Diploma Thesis

Segmentation Algorithms for 2D-Laserscans in an indoor environment

Gabriel Nock

Universidad de Zaragoza
Dept. de Informática e Ingeniería de Sistemas
Grupo de Robótica y Tiempo Real

July, 2003

Abstract

This work treats with the segmentation of 2D environment Laser data, captured by an *Autonomous Mobile Indoor Robot*. It is part of the data processing, which is necessary to navigate a mobile robot error free in its environment. The whole process can generally be described by data capturing, data processing and navigation. In this project the data processing deals with data, captured by a Laser-Sensor, which provides two dimensional data by a series of distance measurements i.e. point-measurements of the environment. These point series have to be filtered and processed into a more convenient representation to provide a virtual environment map, which can be used of the robot for an error free navigation. This project provides different solutions of the same problem: the conversion from distance points to model segments which should represent the real world environment as close as possible. The advantages and disadvantages of each of the different *Segmentation-Algorithms* will be shown as well as a comparison taking into account the *Computational Time* and the *Robustness* of the results.

Acknowledgement

I want to thank to all the people that helped me to accomplish this Diploma Thesis at the *Centro Politécnico Superior - Universidad de Zaragoza*.
Special thanks I want to give to my spanish tutor Prof. Jose Neira who made it possible for me to carry out my Diploma Project in Spain and who advised the entire project.
Thanks to my german tutor Prof. Dr. Oliver Bittel, *University of applied sciences of Konstanz Fachhochschule Konstanz* for his supervision of the project in Germany.
As well I want to thank all the members of the *Grupo de Robótica y Tiempo Real* for their help and advices on my project, especially to Diego Ortin for his support on mathematical problems.
Finally I want to thank Carlos Muñoz for the innumerable topic-and non-topic related discussions we had in the Laboratory and the Cafeteria and his inexhaustible effort and patience to improve my spanish language.

Table of Contents

List of Figures

1 Introduction

Today robots are applied in many different areas of industry, science etc. with distinguished intentions of usage. They ease the work (e.g. Industry Robots), carry out dangerous or unpleasing tasks (e.g. Military Robots) or make for humans impossible things possible (e.g. Outer space missions). In many cases an autonomic behaviour is desired or required which leads to a more or less complex mechanics-computer system, which is able to react in a correct and reliable way to outside influences. The hardware of a mobile robot consists, beside the mechanical chassis, of actors, which enable it to move, and the sensors, which enable it to capture environment data. Typical robot sensors are optical cameras, tactile-, ultrasonic- or laser sensors.

The general tasks of robot software can be defined by data capturing, data processing and controlling (navigation). In a major part of the cases a virtual environment map is used to navigate a robot error free in his environment. The closer the virtual map is to the real world the bigger is the possibility to react without errors. Captured data always is afflicted with errors resulting from the sensors so the goal of the data processing is to filter out and minimize the provided errors to achieve a result as optimal as possible.

The data capturing rely on the kind of sensor and used techniques. This normally calls for data pre-processing to receive convenient data for the following data processing. In our case we will work on data obtained by a two-dimensional laser sensor mounted on a mobile indoor robot. The given hardware will be explained closer in Chapter 2. The available data, which consist of a list of ordered distance measurements relative to the laser sensor, has to be filtered, optimised and brought into a representation, which provides a convenient way to create the virtual environment map. In our case the required representation are *Segments* (e.g. wall segment). Due to this representation the above explained data process is called *Segmentation*. The resulting segments are used to create the virtual map.

In this project different *Segmentation Algorithms* were implemented and compared. The emphasis lies on the definition of the different algorithms and the presentation of the advantages and the disadvantages of each of them given by their characteristics.

The important characteristics which have to be given by the algorithms are *Velocity* and *Robustness*.

