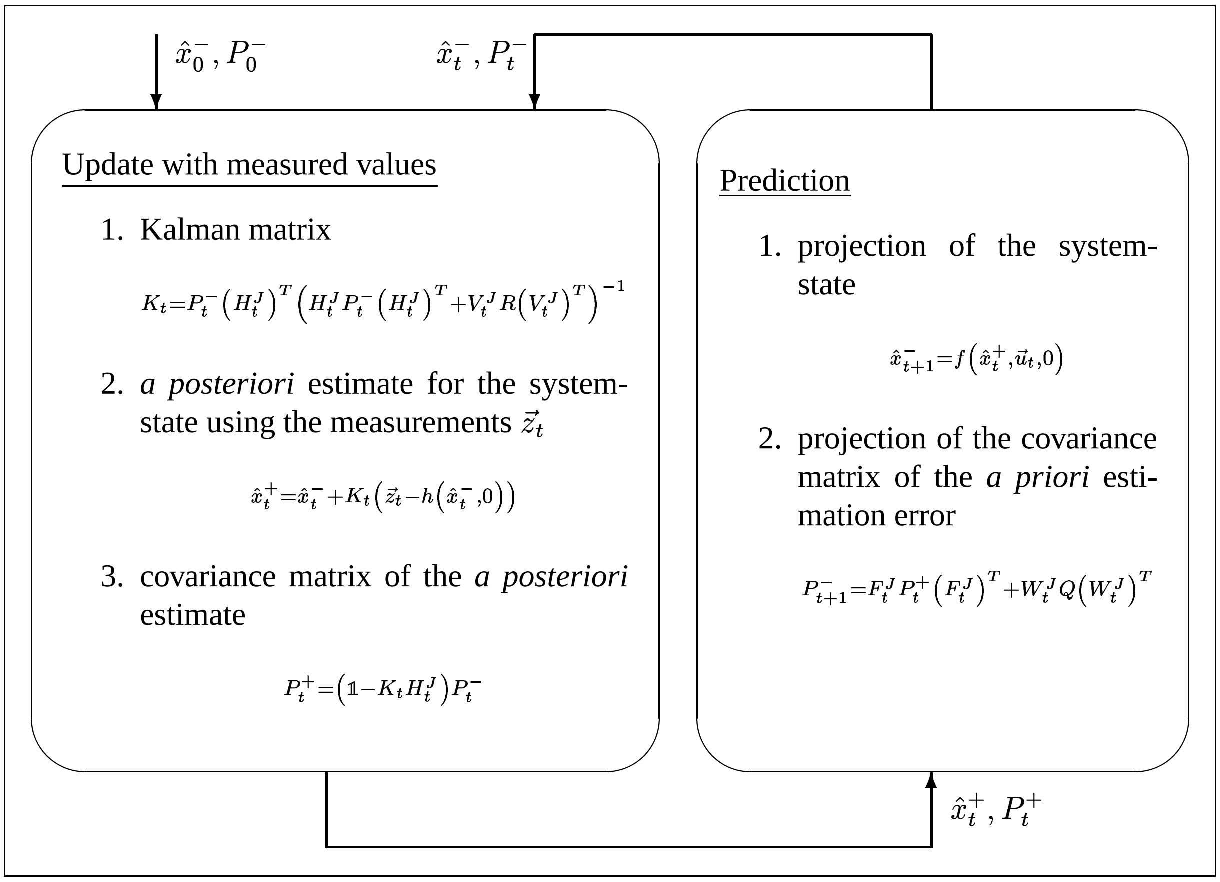

$\hat{x}_0^-, P_0^-$
$\hat{x}_t^-, P_t^-$
Update with measured values
1. Kalman matrix
$K_t = P_t^- \left(H_t^J\right)^T \left(H_t^J P_t^- \left(H_t^J\right)^T + V_t^J R \left(V_t^J\right)^T\right)^{-1}$
2. *a posteriori* estimate for the system-state using the measurements $\vec{z}_t$
$\hat{x}_t^+ = \hat{x}_t^- + K_t\left(\vec{z}_t - h\left(\hat{x}_t^-, 0\right)\right)$
3. covariance matrix of the *a posteriori* estimate
$P_t^+ = \left(\mathbb{1} - K_t H_t^J\right) P_t^-$
Prediction
1. projection of the system-state
$\hat{x}_{t+1}^- = f\left(\hat{x}_t^+, \vec{u}_t, 0\right)$
2. projection of the covariance matrix of the *a priori* estimation error
$P_{t+1}^- = F_t^J P_t^+ \left(F_t^J\right)^T + W_t^J Q \left(W_t^J\right)^T$
$\hat{x}_t^+, P_t^+$

Figure 3.10: *Extended Kalman filter - sequence of operations*

If put to use, the formulations which have been introduced for the standard and extended Kalman filter can cause numerical problems. For example, due to rounding errors, the properties of a covariance matrix, namely symmetry and positive definiteness, can be lost. Different implementations and variants of the Kalman filter such as *information filter, sequential updating* or *square root filtering* can be used to avoid or at least reduce numerical problems.[16] One example is the covariance update equation (3.19), which can be written in the following algebraical equivalent form

$$P_t^+ = \left[\mathbb{1} - W_t^{\text{Jacobian}} H_t^{\text{Jacobian}}\right] P_t^- \left[\mathbb{1} - W_t^{\text{Jacobian}} H_t^{\text{Jacobian}}\right]^T + W_t^{\text{Jacobian}} R_t \left(W_t^{\text{Jacobian}}\right)^T$$

This form is called the *Joseph form covariance update*. It is computationally more expensive but less sensitive to rounding errors. ”With the proper implementation of the products of three matrices it will preserve symmetry. Furthermore, since the only place it has a subtraction is in the term $\mathbb{1} - WH$, which appears ’squared’, this form of the covariance update has the property of preserving the positive definiteness of the resulting updated covariance.”[17] In the software implementation which will be explained later in this chapter, this form of covariance update is used.

[16](Bar-Shalom and Li, 1993), chapter 7.

[17]Ibid., p. 294.

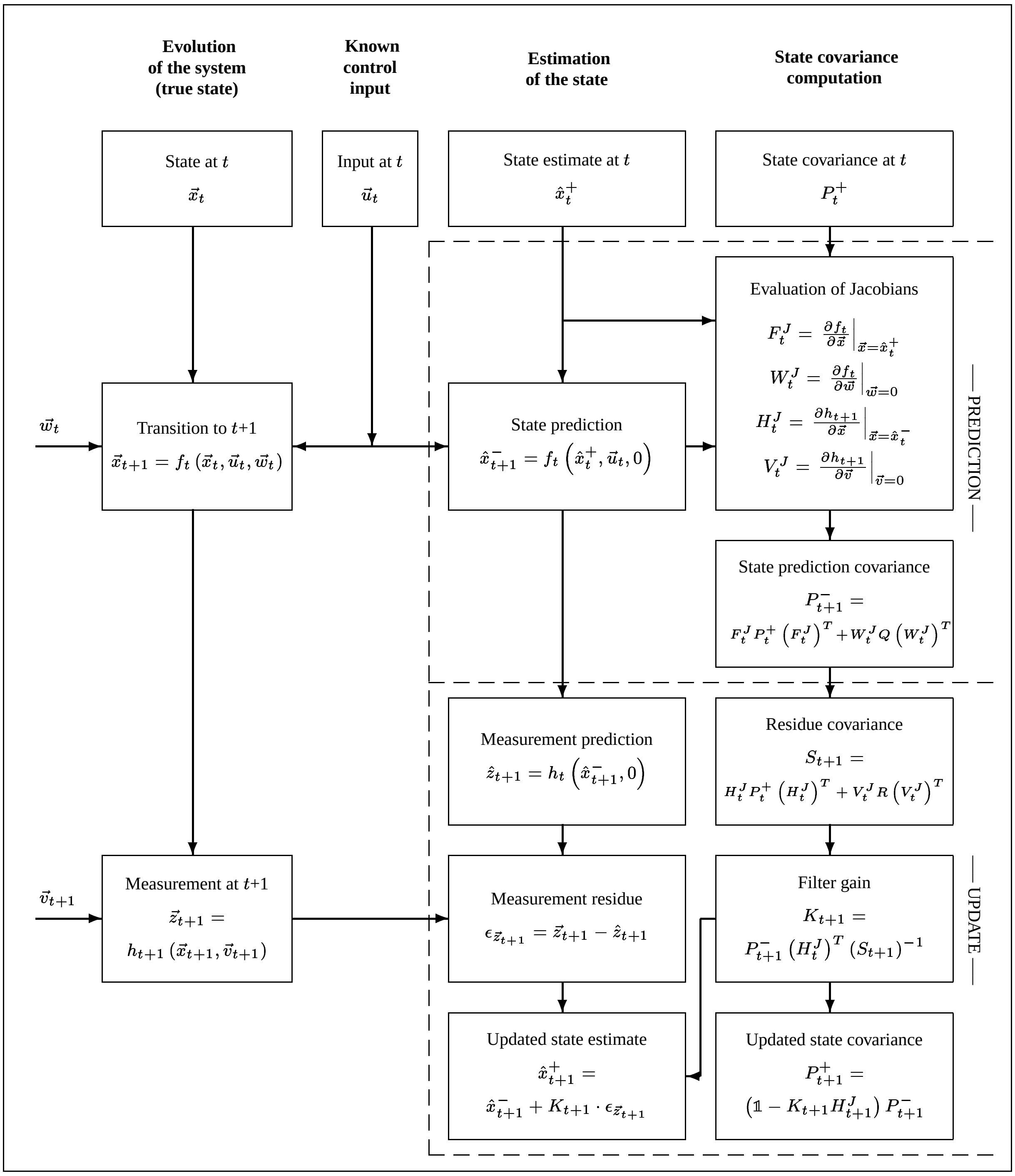

Evolution of the system (true state)
Known control input
Estimation of the state
State covariance computation
State at t
$\vec{x}_t$
Input at t
$\vec{u}_t$
State estimate at t
$\hat{x}_t^+$
State covariance at t
P_t^+
Evaluation of Jacobians
$F_t^J = \left.\frac{\partial f_t}{\partial \vec{x}}\right|_{\vec{x}=\hat{x}_t^+}$
$W_t^J = \left.\frac{\partial f_t}{\partial \vec{w}}\right|_{\vec{w}=0}$
$H_t^J = \left.\frac{\partial h_{t+1}}{\partial \vec{x}}\right|_{\vec{x}=\hat{x}_t^-}$
$V_t^J = \left.\frac{\partial h_{t+1}}{\partial \vec{v}}\right|_{\vec{v}=0}$
$\vec{w}_t$
Transition to t+1
$\vec{x}_{t+1} = f_t\left(\vec{x}_t, \vec{u}_t, \vec{w}_t\right)$
State prediction
$\hat{x}_{t+1}^- = f_t\left(\hat{x}_t^+, \vec{u}_t, 0\right)$
PREDICTION
State prediction covariance
$P_{t+1}^- =$
$F_t^J P_t^+ \left(F_t^J\right)^T + W_t^J Q \left(W_t^J\right)^T$
Measurement prediction
$\hat{z}_{t+1} = h_t\left(\hat{x}_{t+1}^-, 0\right)$
Residue covariance
$S_{t+1} =$
$H_t^J P_t^+ \left(H_t^J\right)^T + V_t^J R \left(V_t^J\right)^T$
$\vec{v}_{t+1}$
Measurement at t+1
$\vec{z}_{t+1} =$
$h_{t+1}\left(\vec{x}_{t+1}, \vec{v}_{t+1}\right)$
Measurement residue
$\epsilon_{\vec{z}_{t+1}} = \vec{z}_{t+1} - \hat{z}_{t+1}$
Filter gain
$K_{t+1} =$
$P_{t+1}^- \left(H_t^J\right)^T \left(S_{t+1}\right)^{-1}$
UPDATE
Updated state estimate
$\hat{x}_{t+1}^+ =$
$\hat{x}_{t+1}^- + K_{t+1} \cdot \epsilon_{\vec{z}_{t+1}}$
Updated state covariance
$P_{t+1}^+ =$
$\left(\mathbb{1} - K_{t+1} H_{t+1}^J\right) P_{t+1}^-$

Figure 3.11: *Flowchart of the extended Kalman filter (one cycle)*

Chapter 4

Self-Localisation of a Mobile Robot

After the Kalman filter has been suggested as a means of state estimation for a dynamic system in the previous chapter, it will now be shown how the Kalman filter can be used for the self-localisation of a mobile robot. This is done with respect to the mobile robot platform *GenBase II* which is used for the experiments of this work. By analyzing the kinematics[1], it will be shown that the *GenBase II* platform is a mobile robot with two degrees of freedom (DOF).[2] Because of this, the results of this chapter can be easily transferred to any mobile robot with the same kinematics.

In section 4.1 a general model of a robot with two degrees of freedom will be presented. This includes considerations about the coordinate transformations of the mobile robot platform and an in detail explanation about the robot kinematics. In section 4.2 a system model for the *GenBase II* robot is set up with the obtained knowledge about the kinematics of a robot with two degrees of freedom. The derived system is used in conjunction with the extended Kalman filter to solve the self-localisation problem. The last section of this chapter describes the software which implements the developed self-localisation system.

4.1 Model of a mobile robot with two degrees of freedom (DOF)

This section is divided into two parts. Firstly, a closer look is taken on the coordinate transformations of the mobile robot *GenBase II*. The knowledge about the transformations allows to convert

[1]"In general, *kinematics* is the science of objects in motion, including aspects of position, velocity and acceleration. This includes, in particular, all of the robot's geometric and time-based physical properties." (ARKIN, 1998), p. 89.

[2]See section 4.1.2 for details on the term *degrees of freedom*.

coordinates from one coordinate system into those of another coordinate system, e.g. from the laser range finder's coordinate system to the robot's base coordinate system. The information is used to provide suitable measurements for the extended Kalman filter and to tell the manipulator on the mobile platform where its base is in world coordinates. Secondly, a kinematic model for mobile robot's with two degrees of freedom is discussed.

4.1.1 The coordinate transformations of the mobile robot platform *GenBase II*

In 1987, Muir and Neuman formulated the kinematic equations of motion for a variety of wheeled mobile robots.[3] Their kinematic models depend heavily upon the concepts of manipulator kinematics. There are differences between the kinematics of a mobile robot and a manipulator arm which are stated in McKerrow (1993).[4]

The kinematics can be decomposed into *internal* and *external* kinematics. The internal kinematics describe the relationships within the robot and the external kinematics characterize the relationships between the robot and its environment.

In this section the assignment of different coordinate frames to the robot and the transformations between the frames are discussed. Since the joint order of a mobile robot may not be obvious, the Denavit-Hartenberg convention for coordinate frame assignment leads to ambiguous transformation matrices.[5] This problem can be avoided using the Sheth-Uicker (1971) convention.[6] In the Sheth-Uicker convention, coordinate frames are assigned to both ends of each link. When these frames coincide the joint variable is zero. The coordinate frames for the mobile robot *GenBase II* are shown in **figure 4.1**. Some coincident frames are not drawn in this figure.

The links of a wheeled robot are the floor, the robot body and the steering links. They are connected by three joints (the wheel, the steering axis and the robot's centre point). Since in all frames the z axis is vertical, it is neglected in a two-dimensional analysis. The floor coordinate frame is stationary and serves as a reference frame for the moving robot. Since all wheels of a mobile robot are in contact with the ground, the set of transformations forms a multiple closed-link chain. In contrast to that, a manipulator arm has a single open-link chain.

Homogeneous transformations[7] are used to describe the conversions between different coordinate systems. The transformations are simple because in the two-dimensional case rotation only occurs about the z axis.

[3](MUIR AND NEUMAN, 1987).
[4](MCKERROW, 1993), pp. 402-403.
[5]See (DENAVIT AND HARTENBERG, 1955) and (DENAVIT AND HARTENBERG, 1964) for details on the Denavit-Hartenberg convention.
[6](SHETH AND UICKER, 1971).
[7]See appendix A.2 for details on homogeneous transformations.

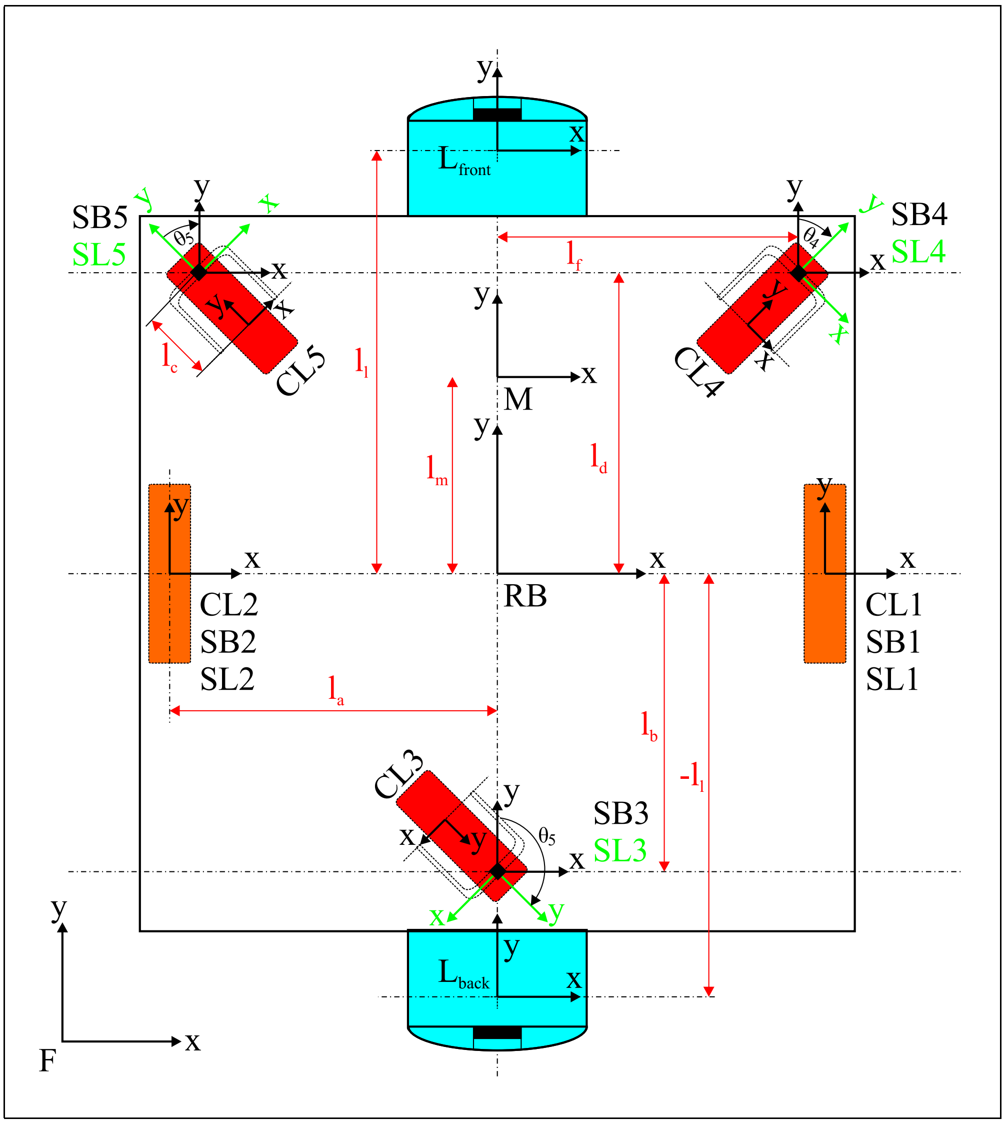


Figure 4.1: *GenBase II - coordinate systems*

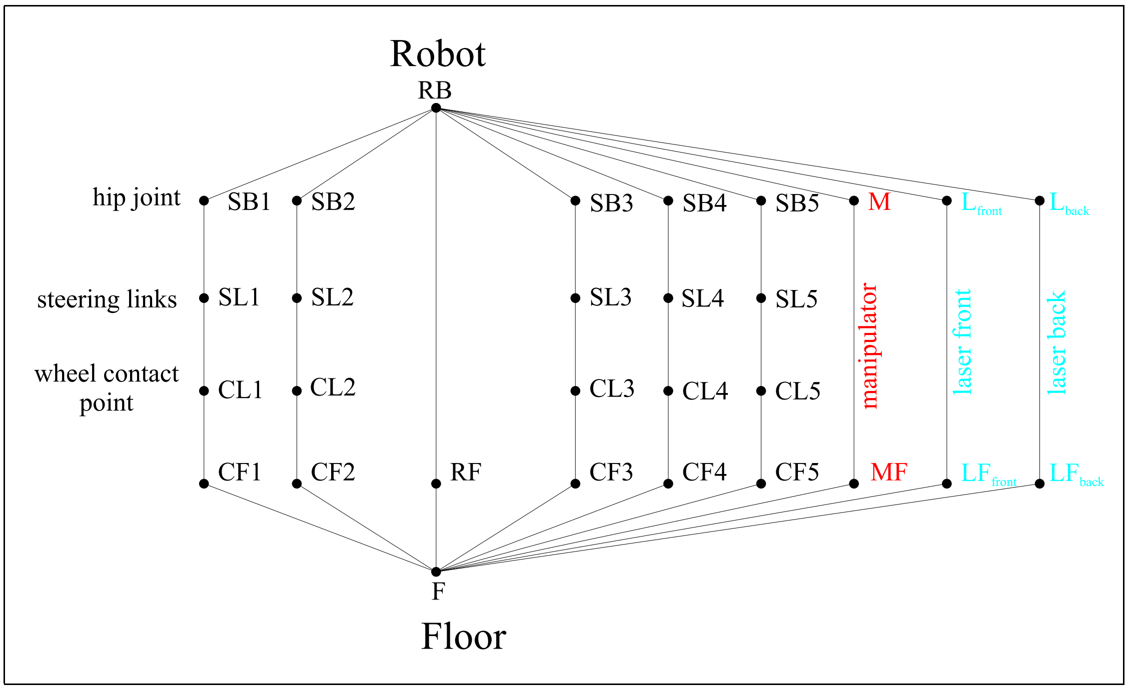


Figure 4.2: *GenBase II - transform graph*

Thus, a transformation matrix for a wheeled mobile robot is

$$^{R}T_{N} = \begin{bmatrix} \cos\phi & -\sin\phi & p_x \\ \sin\phi & \cos\phi & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

where $\phi = {}^{R}\phi_{N}$ is the rotation angle between two arbitrary frames R and N.

The complete transform graph for the mobile robot *GenBase II* is shown in **figure 4.2**. The following contexts can be derived from the robot's transform graph:

$$\begin{aligned}
^{RF}T_{RB} &= {}^{RB}T_{SB1}\ {}^{SB1}T_{SL1}\ {}^{SL1}T_{CL1}\ {}^{CL1}T_{CF1}\ {}^{CF1}T_{F}\ {}^{F}T_{RF} && (4.1)\\
&= {}^{RB}T_{SB2}\ {}^{SB2}T_{SL2}\ {}^{SL2}T_{CL2}\ {}^{CL2}T_{CF2}\ {}^{CF2}T_{F}\ {}^{F}T_{RF} && (4.2)\\
&= {}^{RB}T_{SB3}\ {}^{SB3}T_{SL3}\ {}^{SL3}T_{CL3}\ {}^{CL3}T_{CF3}\ {}^{CF3}T_{F}\ {}^{F}T_{RF} && (4.3)\\
&= {}^{RB}T_{SB4}\ {}^{SB4}T_{SL4}\ {}^{SL4}T_{CL4}\ {}^{CL4}T_{CF4}\ {}^{CF4}T_{F}\ {}^{F}T_{RF} && (4.4)\\
&= {}^{RB}T_{SB5}\ {}^{SB5}T_{SL5}\ {}^{SL5}T_{CL5}\ {}^{CL5}T_{CF5}\ {}^{CF5}T_{F}\ {}^{F}T_{RF} && (4.5)\\
&= {}^{RB}T_{L_{front}}\ {}^{L_{front}}T_{LF_{front}}\ {}^{LF_{front}}T_{F}\ {}^{F}T_{RF} && (4.6)\\
&= {}^{RB}T_{L_{back}}\ {}^{L_{back}}T_{LF_{back}}\ {}^{LF_{back}}T_{F}\ {}^{F}T_{RF} && (4.7)\\
&= {}^{RB}T_{M}\ {}^{M}T_{MF}\ {}^{MF}T_{F}\ {}^{F}T_{RF} && (4.8)
\end{aligned}$$

The transformation between any two coordinate frames is the multiplication of the intermediate transformations.

With the equations (4.1) to (4.8), the position of any point p on the robot, defined with respect to one frame N, can be found with respect to any other frame R (reference frame):

$$^{R}p = {}^{R}T_{N}\ {}^{N}p$$

The transformation is done by premultiplying the appropriate transformation matrix.

The robot has two drive wheels whose steering angles are fixed. This means that the drive wheels cannot swing around. Therefore, the coordinate systems SB1, SL1 and CL1 as well as SB2, SL2 and CL2 coincide. Thus, the coordinate transformation is simplified and becomes the identity matrix:

$$^{SB1}T_{SL1} \quad = \quad {}^{SL1}T_{CL1} \quad = \quad {}^{SB2}T_{SL2} \quad = \quad {}^{SL2}T_{CL2} \quad = \quad \mathbb{1}$$

The following matrices can be set up for the transformation between robot base coordinate system and the contact coordinate systems of the drive wheels:

$$^{RB}T_{CL1} = \begin{bmatrix} 1 & 0 & l_a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$^{RB}T_{CL2} = \begin{bmatrix} 1 & 0 & -l_a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The simplification used above cannot be applied to the three castors. Their matrices for the transformation can be subsumed as follows:

$$\begin{aligned} ^{RB}T_{CL3} &= {}^{RB}T_{SB3}\ {}^{SB3}T_{SL3}\ {}^{SL3}T_{CL3} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -l_b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -l_c \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & l_c\sin\theta_3 \\ \sin\theta_3 & \cos\theta_3 & -l_c\cos\theta_3 - l_b \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} ^{RB}T_{CL4} &= {}^{RB}T_{SB4}\ {}^{SB4}T_{SL4}\ {}^{SL4}T_{CL4} \\ &= \begin{bmatrix} 1 & 0 & l_f \\ 0 & 1 & l_d \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_4 & -\sin\theta_4 & 0 \\ \sin\theta_4 & \cos\theta_4 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -l_c \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta_4 & -\sin\theta_4 & l_c\sin\theta_4 + l_f \\ \sin\theta_4 & \cos\theta_4 & -l_c\cos\theta_4 + l_d \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} {}^{RB}T_{CL5} &= {}^{RB}T_{SB5}\ {}^{SB5}T_{SL5}\ {}^{SL5}T_{CL5} \\ &= \begin{bmatrix} 1 & 0 & -l_f \\ 0 & 1 & l_d \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_5 & -\sin\theta_5 & 0 \\ \sin\theta_5 & \cos\theta_5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -l_c \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta_5 & -\sin\theta_5 & l_c\sin\theta_5 - l_f \\ \sin\theta_5 & \cos\theta_5 & -l_c\cos\theta_5 + l_d \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

The transformation matrix between the robot's base coordinate system and the base coordinate system of the manipulator arm is a simple translatory transformation:

$${}^{RB}T_M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & l_m \\ 0 & 0 & 1 \end{bmatrix}$$

With the help of the above transformation matrix, the position p of a point in manipulator base coordinates can be calculated with respect to the world coordinate frame F. This is done using the following equation:

$${}^{F}p\ =\ {}^{M}T_F\ {}^{M}p \tag{4.9}$$

It has already been stated that any transformation between two frames is the multiplication of all intermediate transformations:

$${}^{M}T_F\ =\ {}^{M}T_{MF}\ {}^{MF}T_F\ =\ {}^{F}T_{RF}\ {}^{RF}T_{RB}\ {}^{RB}T_M \tag{4.10}$$

With the help of equation (4.10), equation (4.9) can be rewritten:

$${}^{F}p =^{F} T_{RF}\ {}^{RF}T_{RB}\ {}^{RB}T_M\ {}^{M}p$$

Now, the resulting equation to calculate the world coordinates of any point which is given in manipulator base coordinates is:

$${}^{F}p = \begin{bmatrix} 1 & 0 & x_{robot} \\ 0 & 1 & y_{robot} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\phi_{robot} & -\sin\phi_{robot} & 0 \\ \sin\phi_{robot} & \cos\phi_{robot} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & l_m \\ 0 & 0 & 1 \end{bmatrix} {}^{M}p \tag{4.11}$$

Let, for example, the position of the mobile robot in world coordinates be $x_{robot} = 2.0\ m$ and $y_{robot} = 3.0\ m$, the orientation of the robot is $\phi_{robot} = 45°$ and $l_m = 0.1675\ m$.[8] Then, the world coordinates of the manipulator arm's base are $x \approx 1.8816\ m$ and $y \approx 3.1184\ m$, which can be easily calculated using equation (4.11).

[8]The value of l_m is taken from **figure 2.6**.

The transformations between the mobile robot's base and the laser range finders' coordinate frames are simple translatory transformation matrices as well:

$$ {}^{RB}T_{L_{front}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & l_l \\ 0 & 0 & 1 \end{bmatrix} \tag{4.12} $$

$$ {}^{RB}T_{L_{back}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -l_l \\ 0 & 0 & 1 \end{bmatrix} \tag{4.13} $$

These transformations are used to represent the distance and angle measurements of the laser range finders in robot base coordinates.

If, for example, a reflector mark is seen in front of the robot at a distance $d_l = 2.0\ m$ and under an angle $\alpha_l = 60°$, the coordinates of the reflector mark can be calculated with respect to the laser range finder:

$$ \begin{pmatrix} x_l \\ y_l \end{pmatrix} = d_l \cdot \begin{pmatrix} \cos(\alpha_l) \\ \sin(\alpha_l) \end{pmatrix} $$

The coordinates can be transformed with the use of the homogenous transformation matrix in equation (4.12). Therefore, the coordinates x_r and y_r of the reflector mark with respect to the robot base's coordinate system are:

$$ \begin{pmatrix} x_r \\ y_r \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & l_l \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} d_l \cdot \cos(\alpha_l) \\ d_l \cdot \sin(\alpha_l) \\ 1 \end{pmatrix} $$

Now, the measurement can be expressed with respect to the robot base:

$$ d_r = \sqrt{x_r^2 + y_r^2} = \sqrt{(d_l \cdot \cos(\alpha_l))^2 + (d_l \cdot \sin(\alpha_l) + l_l)^2} \tag{4.14} $$

$$ \alpha_r = \arctan\left(\frac{y_r}{x_r}\right) = \arctan\left(\frac{d_l \cdot \sin(\alpha_l) + l_l}{d_l \cdot \cos(\alpha_l)}\right) \tag{4.15} $$

where d_r is the distance from the robot base to the reflector mark and α_r is the angle between the x axis of the robot base's coordinate system and the vector pointing at the reflector mark. With the equations (4.14) and (4.15), the example measurement is $d_r \approx 2.2971\,m$ and $\alpha \approx 64.194°$ with respect to the coordinate frame attached at the robot's centre point.[9] Later, these equations provide measurements for the extended Kalman filter.

The equations to transform the observations of the laser range finder at the back of the mobile robot can be derived using the homogenous transformation of equation (4.13). The only difference to the equations (4.14) and (4.15) is that l_t is negative. Therefore, the calculations are not additionally presented.

[9]The value of l_l can be received from **figure 2.6** and is $0.336\ m$ for the robot platform *GenBase II*.

4.1.2 Kinematic model

Although Muir and Neuman (1987) described the kinematics for robots with two degrees of freedom, their approach is not presented here.[10] Besides Muir and Neuman's way of using the previously introduced transformations and their Jacobian and Hessian matrices, an easier approach can be applied. This method, which is presented in this section, results in a number of equations which can easily be used with the extended Kalman filter later.

First and foremost, it has to be defined what is meant by the term *degrees of freedom*. It means the capability of a robot to translate into the direction of one of the coordinate system axis or to rotate around them. Altogether, there are six different degrees of freedom. A manipulator arm for example has six degrees of freedom if it can translate into any desired direction or rotate around any desired axis. If a robot has more than six degrees of freedom it is called redundant.[11]

Wheeled mobile robots can have different degrees of freedom. For example, robots equipped with *Mecanum wheels*[12] have omnidirectional mobility. They follow any xy path across a plain in an arbitrarily desired orientation. Hence they have three degrees of freedom, translation in the direction of x and y and rotation around the z axis.

Most robots have drive wheels which are diametrically opposed to each other. Because of this simple design, they have only one degree of translation and one degree of rotation. In contrast to a general-purpose robot, a robot with two degrees of freedom has singularities[13] in its workspace. This means that it must first turn in the direction in which it wants to go. Despite that limitation, the simple design consequently leads to simple kinematics.

Based on **figure 4.3**, the translatory velocity vector $\vec{v}$ of the robot base is separated into its components. From now on, the absolute value of the vector $\vec{v}$ is denoted by v.

$$\dot{x} = v \cdot \cos\varphi \tag{4.16}$$
$$\dot{y} = v \cdot \sin\varphi \tag{4.17}$$

The angle between the velocity vector and the x axis and likewise the orientation of the robot is called φ. Its change over time determines the robot's angular velocity ω

$$\dot{\varphi} = \omega = \frac{d\varphi}{dt} \tag{4.18}$$

During the motion of a robot on a plane, ω is needed to calculate the velocities v_R and v_L of the

[10](MUIR AND NEUMAN, 1987).

[11]For example, the *Mitsubishi PA-10* manipulator used on the *GenBase II* platform has seven joints and, therefore, is redundant with its seven degrees of freedom.

[12]The Mecanum AB, a Swedish company, developed and patented the *Mecanum wheel* under its inventor Bengt Ilon in 1973. This drive system has a unique steering system that provides simultaneous vehicle motion in all three directions: longitude (forward/backward), latitude (right/left) and yaw (rotatory).

[13]The meaning of the term *singularity* is described in (CRAIG, 1989) in detail.

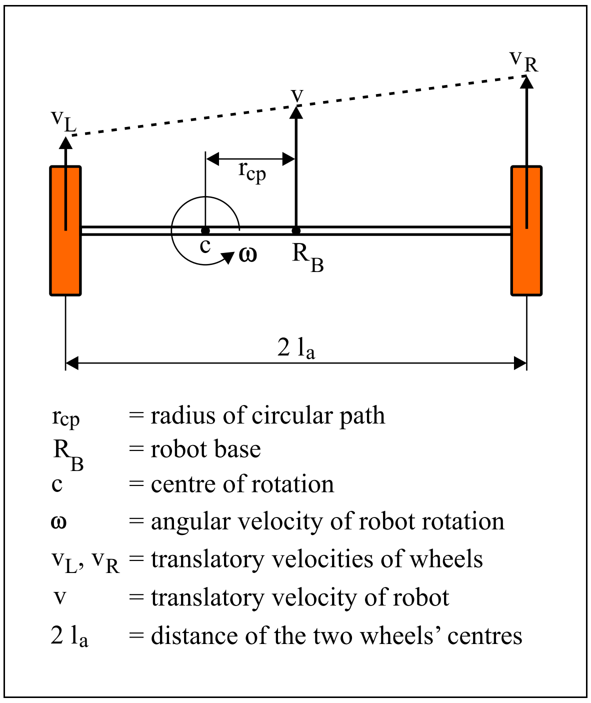

v_R
v
v_L
r_cp
c
ω
R_B
2 l_a
r_cp = radius of circular path
R_B = robot base
c = centre of rotation
ω = angular velocity of robot rotation
v_L, v_R = translatory velocities of wheels
v = translatory velocity of robot
2 l_a = distance of the two wheels' centres

Figure 4.3: *GenBase II - kinematic model*

drive wheels' contact points:

$$v_R = (l_a + r_{cp}) \cdot \omega \tag{4.19}$$

$$v_L = -(l_a - r_{cp}) \cdot \omega \tag{4.20}$$

The robot base is assumed to be coincident with the center of the wheel axis as shown in **figure 4.3**. The length of the segment between the axis' center and one wheel is l_a. The distance between the rotation point and the robot base is the radius of the rotation r_{cp}.

Most robot drive wheels include encoders for the measurement of the wheel speed. With the help of this information about the angular velocities ω_R and ω_L of the left and right drive wheel, the translational velocities v_R and v_L are calculated

$$v_R = \omega_1 \cdot r_R \tag{4.21}$$

$$v_L = \omega_2 \cdot r_L \tag{4.22}$$

where r_R and r_L are the radii of the wheels.

Based on this, w is calculated. A simple transformation of equation (4.19) and (4.20) provides

$$\begin{aligned} v_R - v_L &= 2l_a \cdot \omega \\ \omega &= \frac{v_R - v_L}{2l_a} \end{aligned} \tag{4.23}$$

The rotation radius r_{cp} is determined by another transformation of the same equations

$$\begin{aligned}
\frac{v_R}{l_a + r_{cp}} &= \frac{v_L}{l_a - r_{cp}} \\
v_R l_a - v_L r_{cp} &= -v_R l_a - v_L r_{cp} \\
(v_L - v_R)\, r_{cp} &= -l_a\,(v_R + v_L) \\
r_{cp} &= -l_a \frac{v_R + v_L}{v_L - v_R} \\
&= l_a \frac{v_R + v_L}{v_R - v_L} \qquad (4.24)
\end{aligned}$$

Now, equations (4.23) and (4.24) provide a routine to calculate the linear velocity v of the robot center R_B on a circular motion

$$\begin{aligned}
v &= \omega \cdot r_{cp} \\
&= \frac{v_R - v_L}{2l_a} \cdot l_a \frac{v_R + v_L}{v_R - v_L} \\
&= \frac{v_R + v_L}{2} \qquad (4.25)
\end{aligned}$$

Inserting the translatory velocity equation (4.25) and the wheel-floor contact point velocities of equations (4.21) and (4.22) into the equations (4.16)-(4.18) provides the following dependencies between the angular wheel velocities and robot base velocities:

$$\begin{aligned}
\dot{x} &= v \cdot \cos\varphi \\
&= \frac{v_R + v_L}{2} \cdot \cos\varphi \\
&= \frac{\omega_R r_R + \omega_L r_L}{2} \cdot \cos\varphi \qquad (4.26) \\
\dot{y} &= v \cdot \sin\varphi \\
&= \frac{v_R + v_L}{2} \cdot \sin\varphi \\
&= \frac{\omega_R r_R + \omega_L r_L}{2} \cdot \sin\varphi \qquad (4.27) \\
\dot{\varphi} &= \frac{v_R - v_L}{2l_a} \\
&= \frac{\omega_R r_R - \omega_L r_L}{2l_a} \qquad (4.28)
\end{aligned}$$

The equations from (4.26) to (4.28) describe the robot's kinematic model.

The inverse direction of the kinematics can be obtained by solving the equations (4.26) and (4.28) for ω_R and ω_L. Transforming equation (4.26) into

$$\begin{aligned}
\omega_R r_R \cos\varphi &= 2\dot{x} - \omega_L r_L \cos\varphi \\
\omega_R &= \frac{2\dot{x}}{r_R \cos\varphi} - \omega_L \frac{r_L}{r_R}
\end{aligned}$$

and equation (4.28) into

$$\begin{aligned} \omega_L r_L &= \omega_R r_R - 2\dot{\varphi} l_a \\ \omega_L &= \omega_R \frac{r_R}{r_L} - 2\frac{\dot{\varphi} l_a}{r_L} \end{aligned}$$

leads to an equation for the angular velocity ω_R of the first wheel

$$\begin{aligned} \omega_R &= \frac{2\dot{x}}{r_R \cos\varphi} - \left(\omega_R \frac{r_R}{r_L} - 2\frac{\dot{\varphi} l_a}{r_L}\right)\frac{r_L}{r_R} \\ &= \frac{2\dot{x}}{r_R \cos\varphi} - \omega_R + 2\frac{\dot{\varphi} l_a}{r_R} \\ \omega_R &= \frac{1}{r_R}\left(\frac{\dot{x}}{\cos\varphi} + \dot{\varphi} l_a\right) \end{aligned}$$

Correspondingly, an equation for the angular velocity ω_L of the second wheel can be derived by transforming equation (4.26) into

$$\begin{aligned} \omega_L r_L \cos\varphi &= 2\dot{x} - \omega_R r_R \cos\varphi \\ \omega_L &= \frac{2\dot{x}}{r_L \cos\varphi} - \omega_R \frac{r_R}{r_L} \end{aligned}$$

and equation (4.28) into

$$\begin{aligned} \omega_R r_R &= \omega_L r_L + 2\dot{\varphi} l_a \\ \omega_R &= \omega_L \frac{r_L}{r_R} + 2\frac{\dot{\varphi} l_a}{r_R} \end{aligned}$$

Combining the above equations results in

$$\begin{aligned} \omega_L &= \frac{2\dot{x}}{r_L \cos\varphi} - \left(\omega_2 \frac{r_L}{r_R} + 2\frac{\dot{\varphi} l_a}{r_R}\right)\frac{r_R}{r_L} \\ &= \frac{2\dot{x}}{r_L \cos\varphi} - \omega_L - 2\frac{\dot{\varphi} l_a}{r_L} \\ \omega_L &= \frac{1}{r_L}\left(\frac{\dot{x}}{\cos\varphi} - \dot{\varphi} l_a\right) \end{aligned}$$

Finally, the orientation of the robot has to be calculated. This can be done by using the translatory velocities $\dot{x}$ and $\dot{y}$

$$\varphi = \arctan\left(\frac{\dot{y}}{\dot{x}}\right) \tag{4.29}$$

4.2 Localisation using the Kalman filter and the kinematic model of the robot

In section 4.1.2, a kinematic model of a mobile robot with two degrees of freedom was developed. The model can be used with the *GenBase II* platform. In this section an extended Kalman filter which uses this model is discussed to obtain a localisation system for the mobile robot. In order to set up the Kalman filter system equation $f(\vec{x}_t, \vec{u}_t, \vec{w}_t)$, some assumptions about the hardware and control of the mobile robot's drive system must be made. With the help of these assumptions the system and measurement models for an extended Kalman filter are achieved. This extended Kalman filter is used for the localisation of the mobile robot.

4.2.1 Assumptions about the drive system

Since no detailed information is available from the manufacturer, a simple system, like the one presented in **figure 4.4** is proposed for the drive system of the mobile robot *GenBase II*. It is

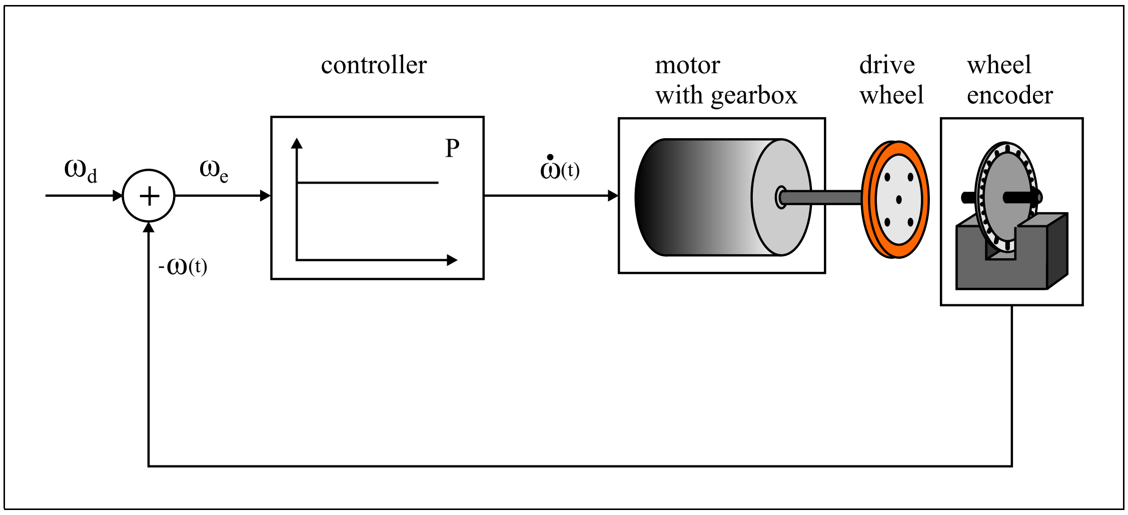


Figure 4.4: *GenBase II - simple model of the drive wheel's velocity controller*

assumed that the motor is controlled by a simple P-controller and the controlled system has a proportional transmission characteristic. The rotation of a drive wheel is measured by a wheel encoder and is fed back to the P-controller. This control circuit leads to the following mathematical description

$$\boxed{\begin{aligned}\frac{\partial\omega(t)}{\partial t} &= \lambda(\omega_d - \omega(t)) \\ &= -\lambda\cdot\omega(t) + \lambda\cdot\omega_d\end{aligned}}$$

where ω_d is the desired angular velocity of a drive wheel.

The above equation is a linear first-order differential equation. This equation can be solved in three steps. Firstly, the associated homogeneous differential equation or characteristic equation is solved. Secondly, the non-homogeneous case is calculated to receive an expression for the constant C (see below). This can be achieved by using the method of variation of the constant. Finally, the solutions for the homogeneous and the non-homogeneous cases are put together to obtain a general solution.

The associated homogeneous differential equation looks as follows:

$$\dot{\omega}_h(t) + \lambda\cdot\omega(t) = 0$$

The general solution for the homogeneous differential equation from above is calculated in a first step:

$$\begin{aligned}\omega_h(t) &= C'\cdot e^{-\int\lambda\,dt} = C'\cdot e^{-\lambda t+c} = C'\cdot e^{-\lambda t}\cdot e^{c} \\ &= C\cdot e^{-\lambda t}\end{aligned} \tag{4.30}$$

In equation (4.30), the constant C and the variable λ are not known. They will be identified later. In the second step, the solution for the inhomogeneous differential equation can be found by variation of the constant. The variation of the constant is shown in the equations (4.31) and (4.32). The constant C is considered to be a function of time for this method.

$$\omega(t) = C(t)\cdot w_h(t) \tag{4.31}$$

$$\dot{\omega}(t) = C(t)\cdot\dot{\omega}_h(t) + \dot{C}(t)\cdot\omega_h(t) \tag{4.32}$$

The insertion of equations (4.31) and (4.32) into the inhomogeneous differential equation leads to

$$C(t)\cdot\dot{\omega}_h(t) + \dot{C}(t)\cdot\omega_h(t) = -\lambda\cdot C(t)\cdot\omega_h(t) + \lambda\cdot\omega_d \tag{4.33}$$

Equation (4.33) can be simplified:

$$\begin{aligned}C(t)\cdot\dot{\omega}_h(t) + \lambda\cdot C(t)\cdot\omega_h(t) + \dot{C}(t)\cdot\omega_h(t) &= \lambda\cdot\omega_d \\ C(t)\cdot\underbrace{\left(\dot{\omega}_h(t) + \lambda\cdot\omega_h(t)\right)}_{=0\text{, see homogeneous differential equation}} + \dot{C}(t)\cdot\omega_h(t) &= \lambda\cdot\omega_d\end{aligned}$$

$$\dot{C}(t) = \omega_h^{-1}(t)\cdot\lambda\cdot\omega_d \tag{4.34}$$

Because of the simplification which is achieved in (4.34), an expression for $C(t)$ can be found

$$C(t) = \int_0^t \omega_h^{-1}(t') \cdot \lambda \cdot \omega_d \, dt' \tag{4.35}$$

If the expression for $C(t)$ in equation (4.35) is inserted into equation (4.31) this results in an expression for $\omega(t)$:

$$\begin{aligned} \omega(t) &= \int_0^t \omega_h^{-1}(t') \cdot \lambda \cdot \omega_d \, dt' \cdot \omega_h(t) \\ &= \int_0^t C^{-1} \cdot e^{\lambda t'} \cdot \lambda \cdot \omega_d \, dt' \cdot C \cdot e^{-\lambda t} \\ &= \int_0^t e^{-\lambda(t-t')} \, dt' \cdot \lambda \cdot \omega_d \\ &= \omega_{\text{special}}(t) \end{aligned}$$

Now, the general solution can be calculated:

$$\begin{aligned} \omega(t) &= \omega_{\text{special}}(t) + \omega_h(t) \\ &= \int_0^t e^{-\lambda(t-t')} \, dt' \cdot \lambda \cdot \omega_d + C \cdot e^{-\lambda t} \\ &= \left[\frac{1}{\lambda} \cdot e^{-\lambda(t-t')} \right]_0^t \lambda \cdot \omega_d + C \cdot e^{-\lambda t} \\ &= \frac{1}{\lambda} \left[e^{-\lambda 0} - e^{-\lambda t} \right] \lambda \cdot \omega_d + C \cdot e^{-\lambda t} \\ &= \omega_d - \omega_d \cdot e^{-\lambda t} + C \cdot e^{-\lambda t} \end{aligned} \tag{4.36}$$

The expansion of equation (4.36) with $-C + C$ and reorganization of the result leads to

$$\omega(t) = C + (\omega_d - C)\left(1 - e^{-\lambda t}\right) \tag{4.37}$$

The value C of equation (4.37) can be determined by setting the time t to zero. It can be seen that C becomes $\omega(0)$. This results in:

$$\omega(t) = \omega(0) + (\omega_d - \omega(0))\left(1 - e^{-\lambda t}\right) \tag{4.38}$$

The following assumption can be made: If equation (4.38) describes the behaviour of the system which evolves from a time step t to a time step $t+1$ and the time which has passed within that step is denoted by Δt then the equation can be rewritten:

$$\boxed{\omega_{t+1} = \omega_t + (\omega_d - \omega_t)\left(1 - e^{\frac{-\Delta t}{\tau}}\right)} \tag{4.39}$$

with $\tau = \frac{1}{\lambda}$.

Equation (4.39) is assumed describing the development of the angular velocity of the mobile robot *GenBase II*'s drive system.

4.2.2 System model for the extended Kalman filter

With the assumptions about the drive system made in section 4.2.1 a non-linear system model for an extended Kalman filter can be developed. The description of a drive wheel's angular velocity in equation (4.39) has to be specified for each drive wheel. Since the system which is used in a Kalman filter is assumed to be disturbed by Gaussian noise, a random variable w_n with zero-mean is added to each of the following system equations. Therefore, the drive wheel's angular velocities are formulated as follows:

$$\begin{aligned}
\omega_{R,t+1} &= \omega_{R,t} + (u_{R,t} - \omega_{R,t})\left(1 - e^{\frac{-\Delta t}{\tau}}\right) + \mathrm{w}_1 \\
&= \omega_{R,t} + u_{R,t} - u_{R,t} \cdot e^{\frac{-\Delta t}{\tau}} - \omega_{R,t} + \omega_{R,t} \cdot e^{\frac{-\Delta t}{\tau}} + \mathrm{w}_1 \\
&= u_{R,t} + (\omega_{R,t} - u_{R,t})e^{\frac{-\Delta t}{\tau}} + \mathrm{w}_1
\end{aligned} \tag{4.40}$$

$$\omega_{L,t+1} = u_{L,t} + (\omega_{L,t} - u_{L,t})e^{\frac{-\Delta t}{\tau}} + \mathrm{w}_2 \tag{4.41}$$

It is possible to describe the translatory and angular velocities of *GenBase II*'s robot centre in terms of the wheel velocities. This is known from the equations (4.26), (4.27) and (4.28) which were derived in section 4.1.2. If the equations (4.40) and (4.41) are substituted in the equations (4.26) and (4.27), the translatory velocity of the robot centre can be rewritten:

$$\begin{aligned}
\dot{x}_{t+1} &= \frac{\omega_{R,t+1} \cdot r_R + \omega_{L,t+1} \cdot r_L}{2} \cdot \cos\varphi_{t+1} + \mathrm{w}_3 \\
&= \left[\frac{(u_{R,t} + (\omega_{R,t} - u_{R,t})e^{\frac{-\Delta t}{\tau}} + \mathrm{w}_1) \cdot r_R + (u_{L,t} + (\omega_{L,t} - u_{L,t})e^{\frac{-\Delta t}{\tau}} + \mathrm{w}_2) \cdot r_L}{2}\right] \cdot \\
&\quad \cos\left(\varphi_t + \dot{\varphi}_t \Delta t + \frac{\ddot{\varphi}_t}{2}(\Delta t)^2\right) + \mathrm{w}_3 \\
&= \frac{1}{2}\left[u_{R,t} \cdot r_R + r_R(\omega_{R,t} - u_{R,t})e^{\frac{-\Delta t}{\tau}} + \mathrm{w}_1 \cdot r_R + u_{L,t} \cdot r_L + r_L(\omega_{L,t} - u_{L,t})e^{\frac{-\Delta t}{\tau}} + \mathrm{w}_2 \cdot r_L\right] \cdot \\
&\quad \cos\left(\varphi_t + \dot{\varphi}_t \Delta t + \frac{\ddot{\varphi}_t}{2}(\Delta t)^2\right) + \mathrm{w}_3 \qquad | r = r_R = r_L \\
&= \frac{r}{2}\left[u_{R,t} + u_{L,t} + (\omega_{R,t} + \omega_{L,t} - u_{R,t} - u_{L,t})e^{\frac{-\Delta t}{\tau}}\right] \cos\left(\varphi_t + \dot{\varphi}_t \Delta t + \frac{\ddot{\varphi}_t}{2}(\Delta t)^2\right) \\
&\quad + \frac{r}{2}\left[\mathrm{w}_1 + \mathrm{w}_2\right] \cos\left(\varphi_t + \dot{\varphi}_t \Delta t + \frac{\ddot{\varphi}_t}{2}(\Delta t)^2\right) + \mathrm{w}_3 \\
&= \frac{r}{2}\left[(u_{R,t} + u_{L,t})\left(1 - e^{\frac{-\Delta t}{\tau}}\right) + (\omega_{R,t} + \omega_{L,t})\, e^{\frac{-\Delta t}{\tau}}\right] \cos\left(\varphi_t + \dot{\varphi}_t \Delta t + \frac{\ddot{\varphi}_t}{2}(\Delta t)^2\right) \\
&\quad + \frac{r}{2}\left[\mathrm{w}_1 + \mathrm{w}_2\right] \cos\left(\varphi_t + \dot{\varphi}_t \Delta t + \frac{\ddot{\varphi}_t}{2}(\Delta t)^2\right) + \mathrm{w}_3
\end{aligned} \tag{4.42}$$

$$\begin{aligned}
\dot{y}_{t+1} &= \frac{\omega_{R,t+1}\cdot r_R + \omega_{L,t+1}\cdot r_L}{2}\sin\varphi_{t+1} + \mathrm{w}_4 \\
&= \frac{r}{2}\left[(u_{R,t}+u_{L,t})\left(1-e^{\frac{-\Delta t}{\tau}}\right) + (\omega_{R,t}+\omega_{L,t})\,e^{\frac{-\Delta t}{\tau}}\right]\sin\left(\varphi_t + \dot{\varphi}_t\Delta t + \frac{\ddot{\varphi}_t}{2}(\Delta t)^2\right) \\
&\quad +\frac{r}{2}\left[\mathrm{w}_1+\mathrm{w}_2\right]\sin\left(\varphi_t + \dot{\varphi}_t\Delta t + \frac{\ddot{\varphi}_t}{2}(\Delta t)^2\right) + \mathrm{w}_4 \qquad (4.43)
\end{aligned}$$

For the angular velocity of the robot's centre, equations (4.40) and (4.41) are inserted into equation (4.28):

$$\begin{aligned}
\dot{\varphi}_{t+1} &= \frac{\omega_{R,t+1}\cdot r_R - \omega_{L,t+1}\cdot r_L}{2l_a} + \mathrm{w}_5 \\
&= \frac{\left(u_{R,t} + (\omega_{R,t}-u_{R,t})e^{\frac{-\Delta t}{\tau}} + \mathrm{w}_1\right) r_R - \left(u_{L,t} + (\omega_{L,t}-u_{L,t})e^{\frac{-\Delta t}{\tau}} + \mathrm{w}_2\right) r_L}{2l_a} + \mathrm{w}_5 \\
&= \frac{1}{2l_a}\Big[u_{R,t}\cdot r_R + r_R(\omega_{R,t}-u_{R,t})e^{\frac{-\Delta t}{\tau}} + \mathrm{w}_1\cdot r_R \\
&\quad -u_{L,t}\cdot r_L - r_L(\omega_{L,t}-u_{L,t})e^{\frac{-\Delta t}{\tau}} - \mathrm{w}_2\cdot r_L\Big] + \mathrm{w}_5 \qquad |r = r_R = r_L \\
&= \frac{r}{2l_a}\left[u_{R,t} - u_{L,t} + (\omega_{R,t} - u_{R,t} - \omega_{L,t} + u_{L,t})e^{\frac{-\Delta t}{\tau}} + (\mathrm{w}_1-\mathrm{w}_2)\right] + \mathrm{w}_5 \\
&= \frac{r}{2l_a}\left[(u_{R,t}-u_{L,t})\left(1-e^{\frac{-\Delta t}{\tau}}\right) + (\omega_{R,t}-\omega_{L,t})e^{\frac{-\Delta t}{\tau}} + (\mathrm{w}_1-\mathrm{w}_2)\right] + \mathrm{w}_5 \qquad (4.44)
\end{aligned}$$

The translatory and angular accelerations of the *GenBase II* robot's centre point can be calculated as the difference between the velocities at time $t+1$ and t divided by Δt. With the above equations for the velocities (4.42) to (4.44), the following equations for the acceleration can be established:

$$\begin{aligned}
\ddot{x}_{t+1} &= \frac{\dot{x}_{t+1}-\dot{x}_t}{\Delta t} + \mathrm{w}_6 \\
&= \frac{r}{2\Delta t}\left[(u_{R,t}+u_{L,t})\left(1-e^{\frac{-\Delta t}{\tau}}\right) + (\omega_{R,t}+\omega_{L,t})\,e^{\frac{-\Delta t}{\tau}}\right]\cos\left(\varphi_t + \dot{\varphi}_t\Delta t + \frac{\ddot{\varphi}_t}{2}(\Delta t)^2\right) \\
&\quad +\frac{r}{2\Delta t}\left[\mathrm{w}_1+\mathrm{w}_2\right]\cos\left(\varphi_t + \dot{\varphi}_t\Delta t + \frac{\ddot{\varphi}_t}{2}(\Delta t)^2\right) + \mathrm{w}_3 - \frac{\dot{x}_t}{\Delta t} + \mathrm{w}_6 \qquad (4.45)
\end{aligned}$$

$$\begin{aligned}
\ddot{y}_{t+1} &= \frac{\dot{y}_{t+1}-\dot{y}_t}{\Delta t} + \mathrm{w}_7 \\
&= \frac{r}{2\Delta t}\left[(u_{R,t}+u_{L,t})\left(1-e^{\frac{-\Delta t}{\tau}}\right) + (\omega_{R,t}+\omega_{L,t})\,e^{\frac{-\Delta t}{\tau}}\right]\sin\left(\varphi_t + \dot{\varphi}_t\Delta t + \frac{\ddot{\varphi}_t}{2}(\Delta t)^2\right) \\
&\quad +\frac{r}{2\Delta t}\left[\mathrm{w}_1+\mathrm{w}_2\right]\sin\left(\varphi_t + \dot{\varphi}_t\Delta t + \frac{\ddot{\varphi}_t}{2}(\Delta t)^2\right) + \mathrm{w}_4 - \frac{\dot{y}_t}{\Delta t} + \mathrm{w}_7 \qquad (4.46)
\end{aligned}$$

$$\begin{aligned}\ddot{\varphi}_{t+1} &= \frac{\dot{\varphi}_{t+1} - \dot{\varphi}_t}{\Delta t} + \mathrm{w}_8 \\ &= \frac{r}{2l_a \Delta t}\left[(u_{R,t} - u_{L,t})\left(1 - e^{\frac{-\Delta t}{\tau}}\right) + (\omega_{R,t} - \omega_{L,t})e^{\frac{-\Delta t}{\tau}} + (\mathrm{w}_1 - \mathrm{w}_2)\right] + \mathrm{w}_5 \\ &\quad - \frac{\dot{\varphi}_t}{\Delta t} + \mathrm{w}_8 \end{aligned} \tag{4.47}$$

Now, the equations of motion for the mobile platform can be formulated. The position and orientation at time $t+1$ can be calculated from the positions, velocities and accelerations at time t. The respective equations for the x and y position and the orientation are:

$$x_{t+1} = x_t + \dot{x}_t \Delta t + \frac{\ddot{x}_t}{2}(\Delta t)^2 + \mathrm{w}_9 \tag{4.48}$$

$$y_{t+1} = y_t + \dot{y}_t \Delta t + \frac{\ddot{y}_t}{2}(\Delta t)^2 + \mathrm{w}_{10} \tag{4.49}$$

$$\varphi_{t+1} = \varphi_t + \dot{\varphi}_t \Delta t + \frac{\ddot{\varphi}_t}{2}(\Delta t)^2 + \mathrm{w}_{11} \tag{4.50}$$

Finally, the positions of the features which are laser reflector marks in this work have to be addressed. This is rather simple because the features are expected to be fixed. This means, the position of feature i at time $t+1$ is the same as at time t:

$$x_{f_i,t+1} = x_{f_i,t+1} + \mathrm{w}_{12} \tag{4.51}$$

$$y_{f_i,t+1} = y_{f_i,t+1} + \mathrm{w}_{13} \tag{4.52}$$

With the equations from (4.40) to (4.52), a system is specified which can be used with the extended Kalman filter. The system-state vector used for the mobile robot *GenBase II* is:

$$\vec{x}_t = \left(\omega_R, \omega_L, \dot{x}, \dot{y}, \dot{\phi}, \ddot{x}, \ddot{y}, \ddot{\phi}, x, y, \phi, x_{f_1}, y_{f_1}, \ldots, x_{f_i}, y_{f_i}\right)_t^T$$

The vector function f is given by the non-linear equations from (4.40) to (4.52) and transforms the system-state from time step t to $t+1$:

$$\vec{x}_{t+1} = f\left(\vec{x}_t, \vec{u}_t, \vec{\mathrm{w}}_t\right)$$

4.2.3 Measurement model for the extended Kalman filter

Still, the measurement model which is needed by the extended Kalman filter has to be put up. It describes how the measurements can be formulated in terms of the system-state. Each measurement is disturbed by Gaussian noise, so therefore, a random variable v_n with zero mean is added to each equation. Because of the types of sensors which are available on the *GenBase II* platform, five different measurements can be made. The first two are the measurements of the drive wheel velocities. These are direct measurements of system-states and lead to two simple equations:

$$\omega_{R,t}^{\text{measured}} = \omega_{R,t} + \mathrm{v}_1 \tag{4.53}$$

$$\omega_{L,t}^{\text{measured}} = \omega_{L,t} + \mathrm{v}_2 \tag{4.54}$$

The gyrocompass measures the angular velocity of the mobile platform's centre. This next measurement is also a direct measurement of one of the system-state's components. The resulting equation for the measurement is again simple:

$$\dot{\varphi}_t^{\text{measured}} = \dot{\varphi}_t + \mathrm{v}_5 \tag{4.55}$$

The last two measurements are made for each feature f_i. One measurement is the distance $d_{f_i,t}$ from the robot's centre point to the feature at time t. The other measurement is the angle $\alpha_{f_i,t}$ between the robot's orientation vector and the vector from the robot's centre to the feature. Both measurements can be calculated from the system-state as follows:

$$d_{f_i,t} = \sqrt{(x_t - x_{f_i,t})^2 + (y_t - y_{f_i,t})^2} + \mathrm{v}_3 \tag{4.56}$$

$$\alpha_{f_i,t} = \operatorname{arctan} 2 \left(\frac{y_{f_i,t} - y_t}{x_{f_i,t} - x_t} \right) - \varphi_t + \mathrm{v}_4 \tag{4.57}$$

The equations from (4.53) to (4.57) specify the non-linear measurement model h which describes the measurements $\vec{z}_t$ in terms of the disturbed system-state at time t:

$$\vec{z}_t = h\left(\vec{x}_t, \vec{\mathrm{v}}_t\right)$$

The measurement vector $\vec{z}_t$ for the *GenBase II* platform looks as follows:

$$\vec{z}_t = \left(\omega_{R,t}^{\text{measured}}, \omega_{L,t}^{\text{measured}}, \varphi_t^{\text{measured}}, d_{f_1,t}, \alpha_{f_1,t}, \ldots, d_{f_i,t}, \alpha_{f_i,t}\right)^T$$

At the end, the Jacobians of the system and measurement model have to be calculated. They are presented in **figures 4.5**, **4.7**, **4.6** and **4.8**.