Velocity The *Velocity* of an algorithm is given by the necessary computational time to obtain the desired results on the given data. Although the existing computer techniques provide fast working processors and hardware systems the calculational time still has limits. This results on one side on the economical and spacial possibilities of the robots, which often restrict the calculational performance of the system and on the other side of the increasing amount of data, which is necessary, to provide more information of the environment to achieve optimized results.

To compare the different algorithms regarding the calculational time time-measurements were made on the same computer system and the complexity of the algorithms will be shown (Chapter 8 and 5).

Robustness The term *Robustness* can be used to describe the quality of the results compared to the real world model. A robust algorithm minimizes data errors and isn't vulnerable to so called "Outliers[1]", which can falsify the desired results considerably.

The consideration if a generated virtual model in fact is good or not depends on the requirements of the tasks as well as on the subjective evaluation of the observer. There are two cases of wrong estimations apart from location deviation: The refusal of an, in reality existing, segment (*"False Negative"*) and the acceptance of a non-existing segment (*"False Positive"*). Obviously it depends on the task to accomplish which of the two cases should be avoided. An estimation of a false negative can cause an accident in navigation i.e. an existing obstacle is not detected and the vehicle could hit it, whereas an estimation of a false positive could cause e.g. an unnecessary and unintentional change in navigation.

The goal of a *Robust Segmentation Algorithm* is to exclude from the segment estimation all points which don't belong to planar surfaces and to find all, in the real world existing, surfaces or segments.

In our case obviously we tried to minimize the errors but with the tendency to the false negatives. So we tended to decide to refuse a segment if we aren't sure i.e. we chose the parameters in a way to

[1]Points that aren't belonging to a planar surface

ensure that the possibility of the existence of an estimated segment can be considered to be high.

In Chapter 2 I will start to describe the available hardware, i.e. the mobile robot and the used laser sensor, as well as the used software to develop and implement the algorithms.

In Chapter 3 I will describe general definitions and terms, which will be used through this thesis. Besides mathematical and geometrical definitions and defined terms I will give an oversight of the parameters, which were used as criteria. These parameters are general and will be used with all algorithms.

The procedure of data preparation and *Pre-Processing* will be described in Chapter 4.

The different segmentation algorithms will be described in Chapter 5 with their definitions and procedures. The general procedure in all cases has the same design: pre-processing, line- or segment extraction, post processing. In this section the *Line- or Segment Extraction* will be described based on the different chosen algorithms.

After the extraction of potential lines, given by their line parameters, or given by sets of points, the *Post-Processing* (Chapter 6) is applied. Here the given line parameters or point groups are used to build the resulting segments. The resulting segments will be checked on characteristics and if they comply the requirements they will be added to the resulting set of segments.

For each algorithm different variations in parameters of procedures are applicable. The different possibilities of parameter choice and procedures and the resulting outcome will be described in Chapter 7.

In Chapter 8 the results and measurements will be listed and compared. For a proper comparison we need a test scenario, which represents the real world as close as possible to provide a relevant simulation. In our case two different test series are available. Each one consists of a collection of ordered scans, which were recorded by the robot and laser on which the algorithms have to be applied on. They were captured in the University building of Zaragoza i.e. computer laboratories and corridors. They represent an adequate part of the real world of an indoor environment i.e. there exist normal room equipment e.g. tables, chairs etc. as well as relevant moving obstacles i.e. persons. All together there are nearly 700 single scans with all together nearly 250.000 scan points.

In this thesis we will illustrate the obtained results of the segmentation algo-

rithms in graphics provided by MATLAB. In general they show the sensor-obtained data and either intermediate resulting lines and segments or the finally resulting segments. The scale and the axes are equal for the major parts of the graphics. The units of the axes are hold in *meters* and the laser in general is illustrated as triangle with the point showing into the scan direction.

2 Technical Environment

In the following chapter I want to give a detailed overview over the used hardware (robot, laser) and the development software, which was used to implement the algorithms.