With this framework, the extended Kalman filter can be used for the localisation of the mobile robot platform *GenBase II*. The filter algorithm must be started with an initial guess of the system-state. Then the position and orientation at time step t can be read off the state vector $\vec{x}_t$.

How the information of the extended Kalman filter is used to control the robot's movements is presented in the next section.

$$F_t^{\mathrm{J.}} = \left[\begin{array}{ccccccccccc|cc}
e^{\frac{-\Delta t}{\tau}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & e^{\frac{-\Delta t}{\tau}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2}e^{\frac{-\Delta t}{\tau}}\cos(\cdot) & \frac{r}{2}e^{\frac{-\Delta t}{\tau}}\cos(\cdot) & 0 & 0 & (-\dot{y}_{t+1}+\mathrm{w}_4)\,\Delta t & 0 & 0 & (-\dot{y}_{t+1}+\mathrm{w}_4)\,\frac{\Delta t^2}{2} & 0 & 0 & (-\dot{y}_{t+1}+\mathrm{w}_4) & 0 & 0 \\
\frac{r}{2}e^{\frac{-\Delta t}{\tau}}\sin(\cdot) & \frac{r}{2}e^{\frac{-\Delta t}{\tau}}\sin(\cdot) & 0 & 0 & (\dot{x}_{t+1}-\mathrm{w}_3)\,\Delta t & 0 & 0 & (\dot{x}_{t+1}-\mathrm{w}_3)\,\frac{\Delta t^2}{2} & 0 & 0 & (\dot{x}_{t+1}-\mathrm{w}_3) & 0 & 0 \\
\frac{r}{2l_a}e^{\frac{-\Delta t}{\tau}} & -\frac{r}{2l_a}e^{\frac{-\Delta t}{\tau}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2\Delta t}e^{\frac{-\Delta t}{\tau}}\cos(\cdot) & \frac{r}{2\Delta t}e^{\frac{-\Delta t}{\tau}}\cos(\cdot) & \frac{-1}{\Delta t} & 0 & -\dot{y}_{t+1}+\mathrm{w}_4 & 0 & 0 & (-\dot{y}_{t+1}+\mathrm{w}_4)\,\frac{\Delta t}{2} & 0 & 0 & \frac{1}{\Delta t}(-\dot{y}_{t+1}+\mathrm{w}_4) & 0 & 0 \\
\frac{r}{2\Delta t}e^{\frac{-\Delta t}{\tau}}\sin(\cdot) & \frac{r}{2\Delta t}e^{\frac{-\Delta t}{\tau}}\sin(\cdot) & 0 & \frac{-1}{\Delta t} & \dot{x}_{t+1}-\mathrm{w}_3 & 0 & 0 & (\dot{x}_{t+1}-\mathrm{w}_3)\,\frac{\Delta t}{2} & 0 & 0 & \frac{1}{\Delta t}(\dot{x}_{t+1}-\mathrm{w}_3) & 0 & 0 \\
\frac{r}{2l_a\Delta t}e^{\frac{-\Delta t}{\tau}} & -\frac{r}{2l_a\Delta t}e^{\frac{-\Delta t}{\tau}} & 0 & 0 & \frac{-1}{\Delta t} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \Delta t & 0 & 0 & \frac{\Delta t^2}{2} & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \Delta t & 0 & 0 & \frac{\Delta t^2}{2} & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \Delta t & 0 & 0 & \frac{\Delta t^2}{2} & 0 & 0 & 1 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]$$

Figure 4.5: *The complete $F_t^{Jacobian}$ for the GenBase II*

$$H_t^{\mathrm{J.}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{x_t - x_{f_i,t}}{\sqrt{\left(x_t - x_{f_i,t}\right)^2 + \left(y_t - y_{f_i,t}\right)^2}} & \frac{y_t - y_{f_i,t}}{\sqrt{\left(x_t - x_{f_i,t}\right)^2 + \left(y_t - y_{f_i,t}\right)^2}} & 0 & \frac{x_{f_i,t} - x_t}{\sqrt{\left(x_t - x_{f_i,t}\right)^2 + \left(y_t - y_{f_i,t}\right)^2}} & \frac{y_{f_i,t} - y_t}{\sqrt{\left(x_t - x_{f_i,t}\right)^2 + \left(y_t - y_{f_i,t}\right)^2}} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{y_{f_i,t} - y_t}{\left(x_{f_i,t} - x_t\right)^2 + \left(y_{f_i,t} - y_t\right)^2} & \frac{x_t - x_{f_i,t}}{\left(x_{f_i,t} - x_t\right)^2 + \left(y_{f_i,t} - y_t\right)^2} & -1 & \frac{y_t - y_{f_i,t}}{\left(x_{f_i,t} - x_t\right)^2 + \left(y_{f_i,t} - y_t\right)^2} & \frac{x_{f_i,t} - x_t}{\left(x_{f_i,t} - x_t\right)^2 + \left(y_{f_i,t} - y_t\right)^2} \end{bmatrix}$$

Figure 4.6: *The complete H_t^{Jacobian} for the GenBase II*

$$W_t^{\text{Jacobian}} = \left[\begin{array}{ccccccccccc|cc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2}\cos(\cdot) & \frac{r}{2}\cos(\cdot) & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2}\sin(\cdot) & \frac{r}{2}\sin(\cdot) & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2l_a} & \frac{-r}{2l_a} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2\Delta t}\cos(\cdot) & \frac{r}{2\Delta t}\cos(\cdot) & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2\Delta t}\sin(\cdot) & \frac{r}{2\Delta t}\sin(\cdot) & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{r}{2l_a\Delta t} & \frac{-r}{2l_a\Delta t} & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right]$$

Figure 4.7: *The complete $W_t^{Jacobian}$ for the GenBase II*

$$V_t^{\text{Jacobian}} = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

Figure 4.8: *The complete $V_t^{Jacobian}$ for the GenBase II*

4.3 Software implementation

During this work, a *C++* software library for the self-localisation has been developed. It implements an extended Kalman filter with the system and measurement model described in the previous section. The library includes six classes. The hierarchy of these classes can be seen in **figure 4.9**. The inheritances are indicated by green arrows. The relations between the different classes are marked by red lines. The digits at the end of the red lines represent the number of instances of the respective classes. In **figure 4.9** only some important attributes and the public methods of the classes are shown.

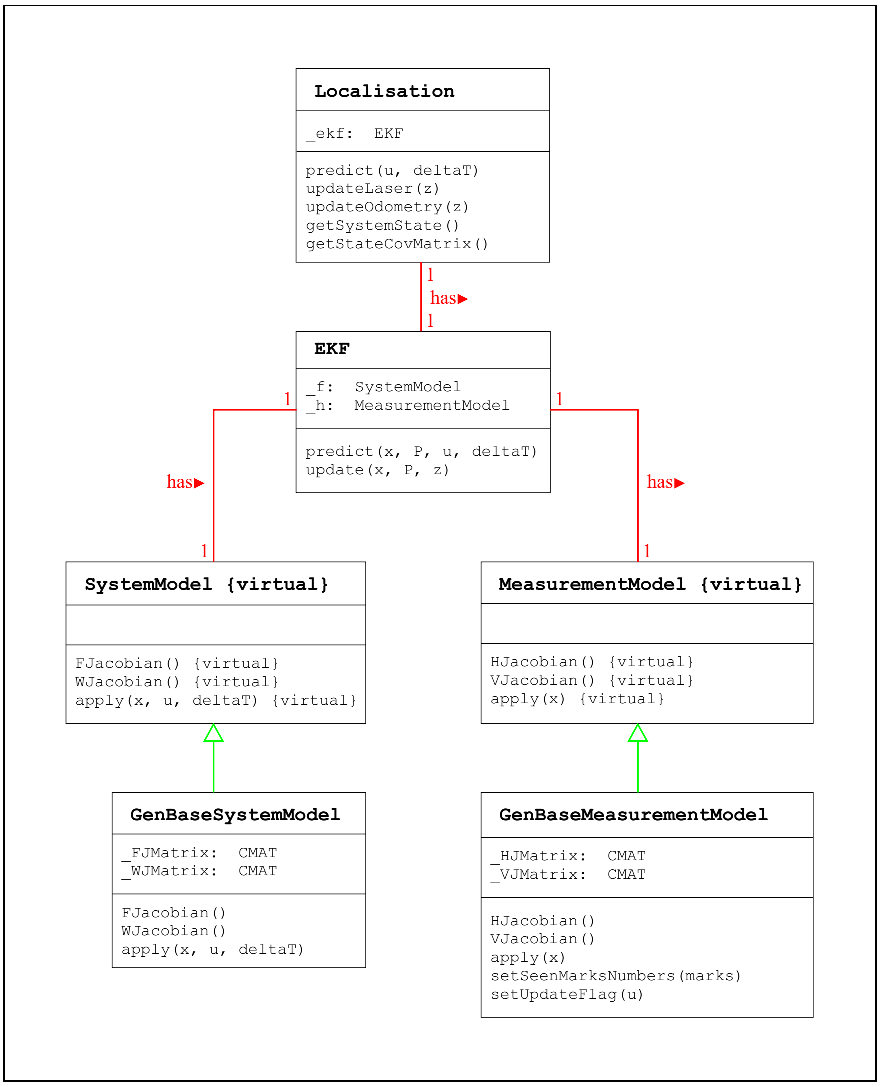


Figure 4.9: *The figure shows the flowchart with the hierarchy of the self-localisation's classes. The inheritances are indicated by green arrows. The relations are indicated by red lines, the digits at their ends represent the number of instances. Only some important attributes and the public methods are stated.*

The main class is *Localisation*. By creating an instance of this class, the self-localisation is ready to start. The constructor of the class takes the initial estimate of the system-state vector and its covariance matrix as arguments. The class *Localisation* has one instance of the extended Kalman filter class *EKF* as a member variable. The predict and update methods of *Localisation* are basically wrappers of the methods *predict(...)* and *update(...)* of the class *EKF*.

The constructor of the generic class *EKF* takes one instance of each of the virtual classes *SystemModel* and *MeasurementModel*. Derived from these are the classes *GenBaseSystemModel* and *GenBaseMeasurementModel* which are the implementation of the system and measurement models set up earlier in this chapter.

Since the measurements of the odometry and the laser range finders are not synchronized, two update methods *updateOdometry(...)* and *updateLaser(...)* are provided by *Localisation*. Before these methods call *update(...)* of the extended Kalman filter object, they change the measurement model. This is done by calling *setSeenMarksNumbers(...)* and *setUpdateFlag(...)*. The class *GenBaseMeasurementModel* is thereby adapted to the measurement which is to be passed to the extended Kalman filter. The sizes of the internal matrices and vectors are fitted to the appropriate sizes. Furthermore, some flags and the numbers of the seen marks are stored. This is necessary to deal with the changing number of reflector marks which the laser range finders detect.

```
#
# Parameter file for GenBase II
#

WheelRadius                = 77    # mm
DistanceDriveWheels        = 600   # mm
MaxDistanceMarkMeasurement = 200   # mm

# OdometryVariance
OdometryVariance           = 0.01

# GyroVariance
GyroVariance               = 0.1

# SystemVariance
SystemVariance             = 1.0

# Tau
Tau                        = 0.08

# LaserDistanceVariance
LaserDistanceVariance      = 4900.0

# LaserAngleVariance
LaserAngleVariance         = 0.01

# LaserMarkVariance
LaserMarkVariance          = 0.01
```

Figure 4.10: *The figure shows the parameter file for the extended Kalman filter which is used by the self-localisation library.*

The calls of *predict(...)*, *updateOdometry(...)* and *updateLaser(...)* of *Localisation* control the operating sequence of the self-localisation. Thus, this class can be seen as a façade of the library.

The parameters of the extended Kalman filter are stored in a file. It includes the dimensions of the mobile robot and the variances of the system-state and measurement components. As an example, **figure 4.10** presents the parameter file used for the experiments of chapter 7. The results of these experiments demonstrate that the parameters are particularly suitable for the *GenBase II* platform.

The positions of the reflector marks are stored in another file. Each line of the file contains the x and y coordinates of a specific reflector mark. An example file is shown in **figure 4.11**. It contains the coordinates of the reflector marks which are installed in the cell culture laboratory.

```
#
# Positions of the reflector marks
#
# units: [mm]
#
  +15.0     +0.0 # mark at origin
+1045.0   +680.0 # ultrafiltration right
+2290.0   +680.0 # ultrafiltration left
+3555.0  +1000.0
+2525.0  +3650.0
+2870.0  +5245.0
+2760.0  +5800.0
+2770.0  +7295.0
+2755.0  +9475.0
-1420.0 +10610.0
  -80.0  +6885.0
-1425.0  +5780.0
-1040.0  +4175.0

# door to  e02-220, left side
+1745.0 +10625.0

# blue control cubicle
-1740.0 +9895.0

# under cedex
-2265.0 +8625.0
```

Figure 4.11: *This figure shows the file with the positions of the reflector marks.*

In both files, empty lines and those lines beginning with the character '#' are ignored. Thus, it is possible to add comments to the files. The files can be easily changed using a text editor.

Chapter 5

Control of a mobile robot with 2 DOFs

In this chapter, the control of the movements of the robot is explained. The motions of the robot are divided into translational and rotatory movements (see sections 5.3 and 5.4). For the rotation and translation controller, two different methods are brought together. The first one is the PI-controller. In this work it is used in combination with the second method called one-dimensional filters which is explained in section 5.2.

The rotation controller uses a one-dimensional filter to adjust the desired orientation of the mobile robot. In this case, a PI-controller does the fine-tuning of the orientation at the end of the whole motion. The translation controller again uses a one-dimensional filter to generate the translational velocity values for the robot and simultaneously a PI-controller to correct deviations from the desired straight path from the start point to the goal point.

5.1 PI-controllers

PI-controllers are used for a lot of technical control problems like controlling the rotational speed of electrical machines, especially for d.c. motors. In this section, the basics of PI-controllers are explained.

PI-controllers are commonly used to regulate the behaviour of dynamic systems in the time domain. They are integral parts of control systems. A control system can be classified as either an open-loop or a closed-loop system. In an open-loop system, the value of the controlled system's output variable $y(t)$ is directly determined by the input (or desired) variable $x_d(t)$. A closed-loop control system is shown in **figure 5.1**. The output value $y(t)$ of the controlled system is measured and fed back to the input of the system where it is subtracted from the desired value $x_d(t)$ to form the error signal $e(t)$. The error signal $e(t)$ is the input signal to the controller.

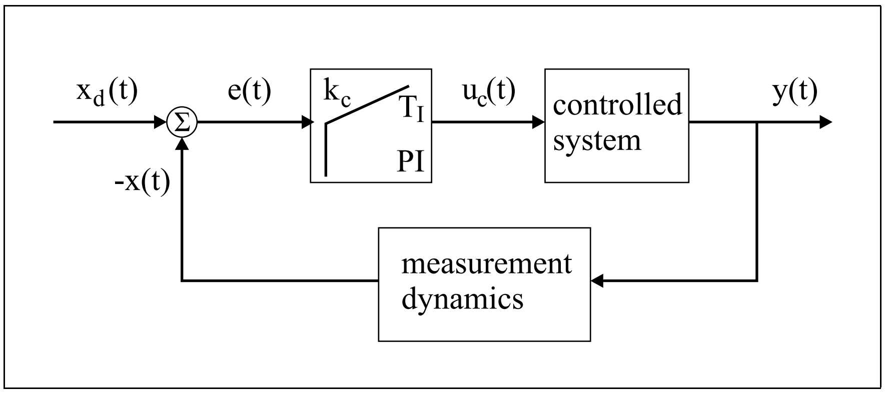


Figure 5.1: *A closed loop control system with a PI-controller*

The task of the PI-controller is to generate the control variable $u_c(t)$ from the error signal $e(t)$. The transfer behaviour of the PI-controller can be led back to the two basic forms of the linearized P- and I-element. Therefore, a PI-controller can be represented by a parallel connection of the respective elements as shown in **figure 5.2**. The block diagram shows the transfer functions of every element in the Laplace domain.

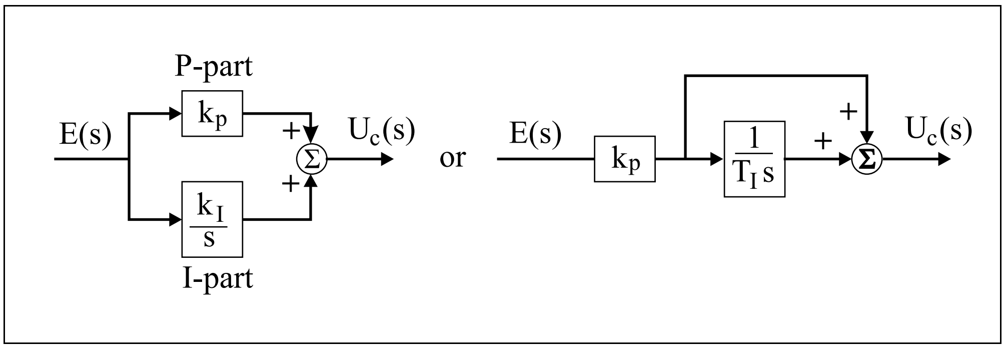


Figure 5.2: *Block diagram of a PI-controller*

From the left-hand side in **figure 5.2** follows the transfer function of the PI-controller

$$G_c(s) = \frac{U_c(s)}{E(s)} = k_p + \frac{k_I}{s}$$

This transfer function can be rewritten after introducing some parameters:

$$\begin{aligned} k_c &= k_p && \text{proportional gain} \\ T_i &= \frac{k_p}{k_I} && \text{reset time or integral time} \end{aligned}$$

The **transfer function** for the PI-controller, using the above parameters, looks as follows:

$$\boxed{G_c(s) = k_c \left(1 + \frac{1}{T_I \cdot s}\right)} \tag{5.1}$$

The shape of equation (5.1) leads to the PI-controller's representation of the right-hand side in **figure 5.2**. Application of the inverse Laplace transformation to equation (5.1) leads to a representation of the control variable in the time domain

$$\boxed{u_c(t) = k_c \cdot e(t) + \frac{k_c}{T_I} \int_0^t e(\tau)\, d\tau} \tag{5.2}$$

where $e(t) = x_{desired} - x(t)$ is the control deviation or error signal.

Equation (5.2) is called the **integral equation of the PI-controller**. This equation can be easily used to find the controller's response to a step impulse $\sigma(t)$ at its input. The step response $h(t)$ for $e(t) = \sigma(t)$ is shown in **figure 5.3**.

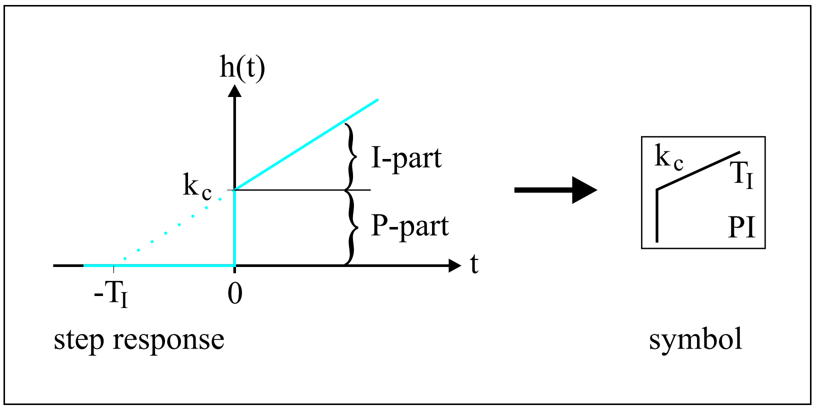


Figure 5.3: *Step response of a PI-controller and the dedicated symbol*

At time 0, $h(t)$ jumps because of the proportional part of equation (5.2). While t grows, $h(t)$ rises linearly due to the integral part of equation (5.2). The shape of the step response in **figure 5.3** is drawn in the PI-controller's block symbol. The characteristical parameters k_c and T_I are given in the block symbol as well.

The PI-controller is provided with advantages as well as some disadvantages. It is a special case of the PID-controller. In a PID-controller, the D-part achieves a faster transient time. It is left out because this is a minor advantage compared to the fact that a controller with a D-part is not always easy to handle. The PI-controller overshoots as much as the P-controller and it has also an equivalent transient time. It yet has no permanent offset because of its I-part.

In this work, the PI-controller is not used in the shape of an electronic circuit but in an algorithmic version which can be used on a computer. Throughout the implementation of PI-controller on a computer system, it has to be the goal, that the output of a PI-algorithm must be essentially the same as the output of an analogous PI-controller. More precisely, the step response of a digitally controlled system must not differ from the results of an analogous controlled system. For to reach that goal, a requirement has to be met to process the digital implementation successfully. The sample time T_{sample}, which appears because of the sequential mode of operation of a computer, has to be very small compared to the time constant of the controlled system T_{system}. Only if this is guaranteed, the digital control circuit can be assumed to be quasi continuous. In that case, the data from the analogous closed-loop control design can be used to parameterize the digital PI-controller. As a rule of thumb, the following numerical proportion is valid:

$$T_{sample} \leq \frac{1}{10} T_{system}$$

If the sample time of the digital controller is too big, it becomes unstable and cannot be used.

There are two different versions of the PI-algorithm. The first one is the position algorithm and the second one is the velocity algorithm.

5.1.1 The position algorithm of the PI-controller

The integral equation of the PI-controller (equation (5.2)) can be transferred from the time continuous domain to the time discrete domain. For small sample times T_{sample}, the integral can be replaced by a sum

$$\boxed{u_{c,K} = k_c \left(e_K + \frac{T_{sample}}{T_I} \sum_{\nu=1}^{K} e_{\nu-1} \right)} \tag{5.3}$$

with $K = \frac{t}{T_{sample}} = 0, 1, 2, \cdots =$ discrete time unit

Mathematically, the exact calculation of the area under the curve for the control deviation is approximately replaced by the summation of small rectangles. This algorithm is referred to as the position algorithm of the PI-controller. In this form, the PI-algorithm is rarely used on computers because the summation of the last $K - 1$ control deviations can be done differently with a recursive version of the algorithm, which is called velocity algorithm of the PI-controller.

5.1.2 The velocity algorithm of the PI-controller

The velocity algorithm of the PI-controller uses only the current value of the control deviation to correct the summation. In order to achieve this, the sum in equation (5.3) can be broken down into the previous $K-1$ summands and the current K^{th} summation.[1]

$$u_{c,K} = k_c \cdot e_K + k_c \frac{T_{sample}}{T_I} \sum_{\nu=1}^{K-1} e_{\nu-1} + k_c \frac{T_{sample}}{T_I} e_{K-1} \tag{5.4}$$

The output of the controller for $K-1$ is

$$u_{c,K-1} = k_c \cdot e_{K-1} + k_c \frac{T_{sample}}{T_I} \sum_{\nu=1}^{K-1} e_{\nu-1} \tag{5.5}$$

Equation (5.5) can be subtracted from equation (5.4). The result is the following algorithm:

$$\boxed{u_{c,K} = u_{c,K-1} + k_c \left(e_K - e_{K-1}\right) + k_c \frac{T_{sample}}{T_I} e_{K-1}} \tag{5.6}$$

Equation (5.6) represents the velocity algorithm of the PI-controller. It can be interpreted as follows: The first term is the previous history of the control variable. The second term represents the current P-part of the controller, and the third term is the current I-part of the controller. Because of the discrete time unit, the current value K just has to be distinguished from the old value $K-1$. Thus, only two values, namely the control deviation e and the control variable u_c, have to be stored. All values which are older than $K-1$ can be deleted.

In the following, the step response of the velocity algorithm of the PI-controller is shown. A constant control deviation $e_K = e_{K-1} =$ *const* is assumed for equation (5.6). The step response in staircase form can be obtained by calculating the control variable for a few sample steps. The result can be seen in **figure 5.4**.[2]The linear rise of the step response, which is known from the PI-controller's block symbol, becomes a staircase shape. The smoothness of the staircase is directly correlated with the sample time T_{sample}.

[1](HOFER, 1998), pp. 83.
[2]Ibid., p. 85.

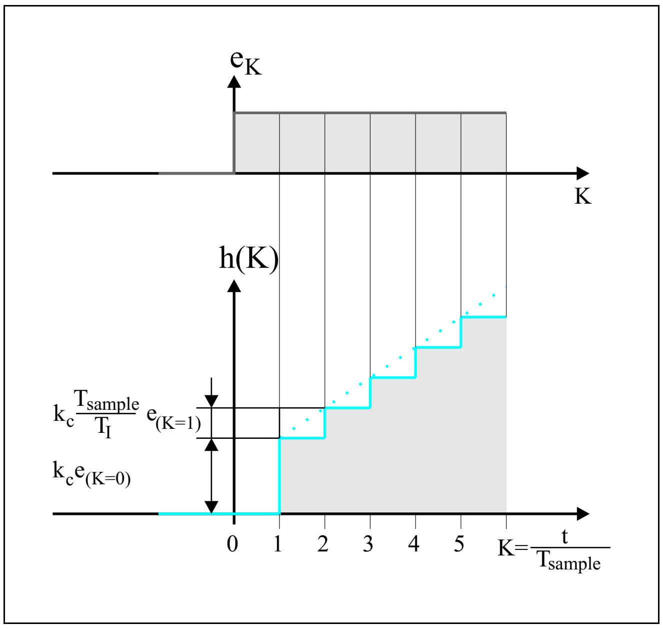


Figure 5.4: *The step response of the PI-controller's velocity algorithm in staircase form*

Finally, the velocity algorithm is shown in **figure 5.5**. The control variable u_c can be calculated via equation (5.6).

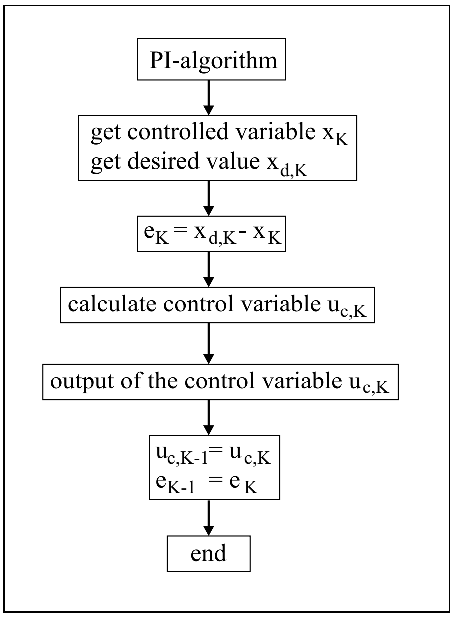

PI-algorithm
get controlled variable x_K
get desired value $x_{d,K}$
$e_K = x_{d,K} - x_K$
calculate control variable $u_{c,K}$
output of the control variable $u_{c,K}$
$u_{c,K-1} = u_{c,K}$
$e_{K-1} = e_K$
end

Figure 5.5: *Flowchart of the PI-algorithm*

5.2 One-dimensional filters

The concept of one-dimensional filters is known from the field of manipulator and arm robotics for the generation of a trajectory. John Lloyd (1998) showed that one-dimensional filters are stable.[3] This section explains what a one-dimensional filter does and examines the different ways of calculating a profile. The explanations are mainly derived from Torsten Scherer (1998) who followed Lloyd's approach but used different computations.[4]

The one-dimensional filter is the task to bring a point p, which has a velocity v_0 at the beginning, to a constant target t with the help of a velocity profile. "This velocity profile must be computed so that the velocity $v(t)$ does not exceed some maximum velocity v_m and that the area under the profile equals the distance d towards the target."[5]The minimum time t_m which the point p needs to reach the target can be calculated. The time t_m can be stretched to $t_s \geq t_m$ so that the velocity limit v_m is not violated. Furthermore, the possibility of stretching the time is useful to sync more than one filter but this aspect is not needed in this work.

Depending on the initial condition, different types of velocity profiles can be computed. If the initial condition is static, which could mean that the velocity of the point p is zero, the computed profile is referred to as a *static profile*. Alternatively, the initial condition may be dynamic, which means that the velocity of the point p is *not* zero, the computed profile is then called a *dynamic profile*.

For the purpose of this work, the target velocity is assumed to be zero. As a result of this, the actual robot stops at the end of each profile. At this point, it should be mentioned, that the following computations could include a target velocity as well. This would result in more case switches.

5.2.1 Static profiles

A static profile is easy to compute. Let without loss of generality the distance to the target be positive. The point p is accelerated to a peak velocity v_p and decelerated so that it stops exactly on the target. **Figure 5.6 a)** shows an example for such a static profile.

From Scherer (1998), it is known that the distance d between the starting and the target position has to be equal to the area under the profile and can be computed with the following equation because of the symmetric situation

$$d = \frac{v_p^2}{a}$$

[3](LLOYD, 1998).
[4](SCHERER, 1998), pp. 17-27.
[5]Ibid., p. 17.

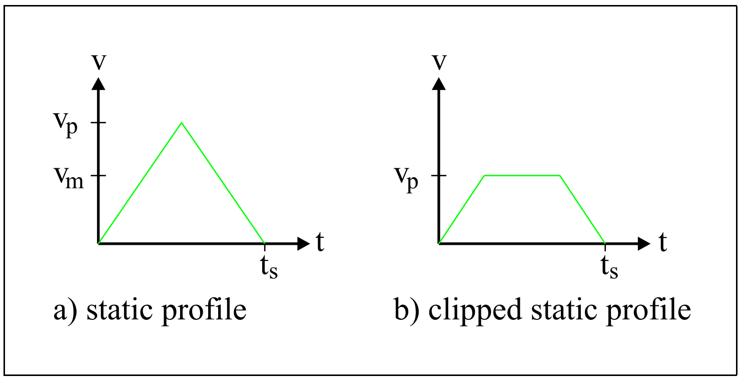


Figure 5.6: *Static profile*

where a is the acceleration.[6] Given the distance and acceleration, the peak velocity can be calculated as

$$v_p = \sqrt{ad}$$

The time t_s which the point p needs to reach the target is

$$t_s = \frac{2v_p}{a}$$

If v_p exceeds the allowed maximum velocity, the profile must be clipped as shown in **figure 5.6 b)**. The resulting profile has a phase of constant velocity and the distance computes as

$$d = v_p t_s - \frac{v_p^2}{a}$$

Then, the time t_s is

$$t_s = \frac{d}{v_p} + \frac{v_p}{a}$$

5.2.2 Dynamic profiles

Under an initial condition, which is dynamical, the computation of a profile is more complex. Three different situations are possible:

1. Velocity towards target
2. Velocity towards target, but too fast to stop on the target
3. Velocity away from target

[6](SCHERER, 1998), p. 18.

In the first situation, the point p has to be accelerated to the peak velocity v_p and then decelerated to stop exactly on the target. In some cases, only deceleration is necessary to stop the point on the target. A closer look on this situation will be outlined later. In the second situation, the point p is decelerated to full stop. Since the velocity was too fast to stop on the target with given maximum deceleration, the point p overshoots. In this case, the target can be reached with the help of a new movement with a negative static profile. In the third situation, the point is decelerated to stop and then moves to the target with a static profile. In all situations, the velocity has to be clipped if $v_p > v_m$. Then, the point moves some time with constant velocity v_m. The three situations are shown in **figure 5.7**.

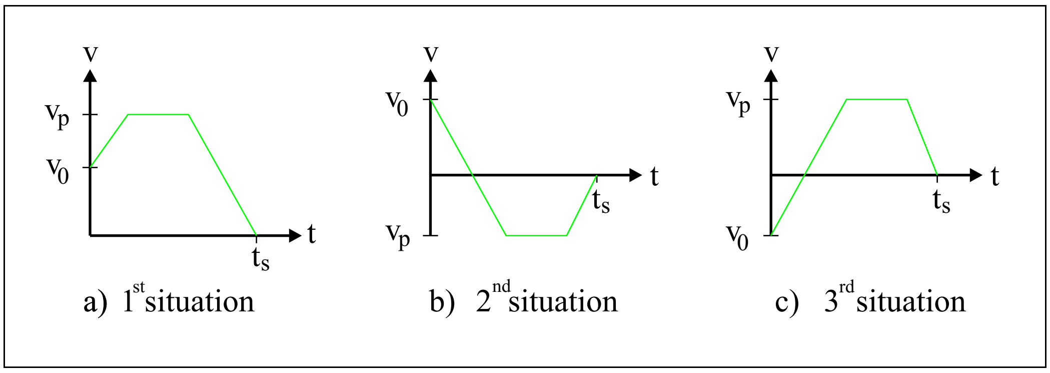


Figure 5.7: *Dynamic profiles*

The second situation can be recognized by checking if the travelled distance $\frac{v_0^2}{2a}$ to full stop is larger than the real distance d. The third situation can be recognized by comparing the sign of the velocity with the sign of the distance. The first situation has to be paid more attention, let again, without loss of generality, the distance be positive. In this situation, the distance d is larger than the distance $\frac{v_0^2}{2a}$ which is the distance the point p travels when doing a full stop. The point has to be moved an additional distance d^*

$$d^* = d - \frac{v_0^2}{2a}$$

which can be achieved very fast "[...] by accelerating to some peak velocity $v_p > v_0$ and from there decelerating back to v_0 [...]"[7]. The resulting profile can be seen in **figure 5.8**.

[7](SCHERER, 1998), p. 21.

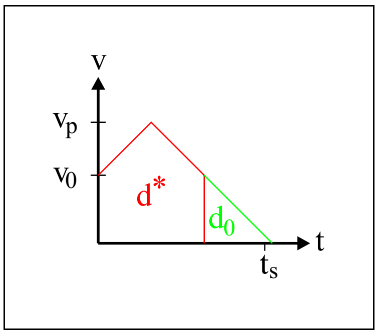


Figure 5.8: *Extra distance d^**

The peak velocity of the resulting profile can be computed as follows:

$$v_p = \sqrt{ad + \frac{v_0^2}{2a}}$$

If v_p is larger than the velocity limit v_m two cases can occur. They are shown in **figure 5.9**.

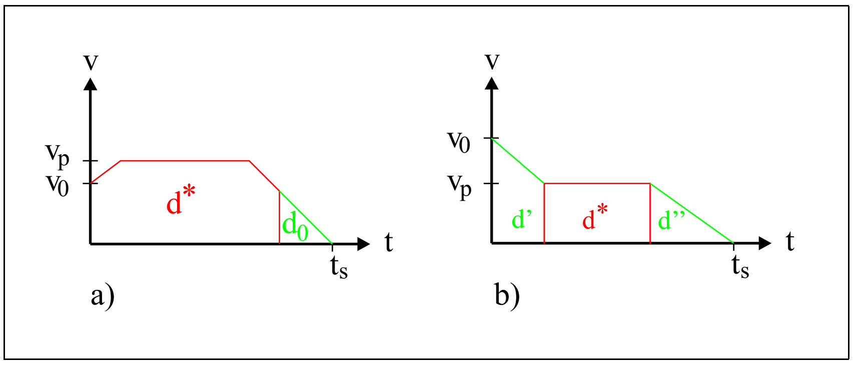


Figure 5.9: *Ramping up or down*

It is possible that v_0 has to be ramped up or down to v_p. Therefore, the travelled distance computes as:

$$\begin{aligned} d_{v0<v_p} &= v_p t_s - \frac{v_p^2}{2a} - \frac{(v_p - v_0)^2}{2a} \\ d_{v0>v_p} &= v_p t_s - \frac{v_p^2}{2a} + \frac{(v_p - v_0)^2}{2a} \end{aligned}$$

The time t_s needed for the motion can be computed in the following ways

$$\begin{aligned} t_{s_{v_0 < v_p}} &= \frac{d}{v_p} + \frac{v_p - v_0}{a} + \frac{v_0^2}{2av_p} \\ t_{s_{v_0 > v_p}} &= \frac{d}{v_p} - \frac{v_0^2}{2av_p} + \frac{v_0}{a} \end{aligned}$$

and the peak velocities for the two cases are

$$\begin{aligned} v_{p_{v_0 < v_p}} &= \frac{v_0 + at_s - \sqrt{2av_0t_s + a^2t_s^2 - 4ad - v_0^2}}{2} \\ v_{p_{v_0 > v_p}} &= \frac{v_0^2}{2(v_0 - at_s)} - \frac{ad}{v_0 - at_s} \end{aligned}$$

More details about the derivation of these equations for dynamic profiles can be found in Scherer (1998).[8]

5.2.3 Discretization

The introduced one-dimensional profiles have still to be fitted to an application like robot trajectory generation. Therefore, the continuous profiles are converted to discrete profiles. Let Δt be the time interval of the trajectory generator. The discretization is achieved by integrating the profile from 0 to Δt. This yields the new position as well as the new velocity of the robot. Since the robot system is inertial, it may not be able to reach the desired velocity and position of the profile. This problem is dealt with in the following sections. The input to the one-dimensional filter profile is the estimate of the current system-state obtained by the extended Kalman filter.

5.3 Control of the rotation

The rotation controller adjusts a desired orientation of the mobile platform. It consists of two stages. At the first stage a control cycle with a one-dimensional filter adjusts the desired orientation angle coarsely. At the second stage a PI-controller (target controller) performs a fine tuning of the robot's orientation angle. **Figure 5.10** shows the two-stage structure of the rotation controller.

In every cycle of the first stage, the one-dimensional profile for the rotation is calculated by the filter. In **figure 5.11**, the profile is shown exemplarily. With an assumed angular acceleration $\dot{\omega}_{robot}$, a maximum angular velocity $\omega_{robot,max}$ and the desired orientation which the robot has to

[8](SCHERER, 1998), pp. 22-26.

reach, the whole shape of the profile is determined. Even the time t_{goal} which the robot needs for this motion is known but is of no importance for further considerations in this section. The definite integral under the profile between two points of time is equivalent to a small fragment $\Delta\varphi$ of the whole rotation.

Just as in the translation controller in section 5.4, the output ω'_{robot} of the one-dimensional filter is passed through the collision avoidance. Depending on the presence of objects in the vicinity of the mobile platform, the robot's angular velocity ω'_{robot} is bounded by the collision avoidance. Its output $\omega_{robot} = \dot{\varphi}$ is transformed into the drive wheels' angular velocities ω_L and ω_R which are sent to the drive controller.

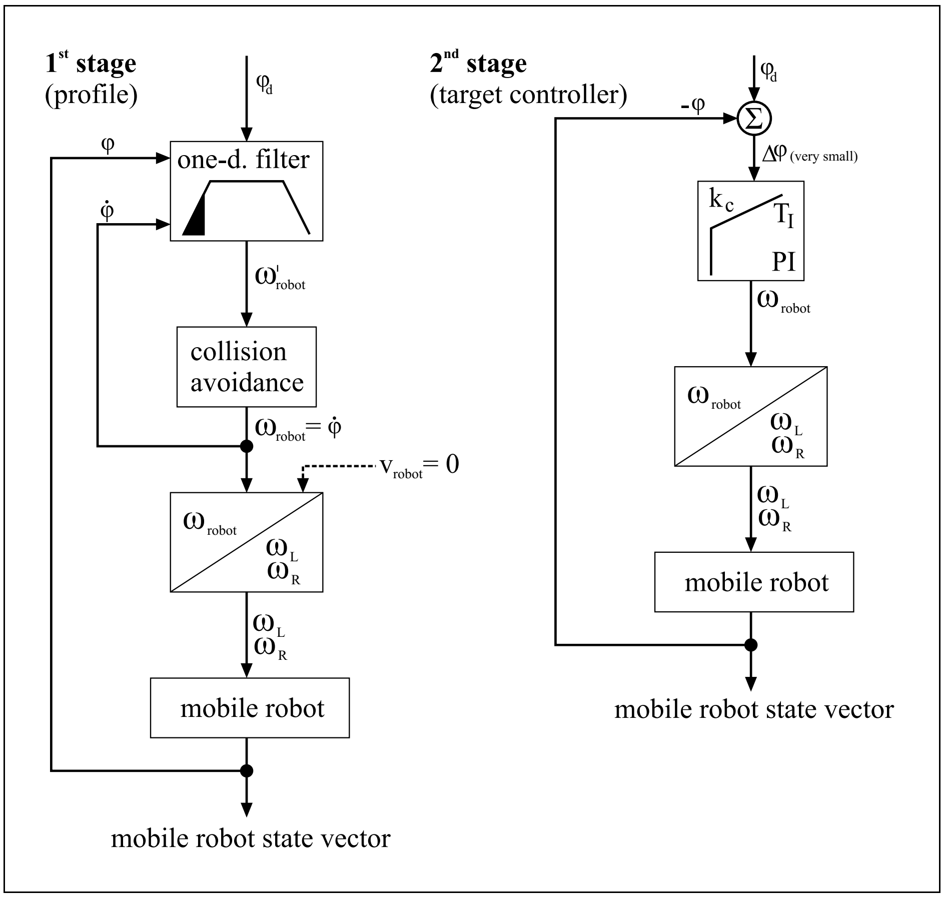


Figure 5.10: *Block diagram of the rotation controller*

At the beginning of each new cycle, the one-dimensional filter calculates the angular velocity profile due to the actual orientation φ , the desired orientation φ_d and the angular velocity ω_{robot}. Once the filter has done that, it puts out the current ω'_{robot} which is the profile's amplitude at that time.

If the filter has reached the target and the filter's output has become zero, the first stage is finished. At this point, the target controller (PI-controller) takes over the control. As shown in **figure 5.10**, the difference between the desired angle φ_d and the current orientation φ is calculated. The PI-controller generates the output value ω_{robot} which, again, is transformed into the angular velocity for the left and right drive wheel. The PI-controller is stopped after n_{max} cycles.

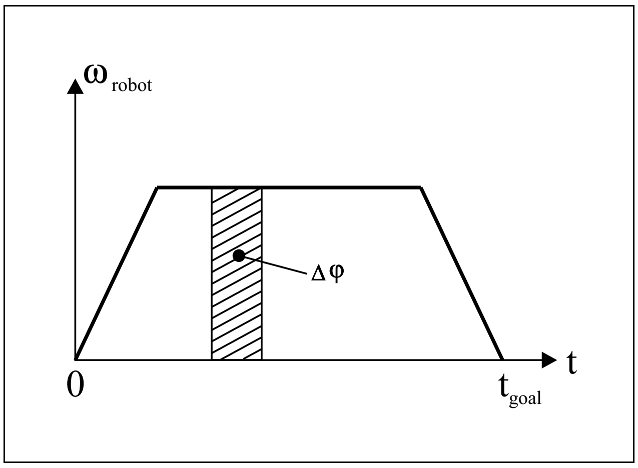


Figure 5.11: *Rotation profile which is generated by the one-dimensional filter.*

If the motion has become small, the PI-controller is stopped earlier but not before a minimum number of cycles n_{min}. A "small motion" in this case means that the difference between the orientation in a cycle and the orientation in the preceding one is smaller than a bound b ($\varphi_n - \varphi_{n-1} < b$). Practical values for all parameters in this section are shown in **table 5.1**.

parameter	value
k_c	2.0
T_I	0.5
n_{min}	50
n_{max}	200
b	0.15°

Table 5.1: *Values of the parameters used in the rotation controller*

5.4 Control of the translation

The translation controller generates the control commands to steer the mobile robot from a start point $\vec{p}_{start}$ to a goal point $\vec{p}_{goal}$ on a straight line. It is a precondition that the orientation of the mobile platform before the beginning of the translation must be approximate to the goal point. The deviation from the straight line during this motion must be kept as small as possible.

The translation controller can be divided into two functional parts. The first part is a one-dimensional filter as described in section 5.2. In every cycle, this filter generates a dynamical profile based on the current position $\vec{p}_{robot}$ and the velocity v_{robot} of the mobile robot as well as the goal position $\vec{p}_{goal}$. The filter output is the translational velocity v'_{robot} of the mobile robot. The second part is a PI-controller which minimizes the deviation from the desired trajectory from $\vec{p}_{start}$ to $\vec{p}_{goal}$. Its output is the angular velocity ω'_{robot}.

In **figure 5.12**, the geometrical setup for the described situation is shown. With the aid of the ex-

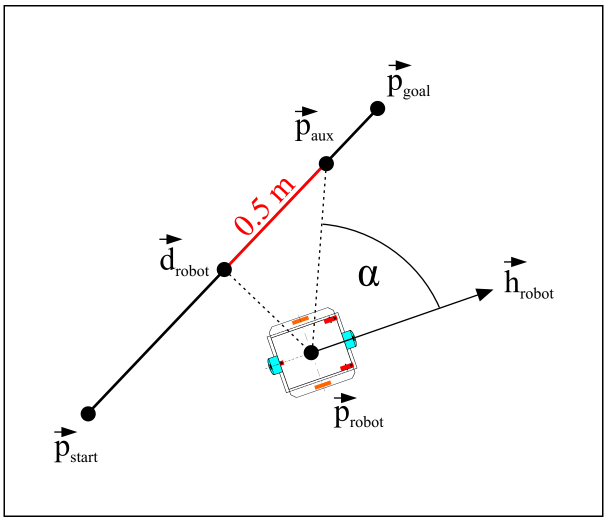


Figure 5.12: *Geometrical diagram for the mobile robot's motion from $\vec{p}_{start}$ to $\vec{p}_{goal}$*

tended Kalman filter, the current robot position $\vec{p}_{robot}$ is identified in every cycle. The orientation of the robot is specified by the vector $\vec{h}_{robot}$. In a first step, the position $\vec{p}_{robot}$ is projected onto the desired trajectory, which is the straight line from $\vec{p}_{start}$ to $\vec{p}_{goal}$ represented by an associated unit vector. The vector $\vec{d}_{robot}$ marks this projection point on the trajectory. After the difference vector $\vec{d}_{robot} - \vec{p}_{start}$ has been determined, the ratio between $||\vec{d}_{robot} - \vec{p}_{start}||$ and $||\vec{p}_{goal} - \vec{p}_{start}||$ can be computed. It indicates the robot's progress along the trajectory. Based on this ratio and on the robot velocity v_{robot}, after being bounded by the collision avoidance, the filter updates the one-dimensional velocity profile for the remaining distance to the goal. Then it gives the output of the current velocity value for the robot in direction of its translation.

Due to small velocity differences between the left and right drive wheel, induced by differences of the low-level motor controllers and slippage errors, the robot drifts away from the desired trajectory. In a second step, the PI-controller minimizes this drift by steering the mobile platform back to the desired path. For this purpose, an auxiliary point $\vec{p}_{aux}$ is introduced which is located on the trajectory in a certain distance from the projection point $\vec{d}_{robot}$ in forward direction. In practice, a value of $0.5\ m$ for the above mentioned distance has proven to be reasonable. The deviation angle α between the current viewing direction $\vec{h}_{robot}$ of the mobile platform and the line through $\vec{p}_{robot}$ and $\vec{p}_{aux}$ is calculated. α is the input value to the PI-controller which decreases this angle by providing a suitable controller output ω'_{robot}.

The respective control flow for the control of the translation is shown in **figure 5.13**. As one

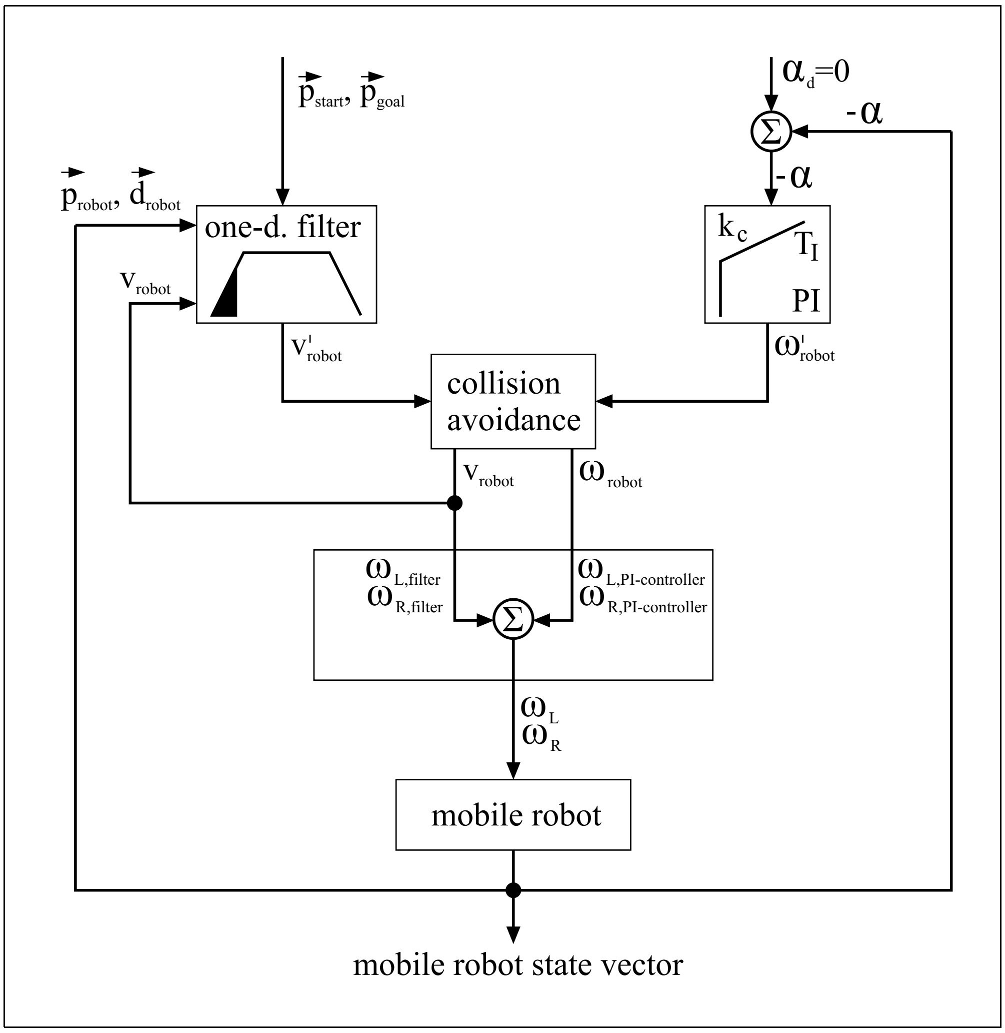


Figure 5.13: *Block diagram of the translation controller*

can see, the filter's output of the translational velocity v'_{robot} and the PI-controller's output of the angular velocity ω'_{robot} are both bounded by the collision avoidance module which reduces the above values depending on the distance to an obstacle. The outputs v_{robot} and ω_{robot} of the collision avoidance then serve as inputs to the following modules where they are transformed into angular velocities ω_L and ω_R for the drive wheels. The transformed outputs of the one-dimensional filter and of the PI-controller are superimposed. Finally, ω_L and ω_R build the drive command which is sent to the mobile platform's drive. This alters the robot's speed, its position and orientation. In the next cycle, the new projection of the robot position onto the trajectory and the velocity v_{robot} as well as the new deviation angle α serve as inputs to the one-dimensional filter and to the PI-controller.

The robot has finished its trajectory when the one-dimensional filter reaches the target or overshoots. In this case, ω_L and ω_R are set to zero and the mobile platform is brought to a rest which can be assumed to be a rather slow velocity. In the case of overshooting, no more correction is applied to the robot's position because in the regarded setup errors in x/y positioning are much more tolerable than errors in orientation, which can occur when the robot tries to go back a small distance after overshooting. The parameters for the PI-controller are given in **table 5.2**

parameter	value
k_c	1.2
T_I	50.0

Table 5.2: *Values of the PI-controller's parameters used in the translation controller*

Chapter 6

Path Planning for Mobile Robots

When a mobile robot moves through its environment, avoiding collisions is a crucial problem to be solved. This problem can be divided into two main tasks. Firstly, the robot has to be able to calculate a collision-free path from its present position to a desired goal point. The foundation for this could be a map in the classical sense (as in this work). Secondly, the robot could come across a dynamic obstacle while following its route to the target. Managing the latter problem, however, is not dealt with in this work. Suggestions for dynamic obstacle avoidance are given in Arkin (1998).[1] The robot platform *GenBase II* already has a low-level collision avoidance which stops the robot immediately whenever an obstacle is within a predefined range of the robot centre point.

This chapter begins by giving a brief overview of different approaches to map, robot and path representation and by discussing which of these approaches are suitable for the *GenBase II* platform. In section 6.2, the particular representation used for the robot and its environment is explained in detail. In consequence of the chosen form of representation, section 6.3 provides conclusions for the generation of collision-free path segments. In this context, a detailed description of the V-Graph and T-Graph models is given. Finally, in section 6.4, the A^* Algorithm is utilized to find the shortest path from start to goal via the set of calculated path segments.

6.1 Approaches to map, robot and path representation

For the development of mobile robots, it is fundamental to find the right design to represent a map. A map is here understood to be the robots knowledge about its environment. It is the only pattern against which the robot can try to match its sensory input, which is very often high-dimensional, and hence difficult to project onto the map. Once a map representation of the

[1](ARKIN, 1998).

environment is found, the next step is to find a suitable representation of the robot itself within this map. Thereafter the path the robot needs to take must be entered into the map. Depending on the representation of the path, an appropriate search algorithm must be used. The algorithm has to find the final path which satisfies a given optimality criterion (for example finding the shortest path). There are different solutions for those four sub-tasks. In the next four sections, a brief overview of approved methods will be given, resulting in a conclusion for the *GenBase II* robot platform.

6.1.1 Map representations

One aim of research in mobile robotics is to increase a mobile robot's efficiency in performing certain tasks. A task in this sense is a particular aspect of the mobile robot's abilities, for example path planning, self-localisation or obstacle avoidance. Performing a task is influenced by the robot's environment. For example, in a real-world scenario the environment can be either dynamic or static, restricted or unrestricted as well as known or unknown to the robot. A highly dynamic setup is represented by the robot soccer (RoboCup[2]) in which a robot has to be aware of all its moving opponents and teammates. Another task with a different characteristic of the environment is the planetary exploration in an unknown landscape. These examples show that the choice of a particular map representation of the environment can be helpful to accomplish a task.

Charles Gallistel (1990) has stated a geometric concept that he called "strength". It allows to compare different map representations. The "strength" is "[...] the range of geometric relations among the mapped points that could in principle be recovered from the map [...]" [3]

The following description holds four basic categories of map types with different "strengths":[4]

Recognizable Locations: The map consists of a list of locations which can be reliably recognized by the robot. The recognition is only a qualitative but not a quantitative one, which means that no exact values for the positions are available.

Topological Map: In addition to the recognizable locations, the map records which locations are connected by traversable paths.

Metric Topological Map: This term is used for maps in which distance and angle information is added to the path descriptions.

Full Metric Map: Object locations are specified in a fixed coordinate system. Precise positional information is provided.

[2]For detailed information about robot soccer and the RoboCup see *http://www.robocup.org*.

[3](GALLISTEL, 1990).

[4](LEE, 1996), p. 18.

Another approach to map representation is to think about a way in which all objects of the real world, the paths and restricted areas as well as free spaces can be stored. These aspects are shown in the following short list.

Path: A list of sub-paths, which can be combined to get the final path, is stored. Obstacles are not considered. A sub-path can be obtained by recorded motion. This is often used in industrial robotics.

Free space modelling: Free spaces in the environment are partitioned with the help of Voronoi cells. The Voronoi cells are stored in a graph.[5]

Object modelling: Objects in the environment are stored in graphs. The objects can be represented by their vertices (vertex graphs).

Composite space: The environment is partitioned by using grids, quadtrees or octrees. Each element of the grid, the quadtree or the octree is either free space or restricted area.

The mobile robot application which underlies this work requires great accuracy in self-localisation and self-positioning. A full metric map is the only representation which gives a precise description of all obstacle positions with respect to a global coordinate system. Therefore, it is an appropriate choice for obtaining the desired accuracy. Since blueprints of buildings are always available, they can easily be converted into an object modelling representation. This representation is employed in combination with the full metric map approach in this work (see section 6.2).