2.1 Hardware

The Robotic- and Realtime Group at the University of Zaragoza has at its disposal, among others, the mobile robot named *"Triton"* (Figure 1), which is based on a wheelchair chassis. Its actors are two electric motors able to actuate independently for the two given rear wheels. It has chargeable batteries so it can operate completely autonomically. A computer system is mounted on it with the operating system Windows2000© and the installed development environment Microsoft Visual C++© makes it possible to program directly "on" the wheelchair.

It has besides various optical cameras a mounted *SICK Laser sensor Type LMS 200* (Figure 2). This type of laser scanner provides the following technical data:

Maximal Distance Range The *Maximal Distance Range* of the SICK Laser depends on the used distance unities. It provides unities of $[m]$ or $[mm]$. The Laser transmits the measured value in a data message with a fixed length. In this data message the measured distance is represented by a 13 Bit-value. In [mm]-mode thus the values are lying between 1 and $2^{13} = 8192mm$. In [m]-mode the maximal range is $\approx 80m$.

Scan Angle Range The maximum angle of the SICK Laser Sensor is restricted to 180°.

Angular Resolution The SICK Laser provides different angular resolutions. This *Switch mode* allows angular steps of $1°, 0.5°$ and $0.25°$.

Systematic Error The systematic Error is specified[2] with $\pm 15mm$.

Statistical Error The statistical Error $(1\,\sigma)$ is specified with $5mm$.

[2]Specified by the SICK Comp. data sheet

Data Interface The SICK Laser provides the data interfaces RS422 & RS232 with the switchable Transfer Rates of 9.6/19.2/38.4/500 $kBaud$.

Figure 1: Mobile Robot Triton

Figure 2: SICK Laser scanner Type LMS 200

The Laser sensor is mounted in front of the wheelchair "Triton". Its altitude is $\approx 0.8m$ and it's located $\approx 0.8m$ in front of the rotation axis which is lying centred between the rear wheels. The laser is almost the foremost point of Triton so the measured distances can be considered to be situated relative to the "front line" of the Robot. Figure 3 shows the general scheme of the robot and the Laser.

2.2 Software

To develop the software and simulate the algorithms two different software packages were used.

For the prototyping and the first simulations the software package $MATLAB^{©}$ was used. This software provides a mathematical environment with extensive mathematical functions. MATLAB is a tool for technical computing and offers a scripting language as well as the possibility to define functions and data structures. It's specialized in matrix and vector calculations and offers