6.1.2 Robot representations

It is crucial for collision avoidance to know the physical dimensions of the mobile robot. Therefore, a high level of detail for the representation of a robot in a map is useful. On the other hand, manageable methods for collision avoidance must be available. A trade-off between an appropriate level of detail and manageable methods has to be found in order to establish a suitable robot representation. The following list shows some common description methods.

Actual physical shape: The shape of the robot is modelled in detail.

Surrounding rectangle: The mobile robot is modelled in the shape of a rectangle whose edges lean against those parts of the robot that stick out furthest.

Surrounding circle: This follows the same idea as the surrounding rectangle, but uses a circle instead.

[5](SEDWICK, 1992), pp. 465-467.

Surrounding circle with negligible radius: The surrounding circle's radius is made infinitely small (dot-shaped) and the other objects in the map are expanded by the original radius of the robot.

Path planning for a detailed robot representation is computationally more expensive than for a robot represented by a rectangle or circle. There are even more simple and robust algorithms for robots represented by a negligible radius. The V-Graphs and T-Graphs are designed for this kind of representation (see section 6.3.1 and section 6.3.2). Since there is a fast and stable way to expand an object's representation in a map (see section 6.2), a dot-shaped robot representation is used in this work.

6.1.3 Path representations

In path planning, there are different possibilities to generate and represent a path. Some of them are given in the following list:

Functional description: The path is described by functions, for example with B-Splines.

Vector of coordinates: In a grid-shaped representation, the path is a chain of positions of the mobile robot (with a statistical probability of the robot's location).

Linear representation: This is a description of the path based on simple line segments.

The first option provides a closed mathematical solution but it is computationally expensive. The second option is memory consuming depending on the map representation's level of detail. The linear representation can be stored efficiently. Algorithms like the V-and T-Graph generate functional descriptions of paths given by line segments. This makes the linear representation of a path advisable. It is used in the following sections.

6.1.4 Search algorithms

Depending on the type of map, robot and path representation, an algorithm must be chosen for finding a complete path from a start point, which might be the current robot position, to the robot's goal point. In computer science, a lot of different search strategies are known. In order to make the right choice for the current path-finding problem, one can distinguish search strategies as follows:

Uninformed search: This is a search that does not make use of prior knowledge about the problem. Instead, new states are generated systematically and then compared with the goal.

Informed search: Problem-specific knowledge is used if it is available.

If the environment of a mobile robot is known informed search algorithms can be applied. Russel and Norvig (1995) split up informed search methods into four subgroups.[6]

1. **Best-first search**
    - minimization of the estimated cost to reach a goal (Greedy search)
    - minimization of the total cost of a path (A^* search)
2. **Search with heuristic functions**
    - minimization of a cost function which is an estimate of the unknown real cost function of the solution
3. **Memory-bounded search**
    - iterative deepening A^* search (IDA^*)
    - simplified memory-bounded A^* search (SMA^*)
4. **Iterative improvement algorithms**
    - hill-climbing search
    - simulated annealing
    - applications in constraint satisfaction problems

The first search strategy in this list, the best-first search, fits in with the given presentation of the problem of finding an entire path from a start point to a goal point. The idea of minimizing total path costs from the robot's current state to its goal can be easily applied to the path-finding problem. Therefore, the A^* algorithm has been chosen to solve this problem (see section 6.4.1).

The cost function which is used for the A^* algorithm is divided into known costs and a heuristic estimate of the unknown costs. Because of this cost function, the A^* algorithm is better than the search with purely heuristic functions. Memory-bounded search, as its name suggests, reduces the memory requirements during the search operation. The representation of maps chosen in this work does not indicate the necessity to reduce the amount of required memory. Iterative improvement algorithms like hill-climbing search are less suitable in this context because of their well-known drawbacks like local minima (maxima), plateaux and ridges.

[6](RUSSEL AND NORVIG, 1995), pp. 92-115.

6.1.5 Conclusions for the mobile robot platform *GenBase II*

For the present system, some preconditions have already been stated.

- The positioning abilities of the system must have a precision in the order of a few millimetres.
- The orientation must be adjustable in the order of less than one degree.
- The shortest amongst all possible paths must be found.

If all these prerequisites are taken into consideration, the choice of reasonable map representations is restricted. The composite space representation can be excluded because a discretization of the environment using a millimetre-grid needs a lot of memory space especially for big maps. Furthermore, areas of small interest would be represented with undue accuracy. As stated at the end of section 6.1.1, a full metric map representation together with object modelling fits best to the prerequisites. In addition, object modelling provides further advantages. For the generation of the robot's configuration space, the objects can be easily scaled.[7]

For solving the path-finding problem, the A^* algorithm has been suggested in section 6.1.4. This choice implies the representation of the robot as a circle with negligible radius. Moreover, a linear path representation is used. These approaches are described in the following sections.

6.2 Representation of the robot and its environment

In this section, the design of the map and robot representation chosen for this work is explained. The concept of configuration maps is introduced. The resulting problems and their solutions are pointed out.

6.2.1 Shrinking of the robot, expansion of the map

A full metric map can be represented geometrically by a set of lines or a set of polygons. At the end of section 6.2.2 it is shown that using polygons results in difficult problems. These problems can be avoided by using a set of lines instead. A line is given by its start and end point (x,y pair).

[7]For a detailed explanation of the concept of configuration space, see section 6.2.1. In section 6.2.2, the concept of map expansion is explained.

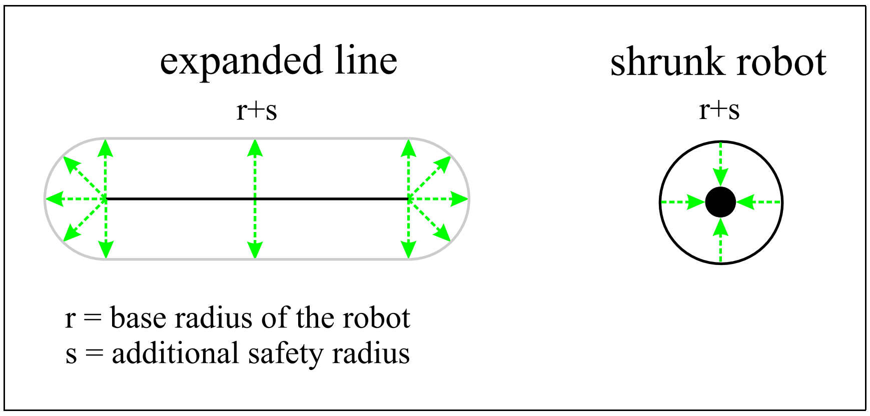


Figure 6.1: *Line expansion ("cigar shape") and shrunk robot*

Each line is interpreted as an impassable obstacle for the robot. For the solution of the path-planning problem, this means the robot is never allowed to cross a line. The algorithms introduced in the following sections conduct a path search under the assumption of a dot-shaped robot. Obviously, no existing robot can satisfy this constraint. For practical use, however, it is suitable to describe a robot as a disc which encloses the robot's real physical shape. Expanding each line of the map by the radius of that disc allows to shrink the robot to an infinitely small circle. Exemplarily, the expansion process is described in **figure 6.1**. Path planning now means to find a path for the disc's centre point in the expanded map.

The expanded map is referred to as the **configuration map** or the **configuration space**.

For the sake of security, the radius of the disc is considered to consist not only of the robot's actual shape, but also of an additional safety ring around it (see **figure 6.1**).

Whereas the earlier restriction required that the robot must not cross any line of the map, the robot is now not allowed to be in any of the cigar-shaped line expansions.

The revised strategy has the following advantages and disadvantages:

Advantages:

- Every path in the configuration map is passable for the actual robot.
- A minimum safety distance can be simply selected.
- No attention has to be paid to the orientation of the robot.
- Map expansion is only a one-time computational effort.
- Map expansion can be done in linear time $O(n)$.

Disadvantages:

- Possibly existing paths can be lost through the assumption that the robot is disc-shaped.
- The approximation of the configuration map is another source of error (see following section).

For drawing a conclusion, there is a need to pay attention to the disadvantages. The loss of paths concerns mainly those which lead through small gaps in the original map. These gaps are closed because of the line expansions due to the assumption of a circular robot. The real robot might be able to pass these gaps. Since at least one point of the robot's body lies on the bounding circle, it wouldn't be able to rotate when being in the gap. Therefore the above strategy avoids situations in which the robot is not allowed to rotate. In the following sections it is shown that the disadvantage of approximating the configuration map can be eliminated.

6.2.2 Problems of map expansion

Map expansion causes problems in its practical implementation. The correct expansion of a line segment L_i is the union of all discs C_n with the radius $r + s$. The centre points $\vec{c}_n$ of the discs have to be elements of the line segment L_i.

$$\text{Expansion}(L_i) = \left\{ \bigcup C_n \in C \;\mid\; \vec{c}_n = \vec{l}_i \wedge \vec{l}_i \in L_i \right\}$$

with C being the set of all possible circles

$$C = \{C_n(\vec{c}_n, r + s)\}$$

and L_i being the set of all points belonging to the respective line segment

$$L_i = \left\{ \vec{l}_i \;\mid\; \vec{l}_i = \vec{p}_{i,start} + \lambda(\vec{p}_{i,end} - \vec{p}_{i,start}) \wedge \lambda \in [0 \ldots 1] \right\}$$

Each line segment is described by its start point $\vec{p}_{i,start}$ and its end point $\vec{p}_{i,end}$. The result of a line expansion can be seen in **figure 6.2 a)**.

The cigar-shaped line expansion can be approximated with any desired degree of precision by a polygon. In principle, there are two options:

The approximating polygon lies within the cigar shape: The area which is covered by the polygon is restricted for the robot. This area is smaller than the real area which is restricted by the cigar shape. That may allow the path planning to generate paths which the robot can not follow.

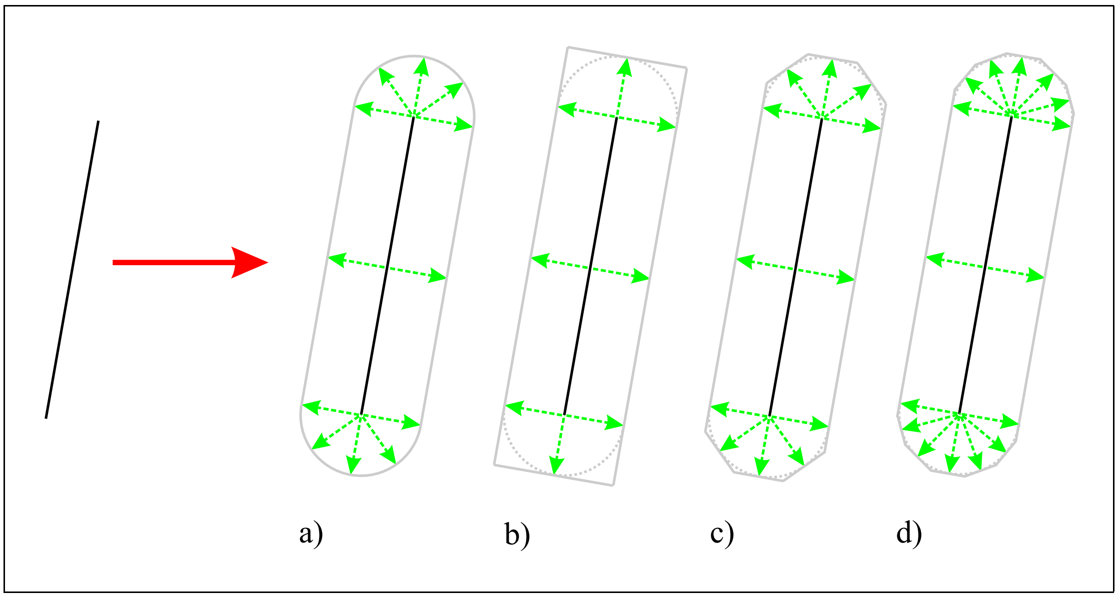


Figure 6.2: *Line expansion ("cigar shape")(a) and approximations with different degrees of precision (b-d)*

The approximating polygon lies outside the cigar shape: The area which is covered by the polygon is bigger than the area which is restricted by the cigar shape. The path planning can not find all possible paths.

For practical use, the second approach is the better choice for two reasons. First of all, safety is a key constraint, as it is important that only paths are found which the robot can follow without bumping into an obstacle. Secondly, one can conclude from **figures 6.2 b), c)** and **d)** that the loss of drivable area becomes negligible when the number of polygon lines used for the approximation is increased.

An algorithm to generate a polygon that lies outside the cigar shaped line expansion is given in **algorithm 6.1**. This algorithm must be applied to both ends of a line segment. The result is the polygon P described in terms of its vertices. With this algorithm, it is possible to expand the map with linear time effort $O(n_l)$ with n_l being the number of lines in the original map.

Figure 6.3 illustrates how the line-expansion algorithm is applied to a given line segment.

If the map contains a closed series of line segments, the cases given in **figure 6.4** and **figure 6.5** can occur. **Figure 6.4** shows an expansion of a set of lines of the original map by a radius which is small in comparison to the actual size of the line segments. A drivable area inside the contour is maintained. At first sight, this might look odd. If the object is considered to be a table, for example, in the majority of cases it is interpreted as an obstacle for a mobile robot with no free space inside at all. The path-planning algorithm introduced later generates only those paths that

Require:

- empty polygon P
- expansion radius r
- line segment given by $\vec{p}_{start}$ (or $\vec{p}_{end}$) and a normalized vector $\vec{dir}_{line}$ which indicates the direction from $\vec{p}_{end}$ to $\vec{p}_{start}$ (or from $\vec{p}_{start}$ to $\vec{p}_{end}$)
  [Depending on the side of the line segment which is created, $\vec{dir}_{line}$ goes from start to end or the other way around.]
- number of lines n_l to be used for the approximation of each semicircle

```
begin
  calculate $\alpha$: $\alpha = (\frac{\pi}{n_l} + 1)$
  $\vec{dir}_p$ = vector perpendicular to $\vec{dir}_{line}$
  for $i = 1$ to $n_l$ do
    $\vec{p} = \vec{p}_{end} + (\vec{dir}_p \cdot r)$   (or $\vec{p} = \vec{p}_{start} + (\vec{dir}_p \cdot r)$)
    $\vec{dir}_{p\prime}$ = rotate $\vec{p}$ counterclockwise by angle $\alpha$
    $\vec{p\prime} = \vec{p}_{end} + (\vec{dir}_{p\prime} \cdot r)$   (or $\vec{p\prime} = \vec{p}_{start} + (\vec{dir}_{p\prime} \cdot r)$)
    calculate the line $l$ going through $\vec{p}$ perpendicular to $\vec{dir}_p$
    calculate the line $l\prime$ going through $\vec{p\prime}$ perpendicular to $\vec{dir}_{p\prime}$
    add intercepting point $\vec{p}_i$ of line $l$ and $l\prime$ to the polygon $P$
    $\vec{dir}_p = \vec{dir}_{p\prime}$
  end for
end;
```

Algorithm 6.1: *Pseudo-code for the line expansion*

do not cross any lines. That means that only those paths are produced which avoid obstacles. Therefore a path into an obstacle is never generated. Taking the example with the table, the mobile robot is also allowed to plan a path on the table. The path planning guarantees that the mobile platform will never fall off the table. If the closed series of line segments represents a room (an office for example) the usefulness of this approach becomes clear. Because the robot must be able to operate in a room, it makes no sense to restrict this closed line structure. Considering the fact that doors or gaps which are too small for the mobile robot lead to restricted areas due to line expansion, one can assume that this example is in no way unrealistic.

If it is necessary to introduce blocked areas in a map, this can be done rather easily by just adding appropriate lines to the map. If the object's lines are expanded by a relatively big radius as shown in **figure 6.5**, the resulting polygons overlap without leaving any free space.

The method of line expansion described here has considerable advantages over a method called

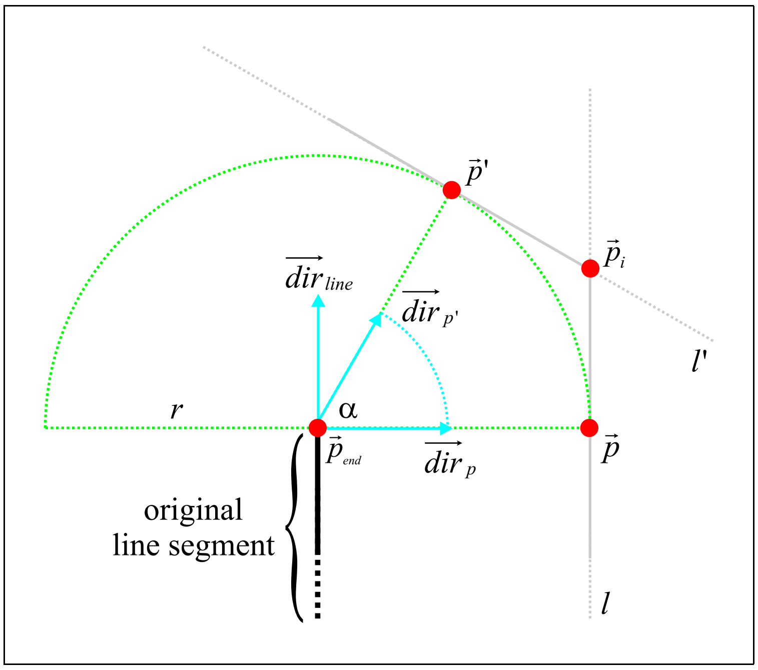


Figure 6.3: *Line expansion for one point of a given line segment (in the figure it is $\vec{p}_{end}$). The point $\vec{p}_i$ is constructed geometrically and added to the Polygon P.*

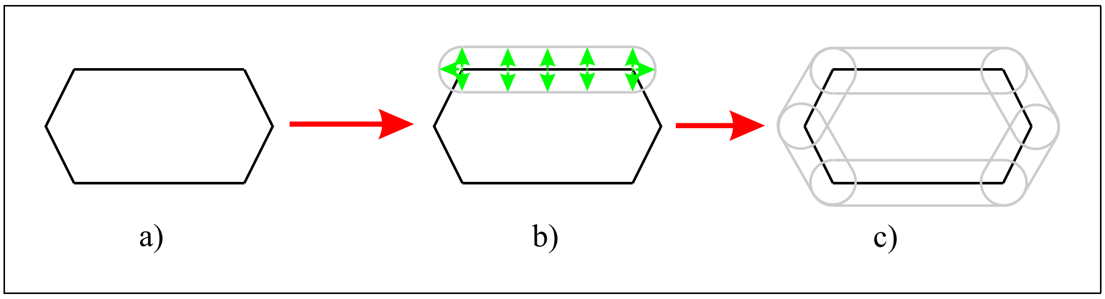


Figure 6.4: *Generation of the configuration space for a robot with a small radius: a) original map element made of line segments, b) expansion of each line segment, c) configuration map (grey) for the original map element.*

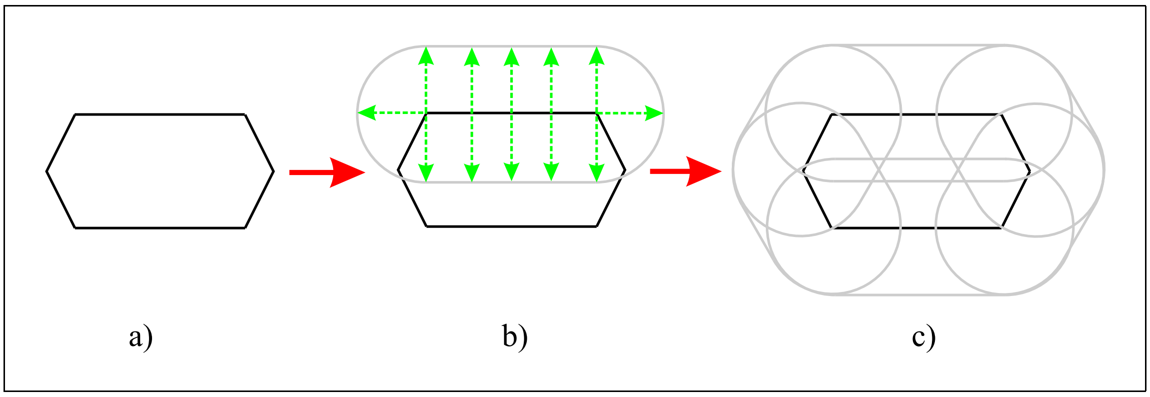


Figure 6.5: *Generation of the configuration space for a robot with a big radius: a) original map element made of line segments, b) expansion of each line segment, c) configuration map (grey) for the original map element.*

edge shifting.[8] In edge shifting, a closed series of line segments is interpreted as an object that is represented by a polygon. The expansion of this polygon is a parallel displacement of its line segments to the outside direction by a value δ. The expanded polygon has new edges at which the shifted lines intersect. This causes problems because every expansion of a line segment can only be done in relation to its neighbours. Typical problems are portrayed by the examples in **figure 6.6**. In **a)**, a part of a polygon with an indentation which is called an *eye* is shown. The edge-shifting method does not work properly because the generated edges cross each other. The line shifting in **b)** transfers a polygon into a self-overlapping polygon with a hole. In **c)**, another problem of this method becomes apparent. Sharp edges with acute angles project into free space when extended. If this method was applied, the underlying map would have to avoid these cases. For example, Hunn (1993) suggests to use maps with a lower level of detail.[9]

All these problems do not occur when line expansion of every single line segment is used. This method is computational less expensive and no constraints must be kept in view.

Another approach using *polygon triangulation*[10] can eliminate the disadvantages of edge shifting. Here polygons are divided into triangles. Then the triangles are expanded with the edge-shifting algorithm. This approach is not suitable for this work because the given maps may include single lines or series of lines and are not constrained to consist of polygons only.

[8](HUNN, 1993), pp. 18-22.

[9]Ibid., p. 19.

[10]See chapter 1 in (O'ROURKE, 1994) for detailed explanations.

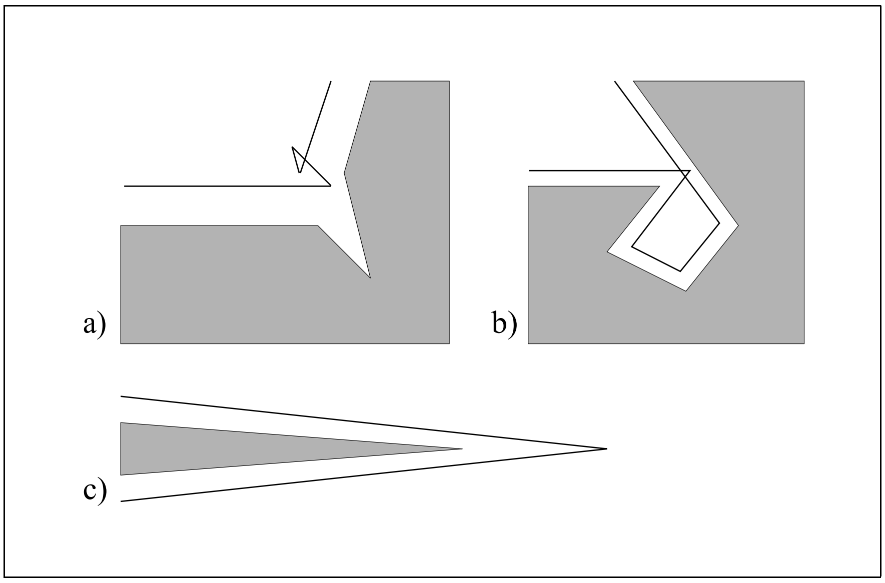


Figure 6.6: *Problems of objects expansion*

6.3 Generating paths

An exemplary situation for a path-planning problem is shown in **figure 6.7**. A mobile robot, represented by a circle with a negligible radius (the black dot in the figure), is to be moved from its start position S to a desired goal position G. The direct path is blocked by obstacles. If the conditions stated in section 6.1.5 and 6.1.3 respectively are met, the path has to be the shortest connection between start and goal, and it should be a combination of straight lines (sub-paths) as in **figure 6.10** (black line). This combination of straight lines joins the start with the goal point via a sequence of object vertices. Moving along the path always means that the robot moves from vertex to vertex until the goal point is in sight.[11]

6.3.1 Visibility graph (V-Graph)

Path finding can be performed on a visibility graph. A visibility graph can be calculated on the basis of a set of polygons and two points, the start S and the goal point G. It is a graph whose nodes correspond to vertices of the polygons and to S or G. It's edges correlate to the straight lines which join the vertices of the polygons in the sense that these straight lines do not cross the inside of any polygon. Descriptively, this means that any two vertices will only be connected if they can "see" each other. The V-Graph is visualized in **figure 6.8**.

[11](LOZANO-PEREZ AND WESLY, 1979), p. 561.

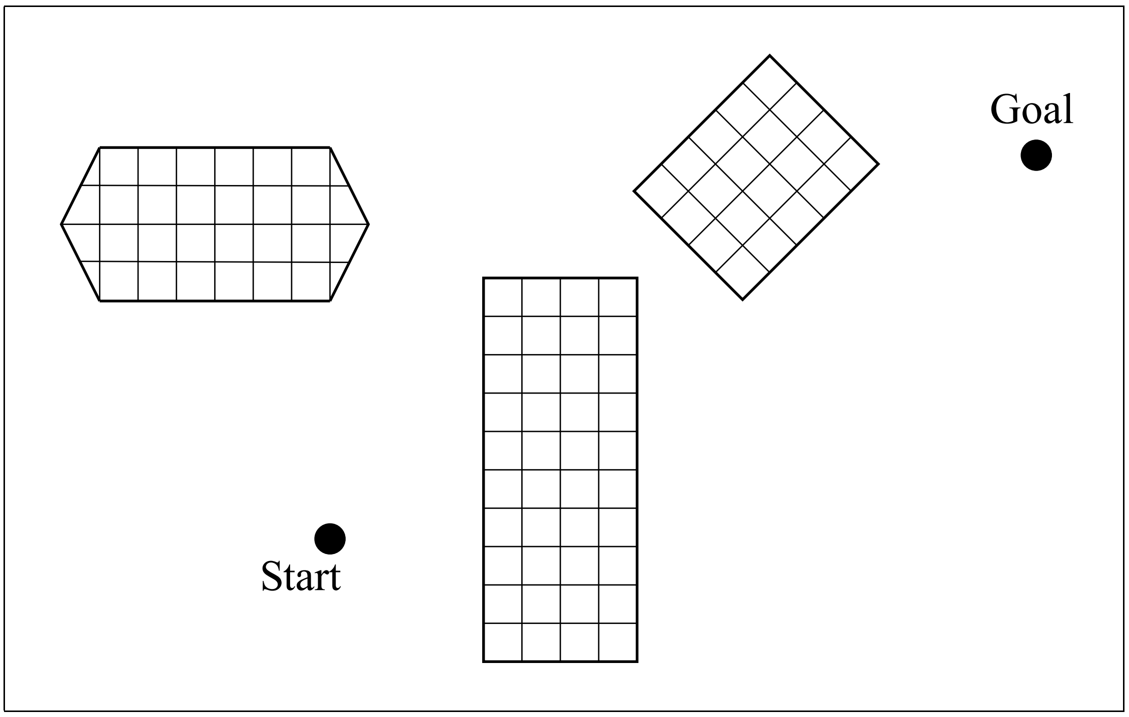


Figure 6.7: *Simple map*

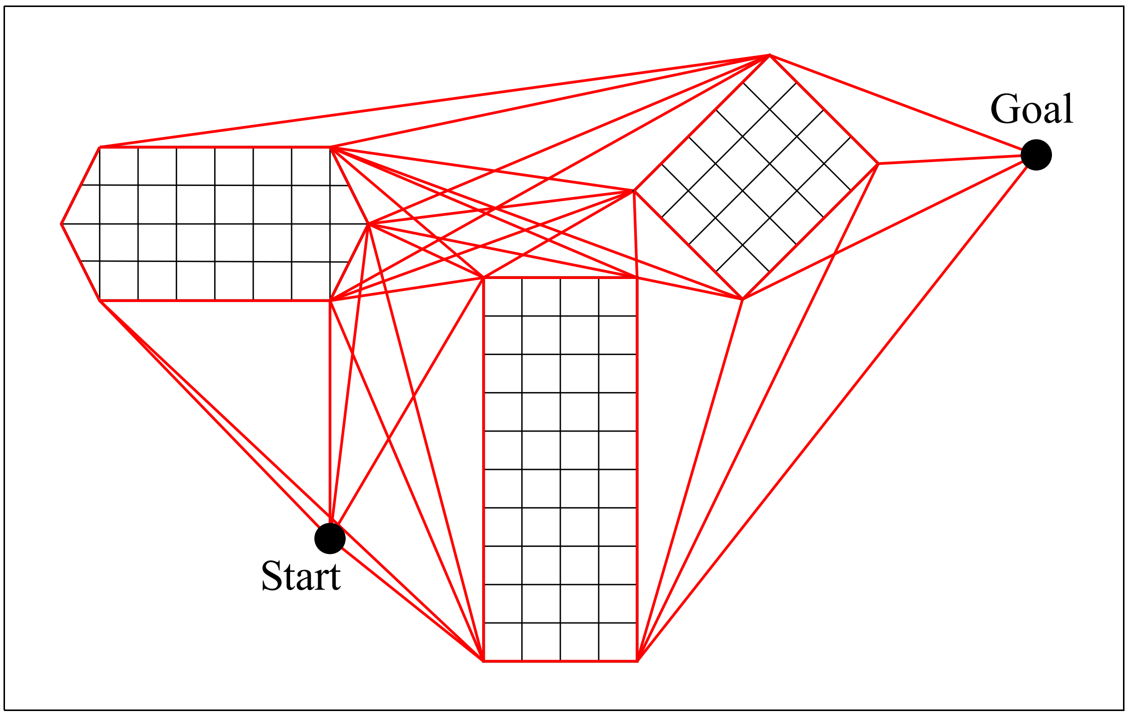


Figure 6.8: *Basic V-Graph*

The above description for an undirected visibility graph $V(N, E)$ can be put into a formal definition.[12] Let $V(N_V, E_V)$ be the V-Graph and N_V and E_V are sets, then the following conditions must be valid:

- The node set N_V is the set union of all polygon vertices V_p and the start and goal point: $N_V = V_p \cup \{S, G\}$
- The edge set E_V is the set of pairs (n_i, n_j) where n_i and n_j are elements of N_V and the edge from n_i to n_j does not overlap any polygon $\forall i \neq j$.

The visibility graph gains its simplicity from the very fact that the object to be moved (for example a mobile robot) has no dimensions, i.e. it is only a point. $V(N_V, E_V)$ includes all necessary lines to find the shortest path (the red lines in **figure 6.8**). Now a suitable search algorithm can be applied to find that optimal path. Examples for such an algorithm is the Dijkstra or A^* algorithm (see section 6.4.1).

6.3.2 Tangent graph (T-Graph)

After the visibility graph has been calculated on the basis of the set of polygons, a lot of lines are left that are not part of the shortest path. A question that can be formulated at this point is how the set of lines can be further reduced to minimize the effort for the search algorithm. An instrument to achieve this is the tangent graph (T-Graph). It was first proposed by Liu and Arimoto (1991).[13] The tangent graph is a subset of the visibility graph in section 6.3.1, but its nodes correspond only to those vertices of the participating polygons that are tangent points. The expression ”tangent” in this context deserves a specific definition.

Tangent: If the elongation of a line segment with the endpoints n_i and n_j, which are either a vertex of a polygon or S or G, does not intersect with any small polygon region near point n_i and n_j, this line is said to be a tangent, and n_i and n_j are tangent points to their respective polygons.

If the visibility graph in **figure 6.8** is reduced by those lines that are not tangents, the result is a tangent graph as in **figure 6.9**. It has to be noted in particular that vertices that build a concave corner in a polygon do not belong to the tangent graph.

A formal definition of the T-Graph can be given. Let $T(N_T, E_T)$ be the T-Graph and N_T and E_T are sets, then the following conditions must be valid:

[12](LOZANO-PEREZ AND WESLY, 1979), pp. 561-562.
[13](LIU AND ARIMOTO, 1991).

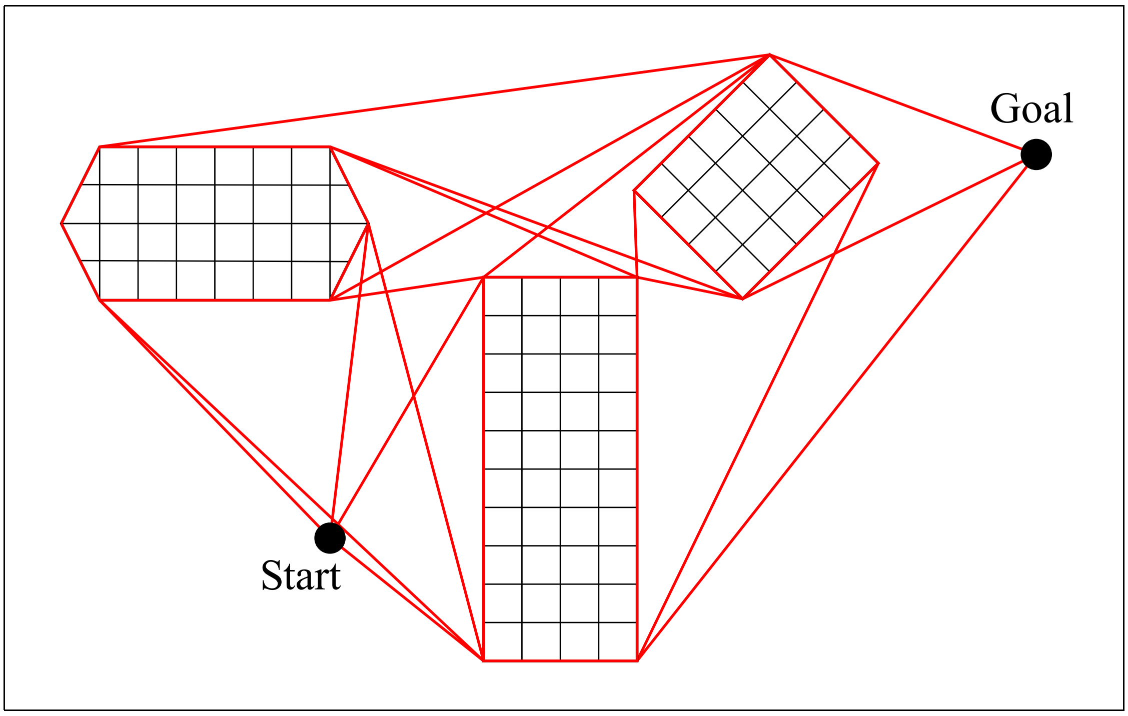


Figure 6.9: *Basic T-Graph*

- The node set N_T is the set union of all *convex* polygon vertices V_p^{convex} and the start and goal point:
  $N_T = V_p^{convex} \cup \{S, G\}$ or $N_T = \{p | p \in V_p \wedge p \text{ is convex}\} \cup \{S, G\}$
- The edge set E_T consists of pairs (n_i, n_j) whose edge from n_i to n_j is a member of the corresponding V-Graph and a tangent.

In other words, the T-Graph contains only those straight lines whose elongations at both ends do not intersect with the involved polygons.

6.4 Finding the optimal path

For the problem of finding the optimal path, costs are assigned for each possible path from start to goal. In this work, every sub-path's length is taken to be its cost. Then the cost of moving from the start to the goal point is the total sum of the associated sub-path costs. The optimality criterion is to find the shortest path. Thus, finding the optimal path means to find the path with the lowest costs.

Beyond that, other costs are conceivable in real-world path planning. Firstly, it could be expensive for a mobile robot to turn in the range of big angles because it might cost time. In that case, a path which might be a bit longer but less time-consuming could be a better choice. Another approach to formulate costs for path planning is to choose the ”safest” path instead of the shortest. The safest path could be the one with the biggest distance between the robot and the obstacles. The cost for a path segment could be the distance to the nearest obstacle. Then the optimal path would even be the one with the biggest costs. Since the calculations for those costs are more expensive, they are not used in this work.

6.4.1 The A^* Algorithm

The T-Graph in section 6.3.2 encompasses all possible sub-paths or path elements that are necessary to build a complete path from a start point to a goal point. The search algorithm A^* is used to determine the best path, which, under the constraints stated above, is the shortest one. The foundation for the algorithm is discussed in this section.

The A^* algorithm was developed by Hart, Nilsson and Raphael (1968).[14] This algorithm has the property of finding the minimum-cost path whenever a path exists at all. This is done under some restrictions which are specified below.

From now on, the graph will be denoted by G and each node in the graph by N.

The A^* algorithm starts with the first node N_{start}, which is the initial point of the mobile robot on its progress to the goal position. Since the algorithm is iterative, it has already visited a set of nodes at the beginning of every new cycle whereas a decreasing number of nodes has remained unvisited so far. For every node N that has been visited, there are some connecting paths to the start node N_{start}. But only one of those is the cheapest at any time and that one is stored by the algorithm as the best path from the start to the current node N. For the whole set of visited nodes N, the algorithm builds a spanning tree T, which contains associations between nodes N and their respective parent nodes.

The cost of path planning has only been regarded as the total length of the resulting path so far. But as already mentioned, the A^* algorithm is iterative and the real total cost can only be known at the end of the whole search process. The question arises, how the algorithm deals with the fact that it only knows costs of sub-paths and why it nevertheless finds the shortest (cheapest) path. The answer is that the A^* algorithm combines two procedures to form an evaluation (cost) function. The first takes into account that - because of the spanning tree T - the total cost from the start node N_{start} to the current node can be calculated by following the pointers back to the origin. This function is called $g(N)$. If only those ”costs so far” were taken into consideration, the procedure would be inefficient. This is the reason why a second function $h(N)$ is used. $h(N)$

[14](HART ET AL., 1968).

is a function that estimates the costs from the current node N to the goal node N_{goal}.

In order to solve a geometric problem like path finding, it is reasonable to estimate the remaining costs from the current point to the goal point by using the Euclidean metric, i.e. the direct distance.

The combination of the two strategies mentioned above leads to the following evaluation function:

$$\boxed{\text{f(N) = g(N) + h(N)}}$$

with

$$\begin{aligned} g(N) &= \text{costs } N_{start} \rightarrow N \\ h(N) &= \text{costs } N \rightarrow N_{goal} \qquad \text{(\textit{heuristic estimate})} \end{aligned}$$

Every node N in the current T is assigned a value by the cost function $f(N)$. $f(N)$ is the estimate for the cost of the best path going from the start node N_{start} via N to the goal node N_{goal}. The value of $f(N)$ may change from cycle to cycle of the iteration.

As mentioned at the beginning of this section, one constraint must be satisfied to guarantee that the minimum-cost path is found by the A^* algorithm. It concerns the heuristic estimate $h(N)$.

The h function must not overestimate the cost to reach the goal state!

$$\forall N \in G : 0 \leq h(N) \leq h^*(N)$$

The real, but yet unknown costs of the path from the current node N to the goal node N_{goal} are referred to as $h^*(N)$, and h is called *admissible heuristic*. One could say that admissible heuristics are optimistic. Since h is admissible heuristic and g represents the already known costs from N_{start} to N, f has to be admissible heuristic as well.

A very simple and apparently underestimating choice for the h function is to set it to 0 ($h(N) = 0, \forall N$). This heuristic function is then referred to as *non-informed*. This version is known as *Dijkstra's algorithm*. For the A^* algorithm, used in path search, the Euclidean metric mentioned earlier is an appropriate choice for $h(N)$. The informed search algorithm best-first search (A^*) is given priority over breadth-first search (*Dijkstra*) in this implementation.

Mathematical proofs for the optimality and completeness of the A^* algorithm can be found in Russel and Norvig (1995).[15]

[15](RUSSEL AND NORVIG, 1995), pp. 96-101.

Based on Latombe (1996),[16] an iterative version of the A^* algorithm is described below.

The following implementation of the A^* algorithm uses a list called OPEN. The OPEN list contains nodes of G sorted by their f values. This list must support the following five operations:

FIRST(OPEN) This operator determines the node with the smallest f value of OPEN and removes it from the list.

INSERT(N, OPEN) The node N is inserted into the list.

DELETE(N, OPEN) The node N is removed from the list.

MEMBER(N, OPEN) The MEMBER operation returns TRUE if N is in the list, otherwise it returns FALSE.

EMPTY(OPEN) This operator returns TRUE if the OPEN list is empty and FALSE in all other cases.

The algorithm in pseudo-code is shown in **algorithm 6.2**.[17] The parameter list of the algorithm takes five arguments. The first argument is the graph G with all nodes N marked as *unvisited*. Apart from the start and goal node, the heuristic estimate h is specified. The last argument c in the parameter list of the A^* algorithm is a cost function which defines the cost for an arc from node N to node N' in G. Therefore, c is the following mapping:

$$c : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}^+$$

where $\mathbb{N}$ is the set of nodes N. The costs are always positive.

The inner **for loop** in the flowchart of **algorithm 6.2** is called the *expansion* of N. The **else if** part deals with the case that the expanded node N' has already been visited. In this case, N' is removed from OPEN and inserted again. This causes the parent pointer of N' to be set to the best predecessor. If the current path is ”cheaper” than the last one, the pointer is set to the current parent N.

6.4.2 Resulting path

The application of the A^* algorithm to the T-Graph in **figure 6.9** delivers the shortest path for the example map in **figure 6.7**. This resulting path, which is coloured in black, is shown in **figure 6.10**.

[16](LATOMBE, 1996), pp. 604-608.
[17]Ibid., p. 606.

```
procedure $A^*$($G$, $N_{start}$, $N_{goal}$, $h(N)$, $c$)
begin
  install $N_{start}$ into $T$;
  INSERT($N_{start}$, OPEN); mark $N_{start}$ visited;
  while ($\neg$ EMPTY(OPEN)) do
    $N \leftarrow$ FIRST(OPEN);
    if ($N = N_{goal}$) then
      exit while-loop;
    end if
    for (every node $N'$ adjacent to $N$ in $G$) do
      if ($N'$ is not visited) then
        add $N'$ to $T$ with a pointer toward $N$;
        INSERT($N'$, OPEN); mark $N'$ visited;
      else if ($g(N') > g(N) + c(N, N')$) then
        modify $T$ by redirecting the pointer of $N'$ toward $N$;
        if (MEMBER($N'$, OPEN)) then
          DELETE($N'$, OPEN);
        end if
        INSERT($N'$, OPEN);
      end if
    end for
  end while
  if ($\neg$ EMPTY(OPEN)) then
    return the constructed path by tracing the pointers in $T$ from $N_{goal}$ back to $N_{start}$;
  else
    return failure;
  end if
end;
```

Algorithm 6.2: *Pseudo-code for the A^* algorithm*

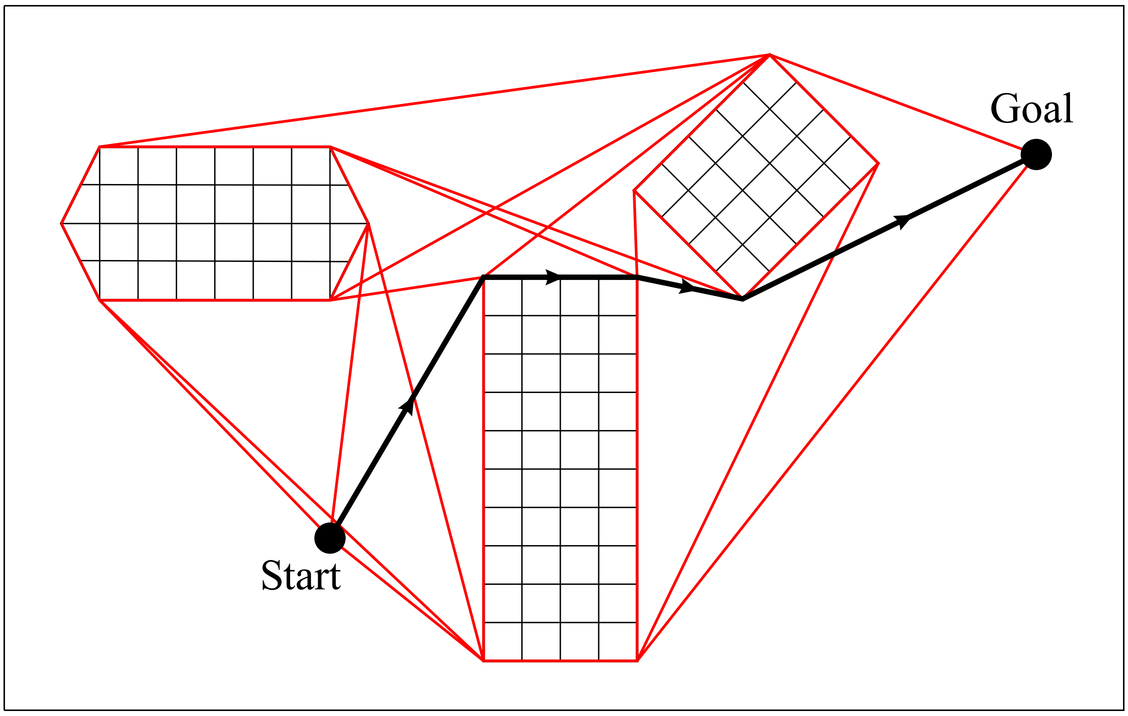


Figure 6.10: *T-Graph with path*

If the robot is assumed to be circular-shaped, the map is expanded by the robot's radius r_{Robot}, as suggested in section 6.2.1. This can lead to an overlap between objects so that gaps which are impassable for the robot are closed.

The outcome of the expansion procedure is displayed in **figure 6.11**.

Due to the overlap, the impassable gap between the two objects on the right-hand side is closed. The plotted path is the result of the A^* algorithm's application to the changed T-Graph.

If one compares **figures 6.10** and **6.11** it becomes clear that the usage of map expansion produces passable tracks for the mobile robot. It is obvious that the example robot in **figure 6.11** with the radius r_{Robot} would not be able to follow the path in **figure 6.10**.

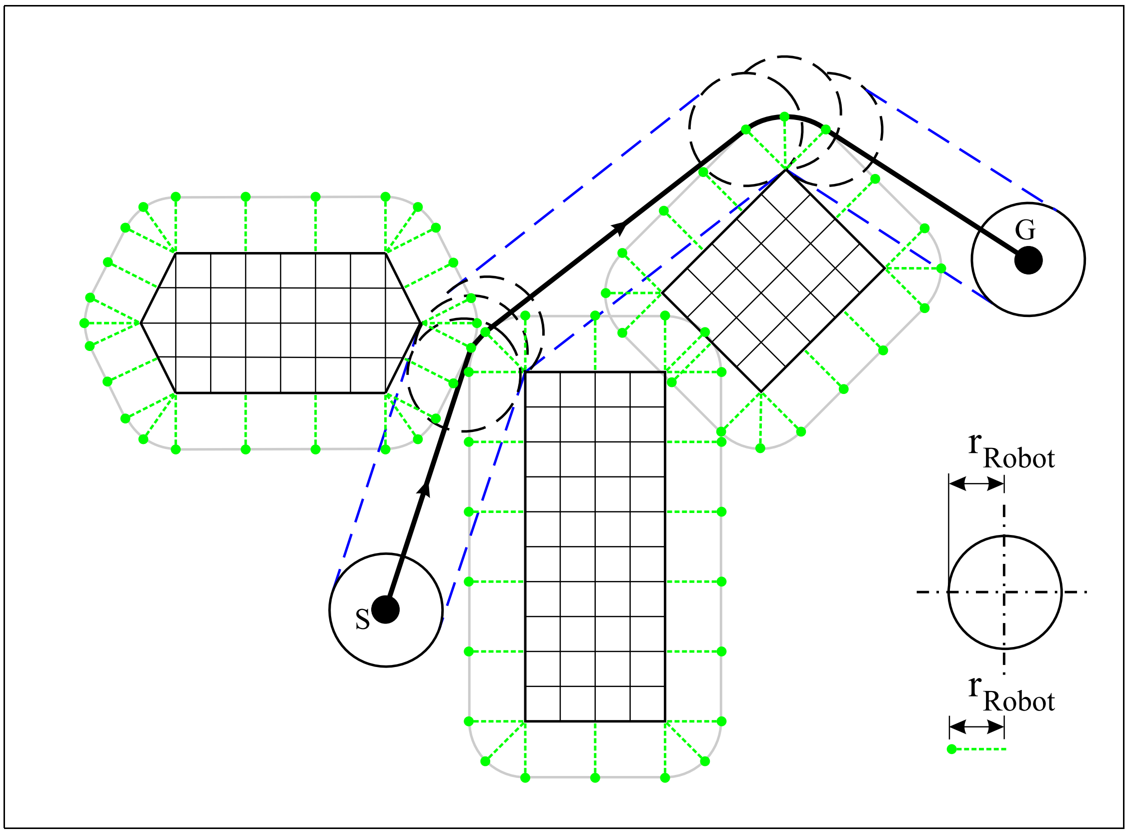


Figure 6.11: *Expanded map with path*

6.5 Software implementation

The introduced algorithms for path planning have been implemented as a *C++* software library. The library includes eleven classes. They can be used for planning a path for the *GenBase II* platform.

A map of the robot's environment must be provided before a path can be planned. Therefore, a file has to be created which includes all the line segments the robot must not cross. As an example, the file with the map data of the cell culture laboratory is shown in **figure 6.12**. A drawn map can be seen in **figure 7.2**. The character '#' is placed in front of comments. The software ignores these comments as well as empty lines. Each line of the file contains the parameters for one line segment of the map. A line is defined by the coordinates of its start and end point.

After a map file has been created, the library can be used. The hierarchy of the classes is shown in **figure 6.13**. The inheritances are indicated by green arrows. The relations are marked by red lines. The digits at the ends of the red lines indicate the number of instances of the respec-

```
#
# Map of the cell culture laboratory
# at the University of Bielefeld
#
+0.000 +0.000 +3.565 +0.000
+3.565 +0.000 +3.565 +5.140
+3.565 +5.140 +2.880 +5.140
+2.880 +5.140 +2.880 +5.515
+2.880 +5.515 +3.565 +5.515
+3.565 +5.515 +3.565 +10.625
+3.565 +10.625 +2.730 +10.625
.
.
.
```

Figure 6.12: *The figure shows the first part of the file with the map data of the cell culture laboratory.*

tive classes. The graphical representation of the classes in **figure 6.13** is reduced to the public methods and the most important attributes.

An instance of the class *Pathplanner* has to be created to start path planning process. Its constructor is called with the path to the file which contains the description of the map. This file is used to create an object of the class *Map* which serves as a basis for generating the configuration space. It is represented by an instance of the class *Maze* and it is created using the line expansion algorithm of section 6.2. All expanded lines of the configuration space are objects of the type *Polygon*.

The class *Pathfinder* searches for a path from a given start to a given goal point using the A^* algorithm on a tangent graph. The search process is started automatically when *getPath(...)* of the class *Pathplanner* is called. *GetPath(...)* takes the start and goal point of the path as arguments. The return vector of this method contains the start, the goal and all computed intermediate points.

Most of the classes of **figure 6.13** are used for internal computations. They are not explained in detail here. Furthermore, some methods are not shown in the figure because they were only used for visualization purposes. For example, the class *Pathplanner* has methods to provide the representations of the map, the configuration space and the path. These methods were used to create **figure 7.2**.

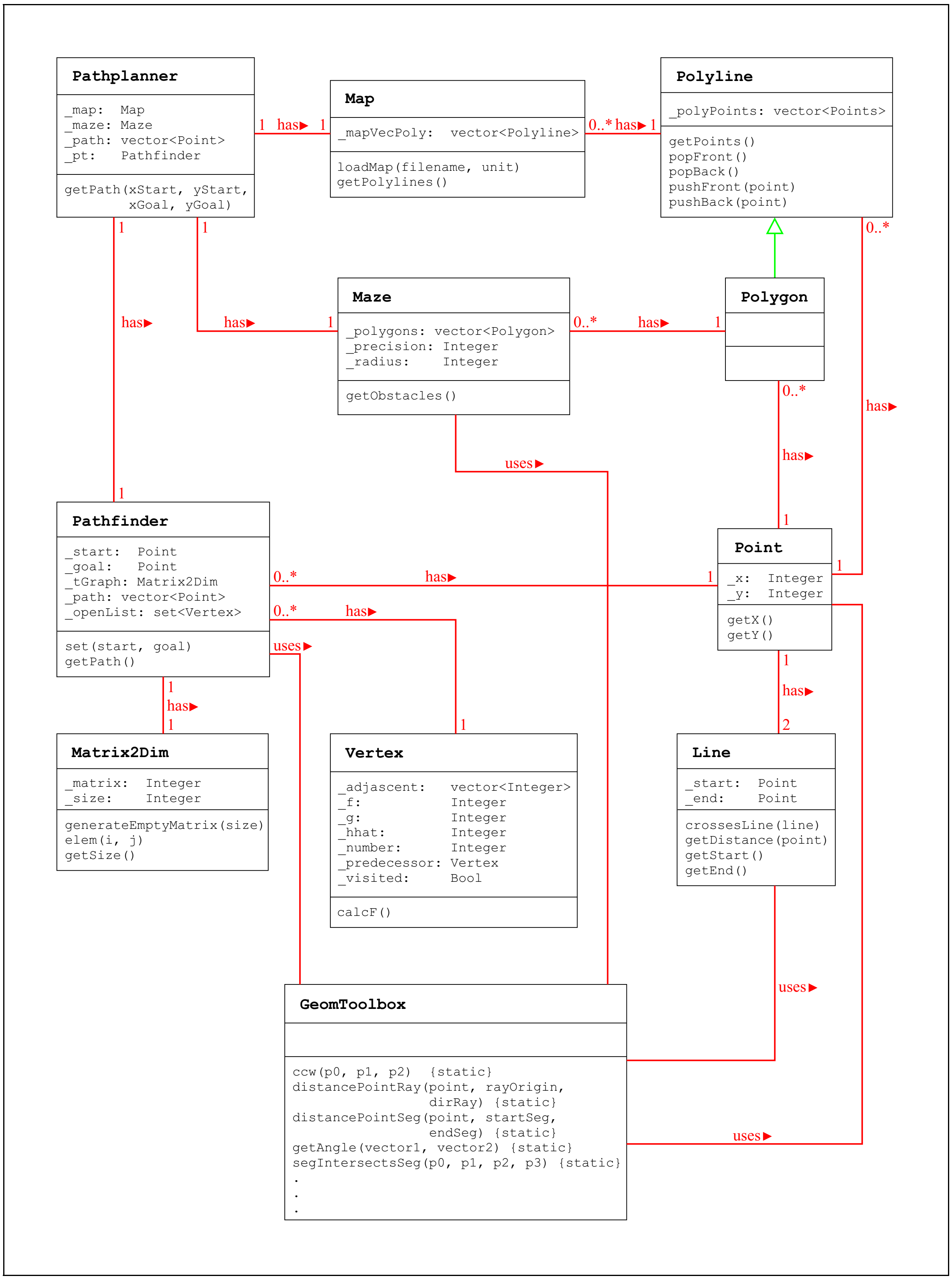


Figure 6.13: *The figure contains the flowchart with the hierarchy of the path planning classes. The inheritances are indicated by green arrows. The relations are marked by red lines. The digits at the ends of the red lines represent the number of instances of the respective classes. The most important attributes and the public methods are specified.*

Chapter 7

Experimental Results

In the previous chapters, different methods for the navigation of the mobile platform *GenBase II* have been introduced. The qualities of these methods are analyzed in this chapter. For this purpose, two experiments were made. Each experiment was done once with the original software *genControl* which was delivered by the manufacturer *genRob*, and once with the software developed in this work (*UniBiControl*). This allows the comparison of the proposed methods with commercial software which represents the current state of the art in the field of navigation of an autonomous mobile robot.

The first experiment investigates the quality of the control system that was introduced in chapter 5 as well as the accuracy of the self-localisation using the mechanisms proposed in chapter 4. The task is to position the robot platform at a predefined location. The errors of the self-localisation and control system are analyzed. The second experiment is concerned with the problem of following a desired path. The *genControl* software and the control system developed in this work use the same strategy which is a motion that is divided into three sub-moves. Firstly, the robot turns to the goal point. Secondly, it tries to move straight to the goal. Finally, it turns into the desired orientation at the goal point. Since both softwares behave similarly, their performances can be compared and their different deviations from the desired path can be analyzed.

Both experiments were done in a pilot scale cell culture laboratory which can be seen in **figure 7.1**. The map that was used is shown in **figure 7.2**. The positions of the available laser reflector marks are indicated by blue crosses.

The chapter ends with a conclusion about the experimental results and points out the advantages and disadvantages of the two softwares.

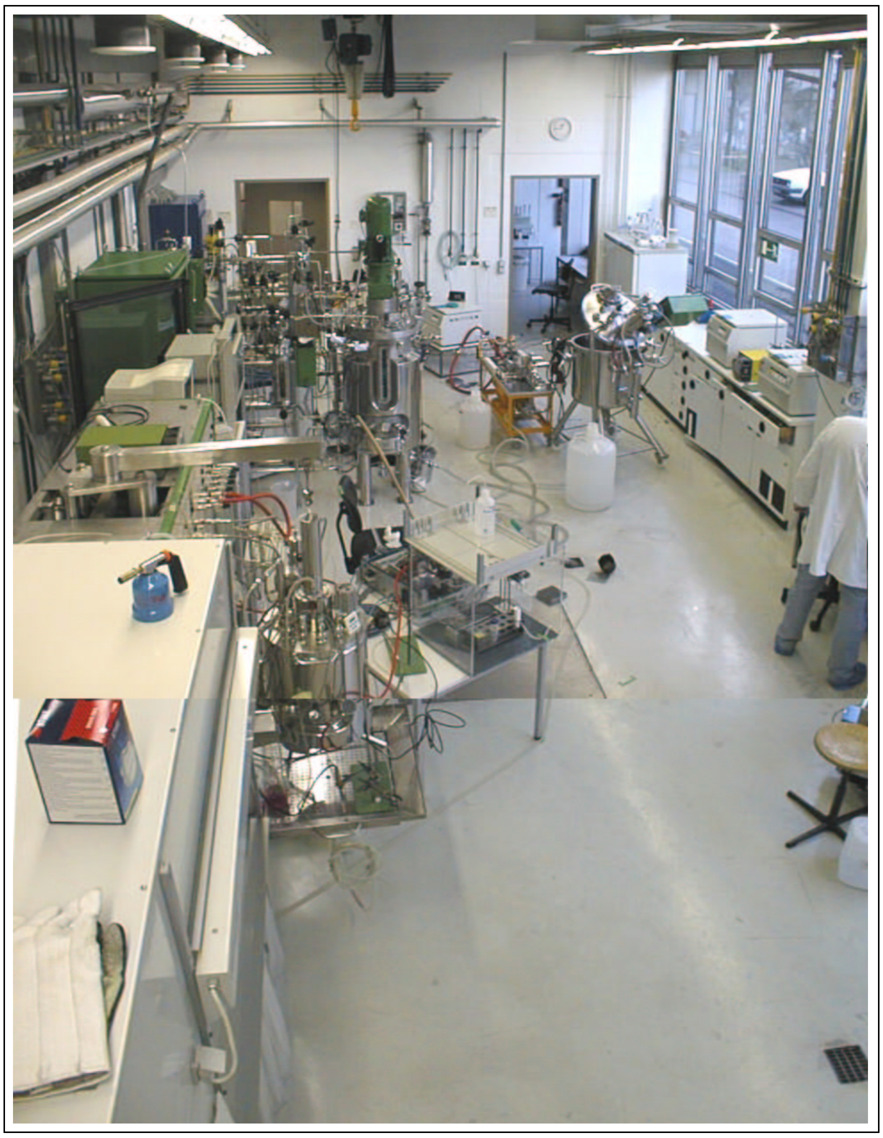
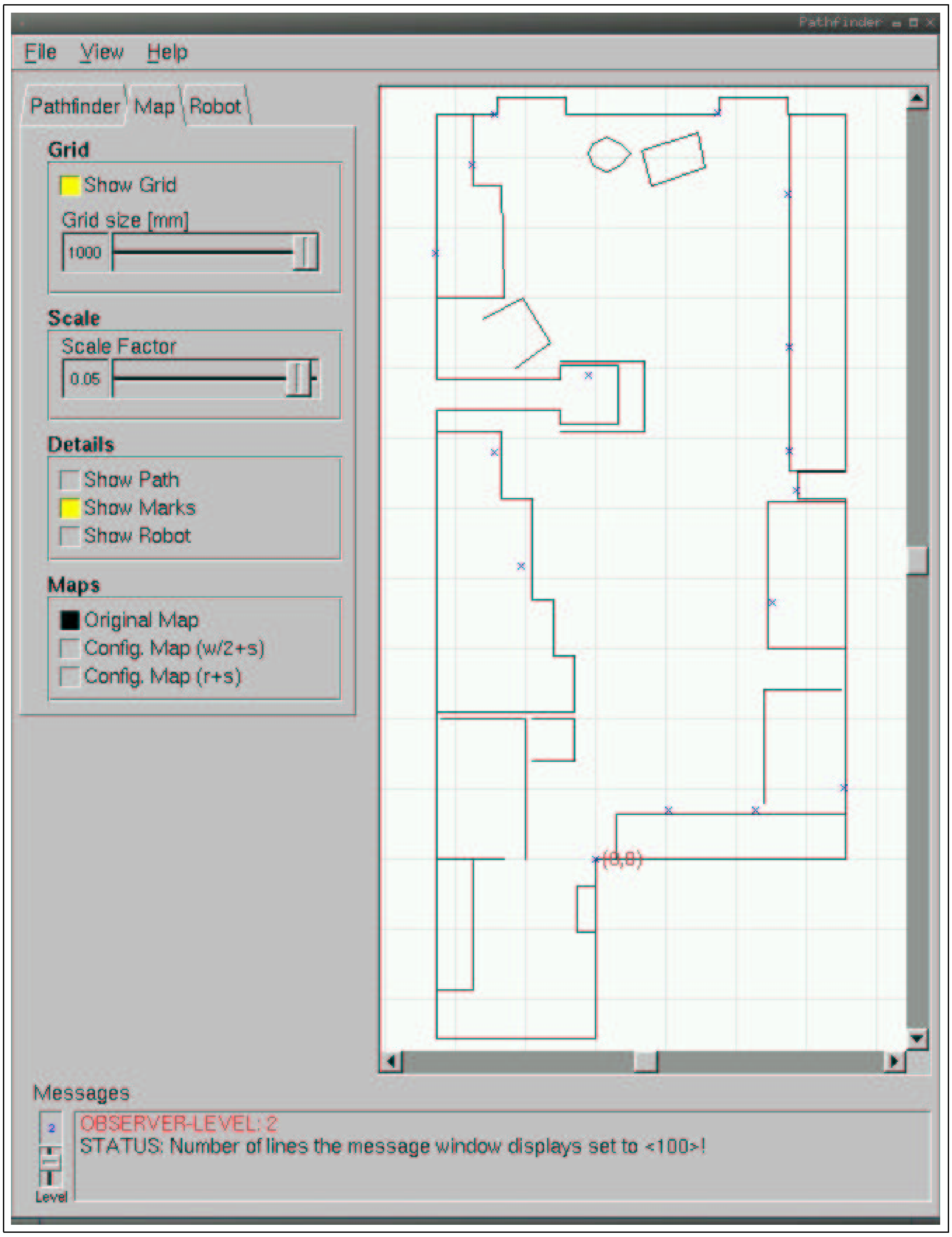


Figure 7.1: *Cell culture laboratory at the University of Bielefeld*



OBSERVER-LEVEL: 2
STATUS: Number of lines the message window displays set to <100>!