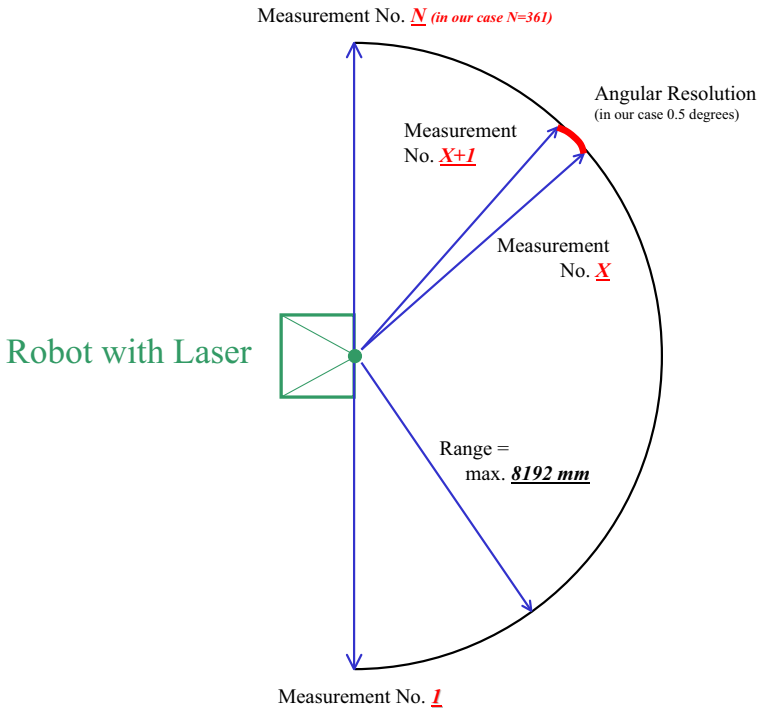

Figure 3: Scheme of Robot "Triton" and SICK Laser sensor

extensive possibilities for fast data visualisation. Different program modules give a plenty of possibilities for debugging and profiling. MATLAB doesn't compile program code or scripts but works as an interpreter what occasionally makes it unsuitable to obtain adequate conclusions on computational time of algorithms and programs for the later use on the real-time robot system. Thus it was used for prototyping, simulation, analysis and testing.

To obtain adequate results on computational time and memory requirements, which are comparable to the later real time robot system, the algorithms were implemented as well in C/C++. To compare the different algorithms the time measurements were obtained on the C/C++ - algorithms. To visualize the results under C/C++ a free graphical library, called *Allegro*[3], was used.

[3]The graphical Library "Allegro" will be explained more detailed in the appendix

Now we know the environment in which this project was realized. In following chapter we will present the fundamentals of the context of the project respective the math and the basic requirements of the algorithms.

3 General Definitions and Fundamentals

This chapter should give an insight of the math and the geometrical basics which were applied in this project. First I will describe the objects given by trivial geometry and further how they were used in the implementation of the algorithms. This should help to understand the algorithms and avoid misunderstandings with the usage of special terms in this thesis.

3.1 Geometrical Objects

In the following section I want to give an overview over the geometrical objects the projects treats with. First I will describe the different objects in the mathematical-geometrical way and I will show the different possibilities they can be respresented. In the implementation of the algorithms it was useful in certain cases to transform the object representation due to the optimization of the algorithms or the simplification of the paradigm which will be shown in the following sections.

3.1.1 Mathematical Representation of Geometrical Objects

The 2-dimensional space In this projects two different representations were used to define and represent geometrical objects. On one side the *Cartesian Representation (or Cartesian Space)* and the *Polar Representation (or Polar Space)* on the other. By the use of transformations we are able to transform geometrical objects from one space into the other.

Cartesian Space
In the two dimensional cartesian space a coordinate is represented by two length parameter $P_{cart} = \begin{pmatrix} X \\ Y \end{pmatrix}$. One represents the distance in direction of the X-Axis and the other in direction of the Y-Axis. The reference point is the origin $O = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ of the coordinate system, which in our case is represented by the Laser Sensor. The units of the parameters are $[m]$ or $[mm]$.

Polar Space
In the two dimensional polar space the two parameters of a coordinate are a length parameter ρ (Rho) and a angular parameter θ (Theta). $P_{pol} = \begin{pmatrix} \rho \\ \theta \end{pmatrix}$ The angular parameter θ represents the angle from the X-Axis counter clockwise to the coordinate and the parameter ρ represents the distance from the

origin to the coordinate. The unit of ρ are $[m]$ or $[mm]$ and the unit of θ are *Degrees* or *Gradients*. Figure 4 shows the different representations.

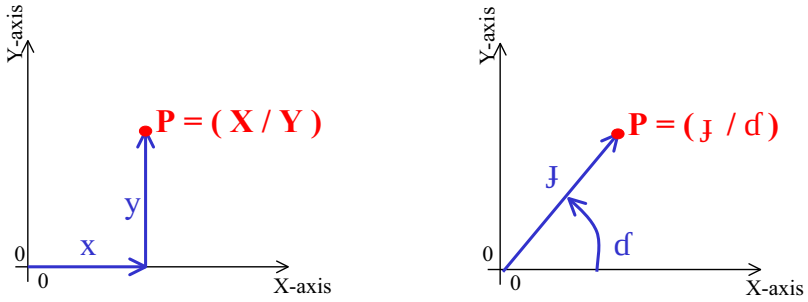

Figure 4: Cartesian and Polar representation of a coordinate in the 2-dimensional space