Figure 7.2: *The figure shows a screenshot of the graphical user interface for the path planning test program. On the right hand side, there is a map of the cell culture laboratory at the University of Bielefeld where the experiments took place. The map contains all obstacles and the reflector mark positions which are indicated by blue crosses.*

7.1 Experiment one: localisation and control

The first experiment analyzes the quality of the used localisation algorithms and the control system of the robot. Since the input of the control system is the position and orientation obtained from the self-localisation module, the control system reacts sensitively to all perturbations of that module.

7.1.1 Experimental setup

The experiment was done in the cell culture laboratory of the Institute of Cell Culture Technology at the University of Bielefeld. The robot was placed on a marked position in the laboratory. The coordinates of the position and the orientation of the robot were determined using the respective localisation software.

The task in the first experiment was to position the robot on a predetermined coordinate with a predetermined orientation. Therefore, two additional coordinates were calculated. The first coordinate was a point two metres in the backward direction of the goal orientation. The robot had to move to this coordinates and turn back into the original orientation. The second coordinate was an intermediate point in the middle of the line segment to the goal point. The robot had to stop at this point and had to correct its heading. The last step for the mobile robot was to move to the goal position and to take in the desired orientation.

The reason for the intermediate point was given by the manufacturer software *genControl* and the original task of the mobile robot which is the transport of probe tubes between the different laboratory devices. During the tests with the mobile robot and the improvement of the softwares it turned out that the best positioning results with the manufacturer software were received if the robot moves to the goal point from a point one metre in the backward direction of the goal orientation. Thus, the first experiment was designed to represent this situation, which reoccurs in the original scenario, namely the approach to a given goal point and orientation.

After the robot reached the goal point, the position and orientation given by the respective localisation software were stored. With the help of two rods at the left front side and the right back side of the robot, the ”real” position and orientation of the robot were measured. In this case ”real” means that the marked goal point is assumed to have the coordinates determined at the beginning of the experiment. The coordinates and orientation at the goal point were measured with respect to this ”real” position. **Figure 7.3** shows the position of the rods for the measurement. The dimensions which are needed for the calculation of the position and the orientation can be found in **figure 7.6**. The experiment was repeated 50 times with the original manufacturer setup and 50 times with the software setup of this work.

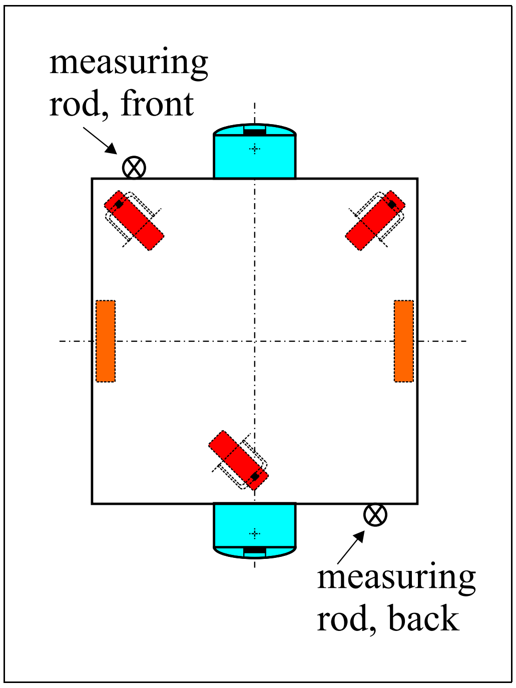


Figure 7.3: *The position of the measuring rods*

7.1.2 Analysis of the control system

The control system works on the output of the self-localisation. The goal is to move the mobile robot as near as possible to the goal point. **Figure 7.4** and **7.5** illustrate the accuracy of the robot's controller. The closer the position of the mobile unit at the end of the motion lies to $(x = 0, y = 0, \varphi = 0)$ the better the controller is.

A closer inspection of **figure 7.4** shows that the expansion of the scatter plot in x-direction is bigger than in y-direction. Since the approach to the goal position took place horizontally along the x-axis it can be seen that the controller of the *genControl* software stopped the robot clearly further before the goal point than the controller of the *UniBiControl* software. The mean deviation in x-direction amounts to $\mu_x = -8.1\ mm$ with the *genControl* software and with the *UniBiControl* software it amounts to $\mu_x = -0.7\ mm$. The mean deviation in y-direction amounts to $\mu_y = 0.4\ mm$ with *genControl* and to $\mu_y = 0.8\ mm$ with *UniBiControl*. Contemplating the standard deviation around the respective mean values, it follows a value of $\sigma_x = 7.5\ mm$ in x and $\sigma_y = 4.1\ mm$ in y-direction for the *genControl* software. As can be gathered from **figure 7.4**, the values of the *UniBiControl* software lie closer together. This is reflected by the smaller standard deviations ($\sigma_x = 2.2\ mm$, $\sigma_y = 0.8\ mm$). Furthermore, it is of some importance to consider the maximum error. A deviation from the goal point which is too large leads to a situation in which the manipulator arm, mounted on the mobile unit, cannot reach certain points anymore.

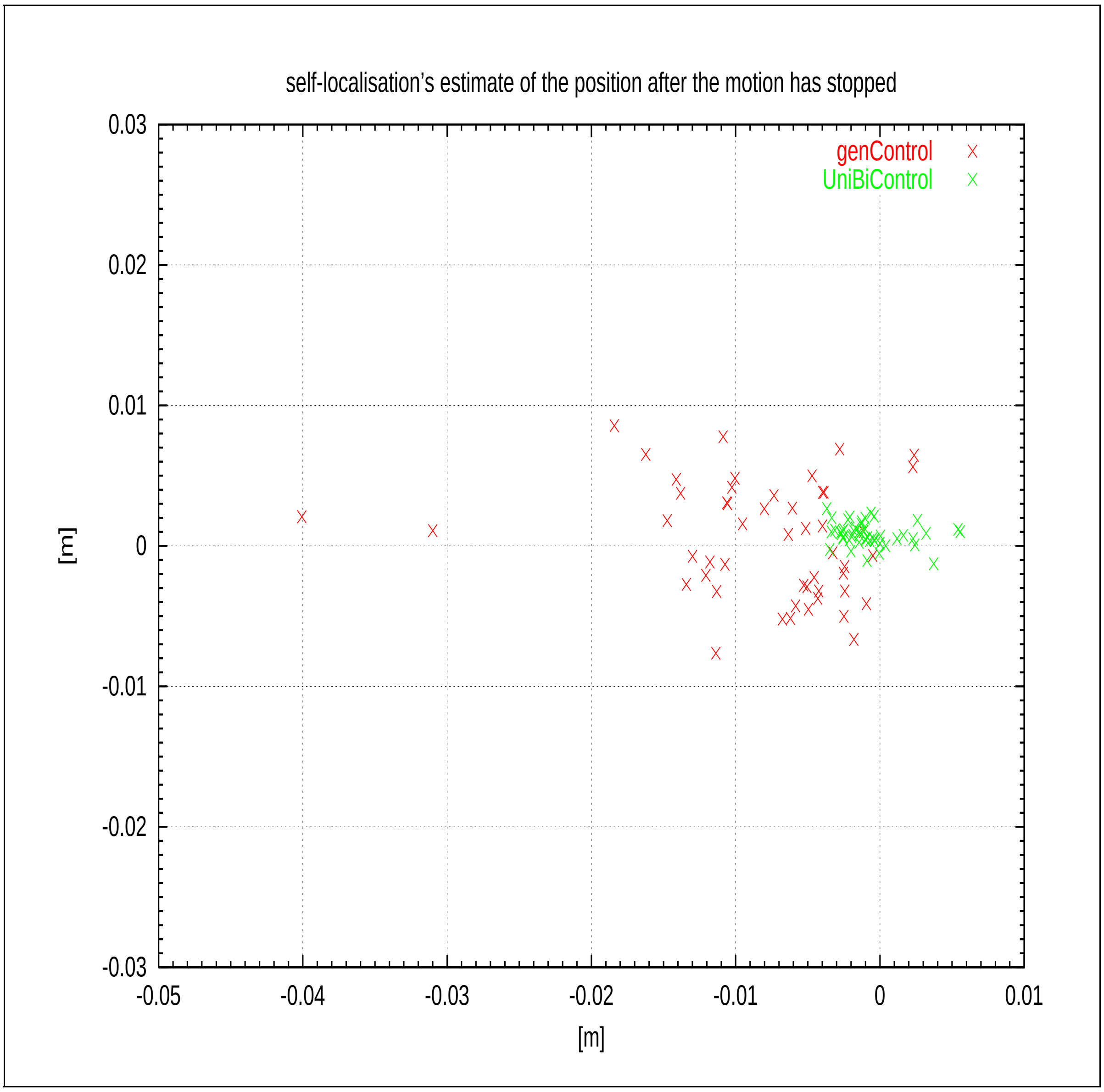


Figure 7.4: *Experiment 1 - The mobile robot had to move 50 times to the goal point* $(x = 0, y = 0, \varphi = 0)$ *with each software. The figure shows the self-localisation software's output of the position when the robot has stopped.*

The maximum positioning error for the *genControl* software is $-40.1\ mm$ in x and $8.6\ mm$ in y-direction. For the *UniBiControl* software the maximum positioning error results in $5.6\ mm$ in x-direction and $2.7\ mm$ in y-direction.

Figure 7.5 shows the orientation angle φ that the mobile platform has reached at the end of the motion. The mean value for *genControl* is $\mu_\varphi = 0.14°$ and for *UniBiControl* it is $\mu_\varphi = 0.02°$.

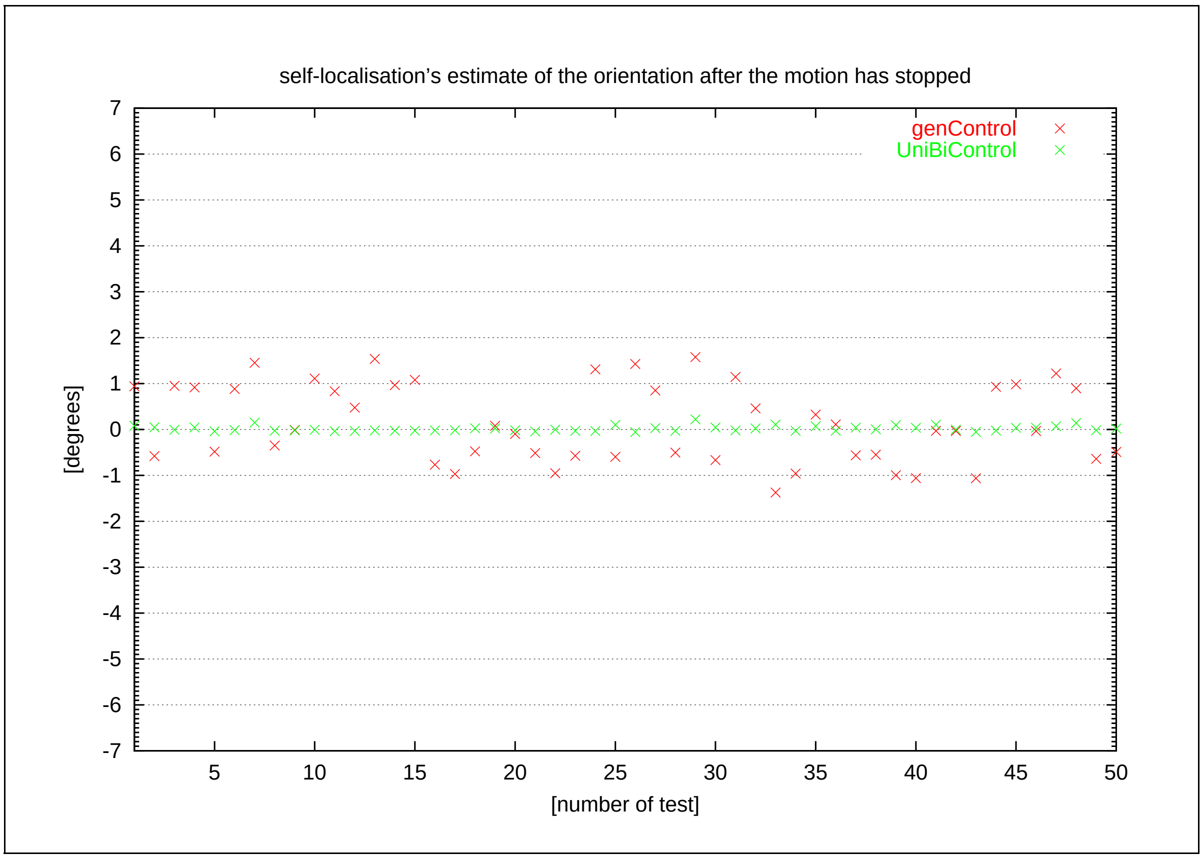


Figure 7.5: *Experiment 1 - The mobile robot had to move 50 times to the goal point* $(x = 0, y = 0, \varphi = 0)$ *with each software. The figure shows the self-localisation software's output of the orientation when the robot has stopped.*

The standard deviation σ for the *genControl* software amounts to $\sigma_\varphi = 0.86°$ and for the *UniBi-Control* software it amounts to $\sigma_\varphi = 0.06°$. The maximum error of the orientation angle φ comes to $1.57°$ for *genControl* and $0.22°$ for *UniBiControl*.

7.1.3 Analysis of the localisation

During the execution of experiment one, the position of the mobile unit was measured with the help of the two measuring rods which are mounted on the robot. For this purpose, a coordinate system was drawn onto the floor. **Figure 7.6** shows the geometrical setup which is necessary for the determination of the mobile robot's position and the orientation at the end of the motion. The positions of the two measuring rods with respect to the world coordinate system is given by the two vectors $\vec{r}_{front}$ and $\vec{r}_{back}$. Thus, the robot's position is determined by

$$\vec{p}_{robot} = \frac{1}{2}\left(\vec{r}_{front} + \vec{r}_{back}\right)$$

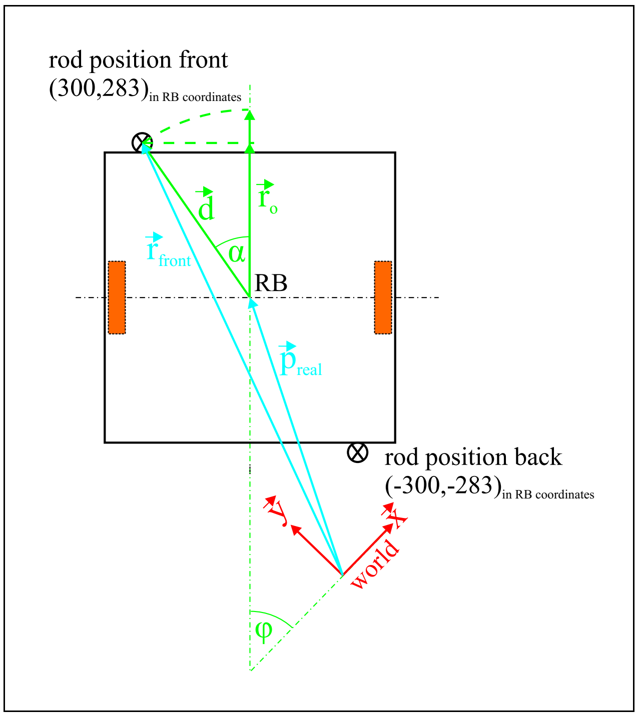


Figure 7.6: *The figure shows the basics for calculating the position of the robot's centre based on the rod positions.*

The orientation of the mobile robot is calculated as follows

$$\varphi = \arccos\left(\frac{\vec{x}\cdot\vec{r}_o}{\|\vec{x}\|\cdot\|\vec{r}_o\|}\right) = \arccos\begin{pmatrix}1\\0\end{pmatrix}\frac{\vec{r}_o}{\|\vec{r}_o\|}$$

where

$$\vec{r}_o = R\vec{d}$$

with

$$R = \begin{bmatrix}\cos\alpha & \sin\alpha\\ -\sin\alpha & \cos\alpha\end{bmatrix}$$

and

$$\vec{d} = (\vec{r}_{front} - \vec{p}_{real})$$

The angle α between $\vec{d}$ and $\vec{r}_o$ is

$$\alpha = \arctan\left(\frac{283}{300}\right) = 43.33^\circ$$

Figure 7.7 contains the robot's position and **figure 7.8** the robot's orientation which were both determined with the above method. The mean of the deviation using the *genControl* software amounts to $\mu_x = -10.1\ mm$, $\mu_y = -1.4\ mm$ and $\mu_\varphi = 1.58^\circ$.

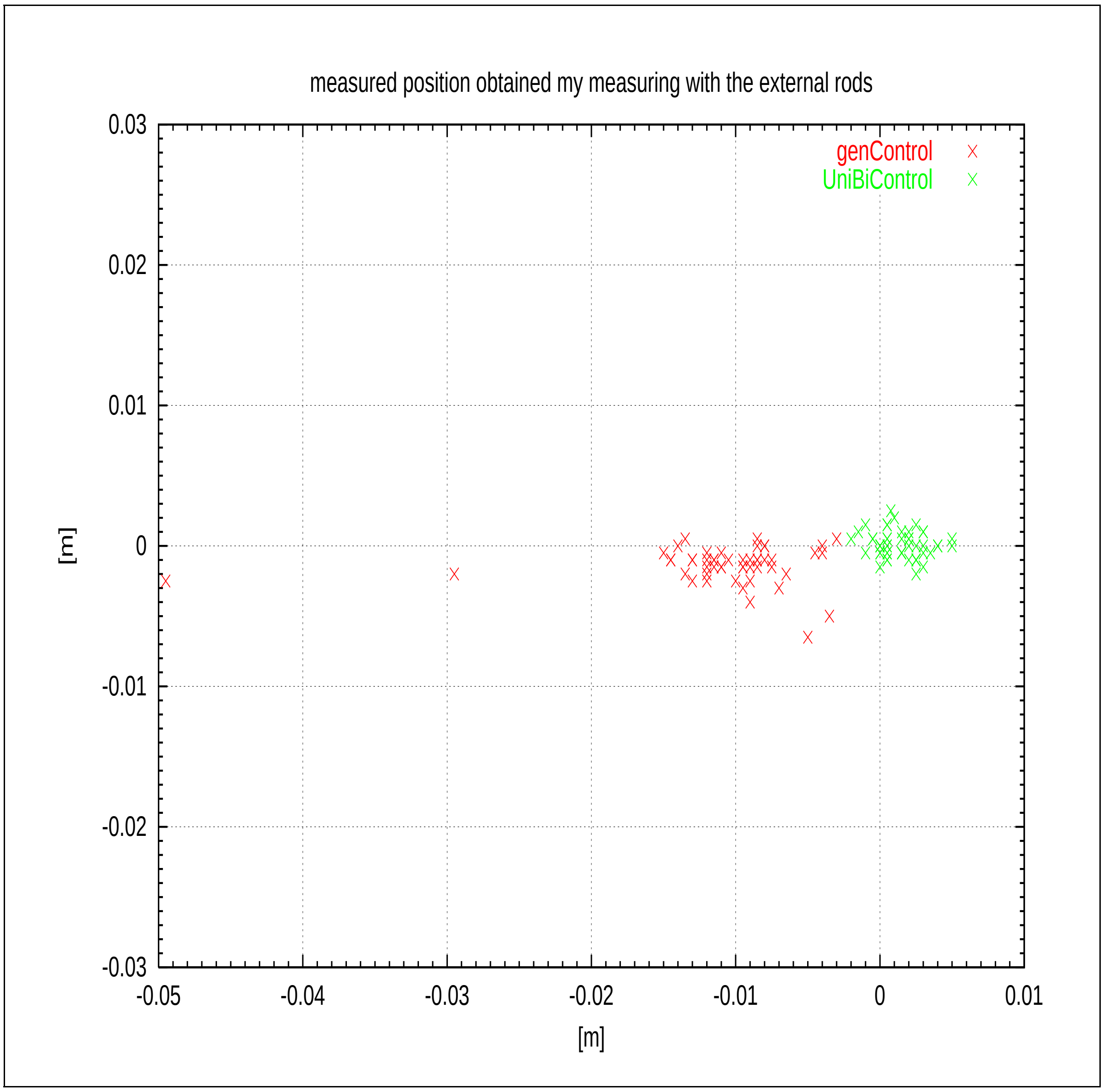


Figure 7.7: *Experiment 1 - The mobile robot had to move 50 times to the goal point* $(x = 0, y = 0, \varphi = 0)$ *with each software. The figure shows the real position which has been measured with the help of the measuring rods.*

The values for the *UniBiControl* software are $\mu_x = -1.2\ mm$, $\mu_y = 0.1\ mm$ and $\mu_\varphi = 0.23°$. The standard deviations around these means of deviation are $\sigma_x = 6.9\ mm$, $\sigma_y = 1.3\ mm$ and $\sigma_\varphi = 1.9°$ for *genControl* and $\sigma_x = 1.6\ mm$, $\sigma_y = 0.9\ mm$ and $\sigma_\varphi = 0.3°$ for *UniBiControl*. If compared with the results in section 7.1.2, the above values show only a very small deviation in the positioning of the robot. This means that both self-localisation systems deliver a good estimate of the real position of the mobile unit.

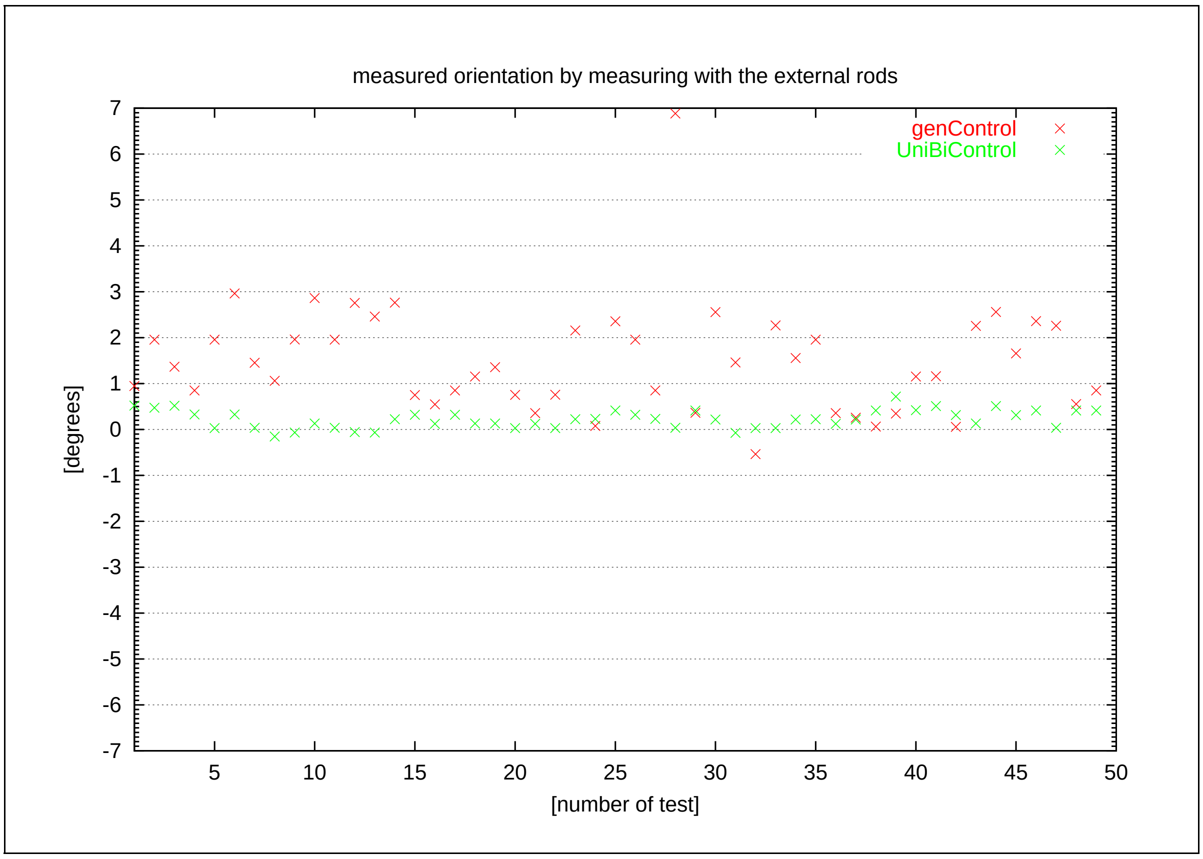


Figure 7.8: *Experiment 1 - The mobile robot had to move 50 times to the goal point* $(x = 0, y = 0, \varphi = 0)$ *with each software. The figure shows the real orientation which has been measured with the help of the measuring rods*

In this context, *genControl* shows a difference of the mean values of $2\ mm$ in x and $1.8\ mm$ in y-direction. With *UniBiControl* the differences are $0.5\ mm$ in x and $0.7\ mm$ in y-direction.

Figure 7.8 shows the real orientation which has been measured with the help of the measuring rods. It can be seen that *UniBiControl* deviates less than $1°$ from the desired orientation ($\mu_\varphi = 0.23°$, $\sigma_\varphi = 0.3°$). The *genControl* software shows a (counterclockwise) tendency of the orientation angle to positive values ($\mu_\varphi = 1.58°$, $\sigma_\varphi = 1.92°$).

It is very important that the mobile unit reaches its goal orientation with a preferably small error since already small errors of a few degrees in the orientation lead to a miss-positioning of the outstretched arm's end effector of a few centimetres.

7.2 Experiment two: accuracy of the trajectory

The second experiment was also carried out in the cell culture laboratory of the Institute of Cell Culture Technology at the University of Bielefeld. With this experiment, the trajectory of the mobile robot is compared to a straight line from the start point to the goal point. Again, the manufacturer software and setup is compared with the software and setup introduced in this work. As mentioned before, the control systems of both robot setups follow a similar strategy when the robot moves to a goal point. In both cases, the robot first turns into the direction of the goal point and then it tries to reach the goal point on a straight line. At the goal point, the robot turns into the desired goal orientation. In this experiment, the movement along the desired straight line is surveyed.

7.2.1 Experimental setup

The experiment started at a given position and orientation in the laboratory. The robot moved on a straight line to its goal point. The distance to the goal point was about $7.5\ m$. While the robot moved, the position and orientation values which were obtained from the localistaion software were stored with a frequency of about $50\ Hz$. The robots maximum velocity was limited. The experiment was repeated 40 times, each time with three different velocities. The first run was done with the maximum possible robot velocity of $0.5\ \frac{m}{s}$. The second run was done with 66% and the third run with 33% of the maximum.

7.2.2 Analysis of the trajectories

Figure 7.9 exemplarily shows a trajectory with a covered distance of $7.5\ m$ which was driven by the mobile platform *GenBase II*. The velocity in this case was $v_{max} = 0.5\ \frac{m}{s}$. It can be seen that *UniBiControl* shows no large deviation from the desired straight path. **Table 7.1** in section 7.3 contains the accurate values. Experiment two has shown that all trajectories of one software have a characteristical shape, independently from the velocity of the mobile platform. At certain spots on the path perturbations in the self-localisation occur. A possible explanation for that is that some laser reflector marks disappear from the visual field of the robot's laser range finders and others appear at the same time.

The deviation of the *genControl* software, which is visible in particular in the right hand part of **figure 7.9**, can be put down to the fact that this software does not try to follow a trajectory but to go in direction to the goal.

Figure 7.10 contains the orientation angle φ during the movement of the mobile platform. The *UniBiControl* software shows an angular deviation of up to one degree. The explanation for that

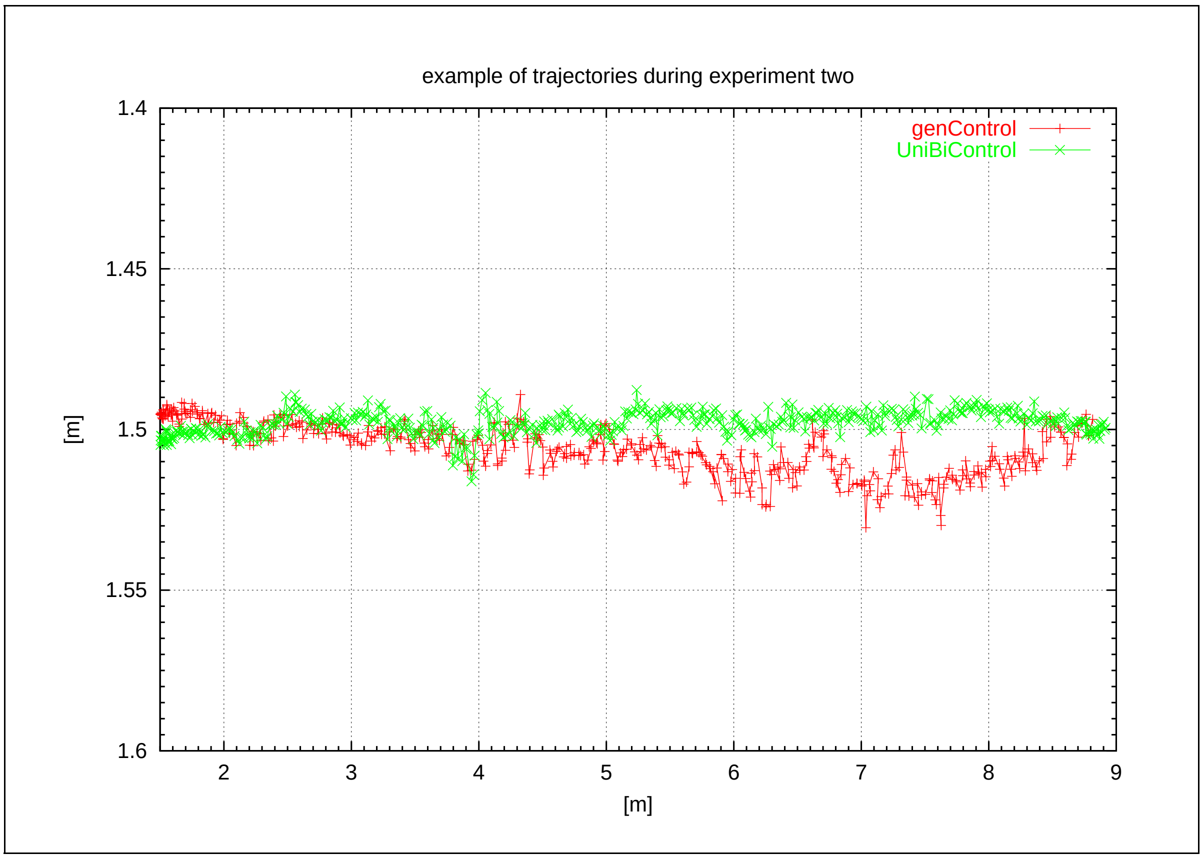


Figure 7.9: *Experiment 2 - example of a trajectory driven by the mobile robot with genControl and UniBiControl (original coordinate system is turned 90 degrees clockwise).*

behaviour is that the software always tries to get back onto the trajectory.

In the context of the analysis of the trajectories in experiment two, the kurtosis of the deviations from the desired trajectory was computed. The kurtosis specifies how peaked or flat a distribution is compared to a normal distribution. A value of zero is equivalent to a normal distribution. A larger value signalizes a peaked distribution and *vice versa*. The values which result from the *genControl* software lie near a normal distribution for all velocities, whereas the values which result from the *UniBiControl* software have a larger kurtosis the smaller the velocities get. This means that the *UniBiControl* software can follow the trajectory better if the velocity is smaller.

7.3 Summary of the results

The *UniBiControl* software represents the implementation of the self-localisation and control mechanisms which have been introduced in this work. The experiments have revealed that this

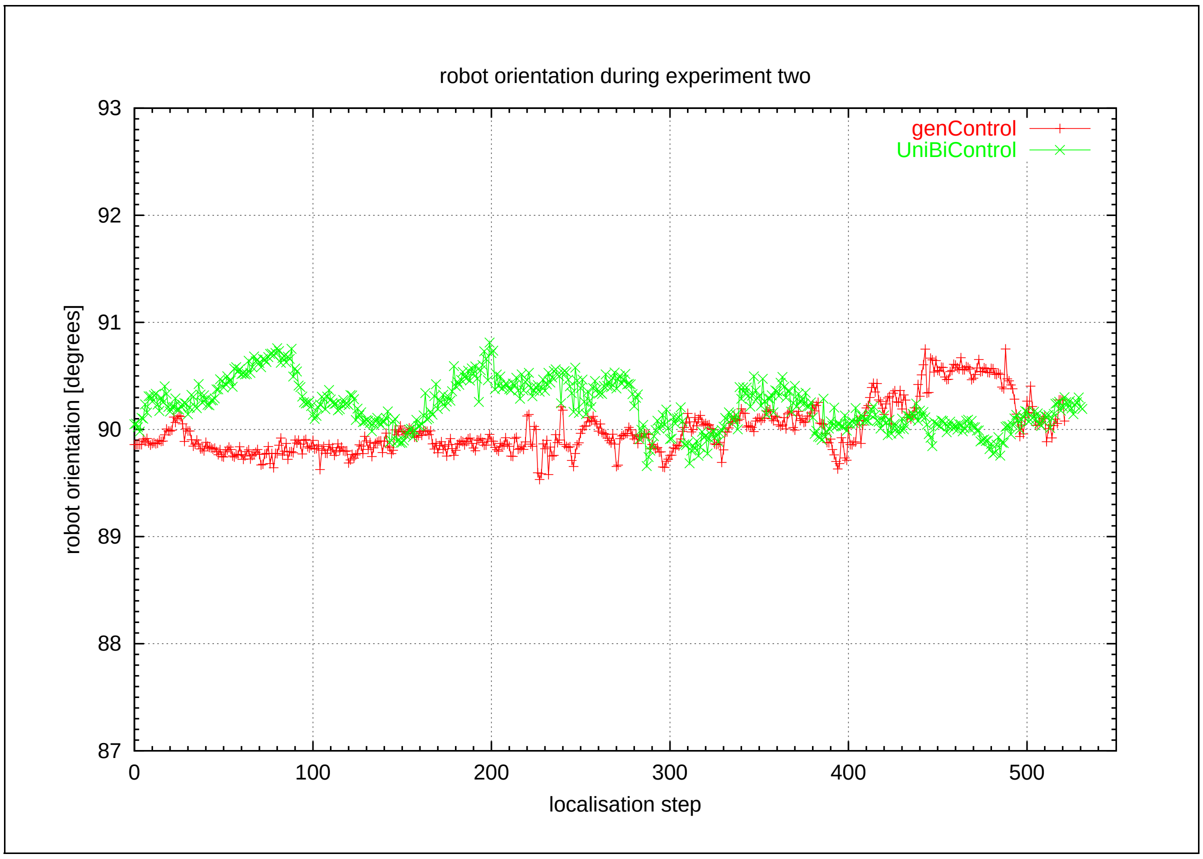


Figure 7.10: *Experiment 2 - the robot's orientation during the trajectories shown in figure 7.9.*

software provides a sufficient precision to fulfill the described transportation task in the cell culture laboratory. In comparison to the *genControl* software, which was provided in delivery condition of the mobile platform *GenBase II*, the obtained results regarding the precision are even better. This can be seen in **table 7.1** where all results are assembled. It can be expected that the transportation task benefits from the higher precision of the *UniBiControl* software.

Results of the experiments			
deviation of the self-localisation software's position/orientation from the goal position/orientation			
		genControl	*UniBiControl*
mean μ of the deviation from the goal	x	-8.1 mm	-0.7 mm
	y	0.4 mm	0.8 mm
	φ	0.14 °	0.02 °
standard deviation σ of μ	x	7.5 mm	2.2 mm
	y	4.1 mm	0.8 mm
	φ	0.86 °	0.06 °
maximum deviation from the goal	x	-40.1 mm	5.6 mm
	y	8.6 mm	2.7 mm
	φ	1.57 °	0.22 °
deviation of the measured position/orientation from the goal position/orientation			
		genControl	*UniBiControl*
mean μ of the deviation from the goal	x	-10.1 mm	-1.2 mm
	y	-1.4 mm	0.1 mm
	φ	1.58 °	0.23 °
standard deviation σ of μ	x	6.9 mm	1.6 mm
	y	1.3 mm	0.9 mm
	φ	1.92 °	0.30 °
maximum deviation from the goal	x	-51.3 mm	3.9 mm
	y	-14.3 mm	-9.8 mm
	φ	6.88 °	0.71 °
deviation from the desired trajectory			
		genControl	*UniBiControl*
mean μ of deviation from desired trajectory	$v_{max} = 0.5 \frac{m}{s}$	9.3 mm	-3.5 mm
	$v_{max} = 0.363 \frac{m}{s}$	9.3 mm	-1.4 mm
	$v_{max} = 0.1815 \frac{m}{s}$	8.3 mm	0.7 mm
standard deviation σ of μ	$v_{max} = 0.5 \frac{m}{s}$	7.8 mm	4.1 mm
	$v_{max} = 0.363 \frac{m}{s}$	7.0 mm	3.7 mm
	$v_{max} = 0.1815 \frac{m}{s}$	6.3 mm	3.3 mm
maximum deviation from desired trajectory	$v_{max} = 0.5 \frac{m}{s}$	51.0 mm	-52.4 mm
	$v_{max} = 0.363 \frac{m}{s}$	80.1 mm	37.8 mm
	$v_{max} = 0.1815 \frac{m}{s}$	69.8 mm	-44.8 mm
kurtosis	$v_{max} = 0.5 \frac{m}{s}$	-0.25	4.29
	$v_{max} = 0.363 \frac{m}{s}$	0.60	5.61
	$v_{max} = 0.1815 \frac{m}{s}$	0.64	11.24

Table 7.1: *In the table, the results of the experiments in the Cell Culture Laboratory are subsumed. The first and second block show the statistical results of the first experiment, the third block shows the results of the second experiment.*

Chapter 8

Conclusion and Future Work

This diploma thesis addresses three important topics in mobile robotics. These are the self-localisation, the path planning and the control of a wheeled mobile robot. Compared to the software provided by the manufacturer, improvements have been achieved in position accuracy. In order to achieve that goal, a precise self-localisation mechanism has been developed. This mechanism together with a path planning algorithm is the basis for the robot's capability of following a certain route through its environment. The results obtained by this procedure provide an adequate basis for the robot to successfully perform the transport task which it was expected to carry out in the cell culture laboratory of the University of Bielefeld. Within the scope of this diploma thesis, however, a lot of alternative approaches could not be tried out. Thus, a comparison of this work's methods with others would be desirable for the future. Some direction for that will be given in the following sections.

8.1 Localisation

For achieving improvements in the control system as well as in the path planning module it is crucial to increase the precision of the self-localisation. A first step in this direction could be the redesign of the used system-model. Such a refined model would have to overcome the problem of occasional perturbations that can be observed in the robot's estimate of its position when new reflector marks appear within the measuring range of the laser range finders. The design of the model can be altered to improve the noise immunity of the self-localisation which occurs due to noise of the measuring devices. A higher noise immunity automatically improves the behaviour of the motion controllers. Besides improving the model, for example by using higher-order extended Kalman filters, alternative estimators like *particle filters* could be implemented.

Another problem of the introduced self-localisation is the lack of global positioning information. The estimates of the robot's position can be ambiguous. Especially when the features are distributed symmetrically, the calculation of the position is not unique. Methods like *multi-hypothesis tracking* could help to find a solution for this problem.

A valuable extension of the self-localisation module could be further extended by methods which allow the robot to explore its environment and to generate its own map from scratch. This problem is referred to as the SLAM-problem.[1] An attempt at solving this problem is presented in Dissanayake et al. (2001).[2]

Another interesting aspect of the self-localisation could be the use of *natural features*. These natural features can be calculated from the robot's measurements. They could be used to replace the reflector marks which are detected by the laser range finders. A lot of different natural features are conceivable. For example the intersections of lines in the laser scans or features like symmetry points in the robot's environment. In addition, data from other measuring devices like vision systems could be used.

8.2 Path planning

As mentioned in chapter 6.4, different optimality criteria can be used for the search of the path. The current path planning module looks for the shortest path. Other applications may need different criteria such as finding the safest path or the one with the lowest energy consumption. Furthermore, it would be possible to compose the path of other functions than the line segments used so far. An example for such a set of functions are splines. This would require a control system that is able to follow this kind of trajectories. Another matter of interest is the extension of the collision avoidance. Up to now, the collision avoidance brings the robot to a halt in front of an obstacle. An obstacle avoidance would be able to compute a detour around the object in the way. Moreover, the detected obstacle could be added to the map. When planning the next path, this object could be avoided *a priori*.

8.3 Control

Apart from an implementation of an obstacle avoidance in the path planning layer, as suggested in the previous section, an integration into the motion control system is possible. This approach is mainly used in behaviour based robotics and can be found in Arkin (1998).[3] As mentioned above, a control system which is able to follow a non-linear trajectory is desirable. This controller has to be able to handle changing set values at its input.

[1]SLAM = simultaneous localisation and map building
[2](DISSANAYAKE ET AL., 2001).
[3](ARKIN, 1998).

Appendix A

Mathematical Comments

A.1 Notes for the Kalman filter

1. note: Some features of positive semi-definite (psd) matrices

   (a) the diagonal elements are not negative, i.e. ≥ 0

   (b) let A be a psd matrix of the size $n \times n$, then for every matrix B of the size $m \times n$ it follows that BAB^T is psd!

2. note: With trace $A^T =$ trace A the following holds

$$\begin{aligned}\left((\mathbb{1} - K_tH)\left(P_t^- H^T \delta K_t^T\right)\right)^T &= \left(\left(P_t^- H^T\right)\delta K_t^T\right)^T (\mathbb{1} - K_tH)^T \qquad | \text{ with } (AB)^T = B^T A^T \qquad \text{(A.1)}\\ &= \delta K_t \left(P_t^- H^T\right)^T (\mathbb{1} - K_tH)^T\\ &= \delta K_t H \left(P_t^-\right)^T (\mathbb{1} - K_tH)^T\\ &= \delta K_t H P_t^- (\mathbb{1} - K_tH)^T \qquad |P_t^- = \left(P_t^-\right)^T \text{ because sym.}\end{aligned}$$

3. note

$$\begin{aligned}\left(K_t R \delta K_t^T\right)^T &= \delta K_t (K_t R)^T \qquad \text{(A.2)}\\ &= \delta K_t R^T K_t^T\\ &= \delta K_t R K_t^T \qquad |R^T = R \text{ because R diag. matr. and sym.}\end{aligned}$$

A.2 Homogeneous Transformations

The concept of *homogeneous transformations* was introduced in Forest (1969).[1] The basis of the concept are the *homogeneous coordinates*. Forest introduced them into computer graphics to overcome problems in matrix calculation. The idea is to represent an n-dimensional space by $n+1$ dimensions. A point (x, y, z) in the three-dimensional space is represented by a point (hx, hy, hz, h) in the four-dimensional space, where h is an arbitrary number. In robotics, a direct mapping between the spaces is used. Thus, h is set to one. The additional coordinate h can be seen as a scaling factor. This was needed in computer graphics, but may also be useful if the computational capacity of a computer is limited.

The transformations and coordinates are referred to as *homogeneous* because the representation for a class of objects involves no explicit constant. For instance, a two-dimensional line which is given by the equation $y = ax + b$ becomes the homogeneous equation $ax - y + bz = 0$, where $z = 1$. This equation is a three-dimensional representation of a line.

Homogeneous coordinates have the same meaning if each component, including the scale factor, is multiplied by a constant. Let $\vec{i}$, $\vec{j}$, and $\vec{k}$ be unit vectors along the x, y and z axis. Then,

$$\vec{v} = a\vec{i} + b\vec{j} + c\vec{k}$$

is a point vector. In homogeneous coordinates it is represented as a column matrix:

$$\vec{v} = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} aw \\ bw \\ cw \\ w \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \\ 1 \end{pmatrix}$$

where $w = 1$.

The use of homogeneous coordinates solves a problem of main interest in robotics. While a 3×3 matrix can describe the rotation, scaling and shear of a three dimensional object, a translation can not be described. The introduction of an extra column can solve the problem, but leads to a matrix that is not square and, therefore, does not have an inverse. This difficulty can be overcome by defining a 4×4 transformation matrix which describes the rotation, translation, shear, projection, local scaling and overall scaling transformations between two views of an object.

$$T = \left[\begin{array}{c|c} \text{rotation} & \text{translation} \\ \text{shearing} & \\ \text{local scaling} & \\ \hline \text{projection} & \text{scaling} \end{array}\right] = \left[\begin{array}{c|c} & \\ 3 \times 3 & 3 \times 1 \\ & \\ \hline 1 \times 3 & 1 \times 1 \end{array}\right]$$

In contrast to computer graphics, in robotics the only interest lies in rotation and translation transformations. Since the scale factor is one, the four-dimensional transformation maps directly

[1](FOREST, 1969).

into three dimensions. Thus, transformed homogeneous coordinates are the same as transformed ordinary coordinates.

$$T = \left[\begin{array}{c|c} & \\ \text{rotation} & \text{translation} \\ & \\ \hline 0 & 1 \end{array}\right]$$

A general transformation matrix, corresponding to a translation by a vector p is:

$$\text{Trans}(p_x, p_x, p_z) = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation is more complex. Rotation about a general axis can be decomposed into a sequence of rotations about the coordinate axes. There are three rotation transforms corresponding to rotation about the x, y and z axes by an angle ϕ:

$$\begin{aligned} \text{Rot}(x,\phi) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) & 0 \\ 0 & \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ \text{Rot}(y,\phi) &= \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ \text{Rot}(z,\phi) &= \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 & 0 \\ \sin(\phi) & \cos(\phi) & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

A detailed explanation with examples in two- and three-dimensional space can be found in McKerrow (1993).[2]

[2](McKerrow, 1993), chapter 3.

A.3 Moments of distributions

For the examination of the experiments in chapter 7, the mean, the standard deviation and the kurtosis of the measured values have been calculated. They represent different *moments* of statistical distributions.

The mean is referred to as the first moment. It is calculated as follows:

$$\mu = \frac{1}{N}\sum_{j=1}^{N} x_j$$

where $x_1, \ldots, x_N$ are the measured values. The mean μ estimates the value around which central clustering occurs.

The width of a distribution is characterized by the standard deviation. It is referred to as the second moment and defined by:

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{j=1}^{N} (x_j - \mu)^2}$$

The peakedness or flatness of a distribution in comparison to a normal distribution is described by the fourth moment. It is termed kurtosis.

$$\kappa = \left\{ \frac{1}{N}\sum_{j=1}^{N} \left[\frac{x_j - \mu}{\sigma}\right]^4 \right\} - 3$$

The -3 term makes the kurtosis zero for a normal distribution. For a flat distribution, the value is negative. A positive value describes a peaked distribution.

More details on the moments of a distribution can be found in Press et al. (1988).[3]

[3](PRESS ET AL., 1988), pp. 472-476.

Appendix B

Poetry

B.1 The Sorcerer’s Apprentice

The Sorcerer’s Apprentice by Johann Wolfgang von Goethe 1779, translation by Edwin Zeydel, 1955:

That old sorcerer has vanished
And for once has gone away!
Spirits called by him, now banished,
My commands shall soon obey.
Every step and saying
That he used, I know,
And with spirits obeying
My arts I will show.

Flow, flow onward
Stretches many
Spare not any
Water rushing,
Ever streaming fully downward
Toward the pool in current gushing.

Come, old broomstick, you are needed,
Take these rags and wrap them round you!
Long my orders you have heeded,
By my wishes now I've bound you.
Have two legs and stand,
And a head for you.
Run, and in your hand
Hold a bucket too.

Flow, flow onward
Stretches many,
Spare not any
Water rushing,
Ever streaming fully downward
Toward the pool in current gushing.

See him, toward the shore he's racing
There, he's at the stream already,
Back like lightning he is chasing,
Pouring water fast and steady.
Once again he hastens!
How the water spills,
How the water basins
Brimming full he fills!

Stop now, hear me!
Ample measure
Of your treasure
We have gotten!
Ah, I see it, dear me, dear me.
Master's word I have forgotten!

Ah, the word with which the master
Makes the broom a broom once more!
Ah, he runs and fetches faster!
Be a broomstick as before!
Ever new the torrents
That by him are fed,
Ah, a hundred currents
Pour upon my head!

No, no longer
Can I please him,
I will seize him!
That is spiteful!
My misgivings grow the stronger.
What a mien, his eyes how frightful!

Brood of hell, you're not a mortal!
Shall the entire house go under?
Over threshold over portal
Streams of water rush and thunder.
Broom accurst and mean,
Who will have his will,
Stick that you have been,
Once again stand still!

Can I never, Broom, appease you?
I will seize you,
Hold and whack you,
And your ancient wood
I'll sever,
With a whetted axe I'll crack you.

He returns, more water dragging!
Now I'll throw myself upon you!
Soon, 0 goblin, you'll be sagging.
Crash! The sharp axe has undone you.
What a good blow, truly!
There, he's split, I see.
Hope now rises newly,
And my breathing's free.

Woe betide me!
Both halves scurry
In a hurry,
Rise like towers
There beside me.
Help me, help, eternal powers!

Off they run, till wet and wetter
Hall and steps immersed are Iying.
What a flood that naught can fetter!
Lord and master, hear me crying! -
Ah, he comes excited.
Sir, my need is sore.
Spirits that I've cited
My commands ignore.

"To the lonely
Corner, broom!
Hear your doom.
As a spirit
When he wills, your master only
Calls you, then 'tis time to hear it."

Appendix C

Acknowledgements

As iron sharpens iron, so one man sharpens another.
Book of Proverbs 27:17 (NIV)

A project like this would not have been possible without the help of numerous people.
Sincere thanks are given to them all!

Jianwei Zhang for giving us the chance to work on this project and for caring about our future

Torsten Scherer for his permanent urge to criticize in a fair and constructive manner, for his ability to think ahead and for the courage to believe in us...

Markus C. Ferch for the willingness to discuss problems even during tea time

Angelika Deister for the nice encouragements when we needed them

Frank Rötling for technical support

Jürgen Lange for the disentanglement of all sentences which were definitely too long

Christoph Reuter for his great knowledge of the English grammar

Gareth C. Charter for being the ”English Man in Bielefeld”

Christine Ströker for being the librarian

All members of the Department of Technical Computer Science for being a team

Hagen Stanek for beer, pencil and Roblets^{TM}

Jure Zakotnik for being the photographer

Appendix D

Editorial

This diploma thesis is the result of a team work. The chapters were composed by Axel Schneider and Daniel Westhoff.

Axel Schneider wrote:

In chapter 1:	section 1.1
In chapter 2:	section 2.1
In chapter 3:	section 3.1
In chapter 4:	sections 4.1 and 4.3
In chapter 5:	sections 5.1 and 5.3
In chapter 6:	sections 6.1 and 6.3
In chapter 7:	sections 7.1 and 7.3

Daniel Westhoff wrote:

In chapter 1:	section 1.2
In chapter 2:	sections 2.2 and 2.3
In chapter 3:	section 3.2
In chapter 4:	section 4.2
In chapter 5:	sections 5.2 and 5.4
In chapter 6:	sections 6.2, 6.4 and 6.5
In chapter 7:	section 7.2

Chapter 8 and the appendix were written by Axel Schneider and Daniel Westhoff together.

Bibliography

ANDERSON, B. D. O. AND MOORE, J. B. (1979). *Optimal Filtering*. Prentice-Hall Information and System Science Series. Prentice-Hall, Englewood Cliffs, NJ.

ARKIN, R. C. (1998). *Behavior-Based Robotics*. Intelligent Robots and Autonomous Agents. MIT-Press, Cambridge, Mass.

BAR-SHALOM, Y. AND LI, X.-R. (1993). *Estimation and Tracking: Principles, Techniques and Software*. The Artech House Radar Library. Artech House Inc., Norwood, Mass.

CRAIG, J. J. (1989). *Introduction to Robotics: Mechanics And Control*. Addison-Wesley Series in Eletrical and Computer Engineering: Control Engineering. Addison-Wesley, Reading, Mass., 2. edition.

DENAVIT, J. AND HARTENBERG, R. (1955). A kinematic notation for lower-pair mechanisms based on matrices. *Journal of Applied Mechanics*, 22(1):215–221.

DENAVIT, J. AND HARTENBERG, R. (1964). *Kinematic Synthesis of Linkages*. McGraw-Hill, New York.

DISSANAYAKE, M. W. M. G., NEWMAN, P., CLARK, S., DURRANT-WHYTE, H. F., AND CSORBA, M. (2001). A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241.

FOREST, A. (1969). Coordinates, transformations and visualization techniques. Technical report, CAD Group Document No. 23, Cambridge University.

GALLISTEL, C. (1990). *The Organisation of Learning*. The M.I.T. Press, Cambridge, Mass. and London.

Gelb, A., editor (1994). *Applied Optimal Estimation*. The M.I.T. Press, Cambridge, Mass. and London. 13. print.

HART, P., NILSSON, N., AND RAPHAEL, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science and Cybernetics*, 4(2):100–107.

HOFER, K. (1998). *Regelung elektrischer Antriebe, Innovation durch Intelligenz*. VDE-Verlag, Berlin, Offenbach.

HUNN, U. (1993). *Diplomarbeit: Implementierung von Planung und Steuerung der Bewegung eines autonomen Fahrzeugs in einer Fuzzy-Umgebung*. Institut fuer Prozessrechentechnik und Robotik, Universtitaet Karlsruhe.

JACOBS, O. (1993). *Introduction to Control Theory*. Oxford University Press, Oxford, 2. edition.

JULIER, S. AND UHLMANN, J. (1996). A general method for approximating nonlinear transformations of probability distributions. Technical report, RRG, Dept. of Engineering Science, University of Oxford.

KALMAN, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82:35–45.

KUMMERT, F. (WS 2000/2001). *Skript: Vorlesung Musteranalyse*. Faculty of Technology, Applied Computer Science, University of Bielefeld.

LATOMBE, J. C. (1996). *Robot Motion Planning*. The Kluwer international series in engineering and computer science. Kluwer, Dordrecht, Boston. 4. print.

LEE, D. (1996). *The Map-Building and Exploration Strategies of a Simple Sonar-Equipped Robot*. Cambridge University Press, Cambridge.

LIU, Y. AND ARIMOTO, S. (1991). Proposal of tangent graph and extended tangent graph for path planning of mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 312–317.

LLOYD, J. E. (1998). Trajectory generation implemented as a non-linear filter. Technical report, Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada.

LOZANO-PEREZ, T. AND WESLY, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570.

MAYBECK, P. S. (1979). *Stochastic Models, Estimation and Control*, volume 1. Academic Press, London. chapter 1-3.

MCKERROW, P. J. (1993). *Introduction to Robotics*. Electronic Systems Engineering Series. Addison-Wesley, Reading, Mass.

MOHINDER, G. S. AND ANDREWS, A. P. (1993). *Kalman Filtering*. Prentice-Hall Information and System Science Series. Prentice-Hall, Englewood Cliffs, NJ.

MUIR, P. F. AND NEUMAN, C. P. (1987). Kinematic modeling of wheeled mobile robots. *Journal of Robotic Systems*, 4(2):281–340.

O'ROURKE, J. (1994). *Computational Geometry in C*. Press Syndicate of the University of Cambridge, Cambridge.

PRESS, W. H., FLANNERY, B. P., TEUKOLSKY, S. A., AND VETTERLING, W. T. (1988). *Numerical Recipes in C*. Press Syndicate of the University of Cambridge, Cambridge.

RUSSEL, S. J. AND NORVIG, P. (1995). *Artificial Intelligence*. Prentice-Hall Series in Artificial Intelligence. Prentice-Hall, Englewood Cliffs, NJ.

SCHERER, T. (1998). *Diploma Thesis: Design and Implementation of a Trajectory Generator for Arbitrarily Moving Targets and On-Line Singularity Robustness*. Faculty of Technology, Technical Computer Science, University of Bielefeld.

SCHMIDT, S. (February 1970). Computational techniques in kalman filtering. In *Theory and Applications of Kalman Filtering, AGARDograph 139*. NATO Advisory Group for Aerospace Research and Development, London.

SCHMIDT, S. (May 1976). Practical aspects of kalman filtering implementation. In *AGARD-LS-82*. NATO Advisory Group for Aerospace Research and Development, London.

Schrick, K.-W., editor (1977). *Anwendungen der Kalman-Filter-Technik*. Methoden der Regelungstechnik. R. Oldenbourg, München.

SEDWICK, R. (1992). *Algorithmen in C++*. Addison-Wesley, Bonn, München, Paris.

SHETH, P. AND UICKER, J. (1971). A generalized symolic notation for mechanisms. *Journal of Engineering for Industry*, 93:102–112.

SPITERI, C. J. (1990). *Robotics Technology*. Saunders College Publishing, Orlando, Fl.

WELCH, G. AND BISHOP, G. (01/2000). An introduction to the kalman filter. [html]. University of North Carolina at Chapel Hill. available at http://www.cs.unc.edu/ welch/kalman/kalman_filter/kalman.html.