The Transformation of a coordinate from the polar into the cartesian representation, $F : \Re^{pol} \rightarrow \Re^{cart}$, results as follows:

$$C_{pol} = \begin{pmatrix} \rho \\ \theta \end{pmatrix} \equiv C_{cart} = \begin{pmatrix} \rho\cos(\theta) \\ \rho\sin(\theta) \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \tag{1}$$

The Transformation of a coordinate from the cartesian into the polar representation, $F : \Re^{cart} \rightarrow \Re^{pol}$, results as follows:

$$\begin{aligned} C_{cart} = \begin{pmatrix} x \\ y \end{pmatrix} \equiv C_{pol} = \begin{pmatrix} \rho \\ \theta \end{pmatrix} &= \begin{pmatrix} \sqrt{x^2+y^2} \\ \arcsin\left(\frac{y}{\rho}\right) \end{pmatrix} \\ &= \begin{pmatrix} \sqrt{x^2+y^2} \\ \arccos\left(\frac{x}{\rho}\right) \end{pmatrix} \end{aligned} \tag{2}$$

Representation of a Point

In this thesis the above used term *Coordinate* corresponds to the geometrical

object *Point* which will be represented equally,

$$P_{cart} = \begin{pmatrix} x \\ y \end{pmatrix} \qquad or \qquad P_{pol} = \begin{pmatrix} \rho \\ \theta \end{pmatrix}$$

depending on the used representation $\Re$.

Representation of a Straight Line
The geometrical object *(straight)Line* also can be represented in cartesian or polar form and is defined by 2 parameters.

In **Cartesian Representation** the parameters are the *line slope* m and the *y-axis offset* c. A point is "lying" on a line[4] if it fulfills following line-function:

$$y = mx + c \tag{3}$$

The line parameters can be obtained by transforming two given line points $P_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$, $P_m = \begin{pmatrix} x_m \\ y_m \end{pmatrix}$ with $\Delta x = x_n - x_m$, $\Delta y = y_n - y_m$.

$$\begin{aligned} L = \begin{pmatrix} m \\ c \end{pmatrix} &= \begin{pmatrix} \frac{\Delta y}{\Delta x} \\ y_n - \left(x_n \frac{\Delta y}{\Delta x}\right) \end{pmatrix} \\ &= \begin{pmatrix} \frac{\Delta y}{\Delta x} \\ y_m - \left(x_m \frac{\Delta y}{\Delta x}\right) \end{pmatrix} \quad with\ \Delta x \neq 0 \end{aligned} \tag{4}$$

Obviously, due to the fraction, the parameters are undefined for $\Delta x = 0$. Therefore this restriction gives preference to another representation without such a restiction.

In **Polar Representation** the parameters are the *Norm* or *Length* ρ of the *Normal Vector*[5] from the origin to the line and the counter clockwise angle θ between the normal vector and the X-Axis .

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{5}$$

[4] Sometimes also referred as "member" of a line
[5] *Normal Vector*: A vector which stands orthogonal to a line or plane

Given two points the line parameter can be obtained as follows:

$$P_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix}, P_m = \begin{pmatrix} x_m \\ y_m \end{pmatrix} \quad \text{with} \quad \Delta x = x_n - x_m \ , \ \Delta y = y_n - y_m$$

$$L = \begin{pmatrix} \rho \\ \theta \end{pmatrix} = \begin{pmatrix} x_n \cos\left(\arctan\left(\frac{\Delta y}{\Delta x}\right)\right) + y_n \sin\left(\arctan\left(\frac{\Delta y}{\Delta x}\right)\right) \\ \arctan\left(\frac{\Delta y}{\Delta x}\right) \end{pmatrix}$$

$$= \begin{pmatrix} x_m \cos\left(\arctan\left(\frac{\Delta y}{\Delta x}\right)\right) + y_m \sin\left(\arctan\left(\frac{\Delta y}{\Delta x}\right)\right) \\ \arctan\left(\frac{\Delta y}{\Delta x}\right) \end{pmatrix} \tag{6}$$

$$\text{with} \quad \theta \in \left]-\tfrac{\pi}{2}; +\tfrac{\pi}{2}\right] \quad \text{and} \quad \rho \in \mathbb{R}$$

Thus a negative ρ specifies a normal-vector in the 2nd or 3rd quadrant. A $\theta = \frac{\pi}{2}$ specifies a normal-vector on the Y-Axis where a positive ρ specifies it in the positive and a negative in the negative direction.

The polar representation avoids problems with vertical lines where $\Delta x = 0$.

Figure 5 illustrates the differences between cartesian and polar line representation.

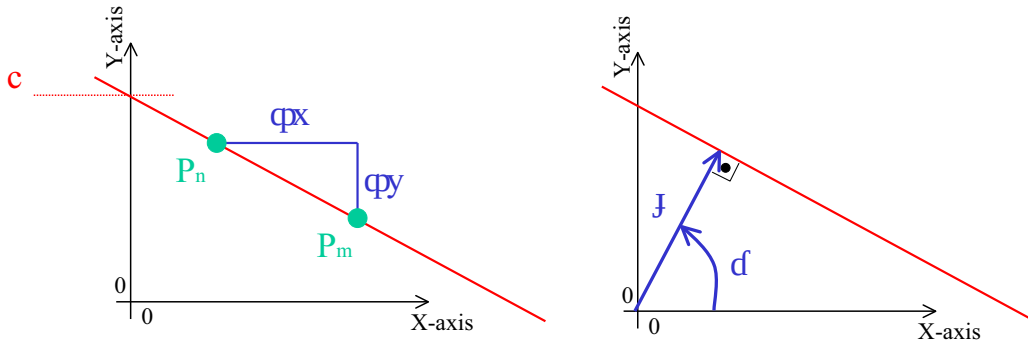

Figure 5: Cartesian and Polar Line Representation

3.1.2 Geometrical Entities used in the Implementation

The previously given geometrical objects build the fundamentals of the algorithms. In this section I want to describe the objects which were used in the implementation and how they were applied on the algorithms.

Measurement Data
The initial point of the implementation are the data captured and transmitted by the laser.

Like explained before the SICK laser measures its environment by discrete distance measurements. It divides its maximum angular range into constant steps and proceeds on each step a distance measurement by a laser beam. His maximum range is specified with exactly 180°. In our case the factor of division was 0.5° so the laser starts with its first measurement exactly 90° to the *right* of his straight alignment and provides a measurement each 0.5° progressing to the left. Hence it provides 361 serial distance values. If we define the coordinate system with the origin exactly on the laser and his straight direction as the positive X-Axis we save the values as a polar coordinate with the measured distance as value ρ and the angle to the lasers alignment as

$$\theta = -90° + (n \cdot 0.5°) = -\frac{\pi}{2} + (n \cdot \frac{\pi}{360}) \qquad \text{with} \qquad n = \{0, 1, \ldots, 360\}$$

This serial data capturing gives us the possibility to index the coordinates by their order of appearance which is essential for certain parts of the further data processing.

Thus one scan is available as an array of 361 indexed polar coordinates captured relatively to the laser location.

Figure 6 displays a captured scan with 361 scan points captured by the laser and represented in a two dimensional cartesian coordinate systems with the laser sensor as dimension origin ($x = y = 0$).

Point Representation
A point is represented in the trivial way explained in the previous chapter either in his polar or cartesian coordinates with its respective index.

Point Groups
Point groups play an important role in segmentation, since a segment of the world model normally should be represented by a number of according points.

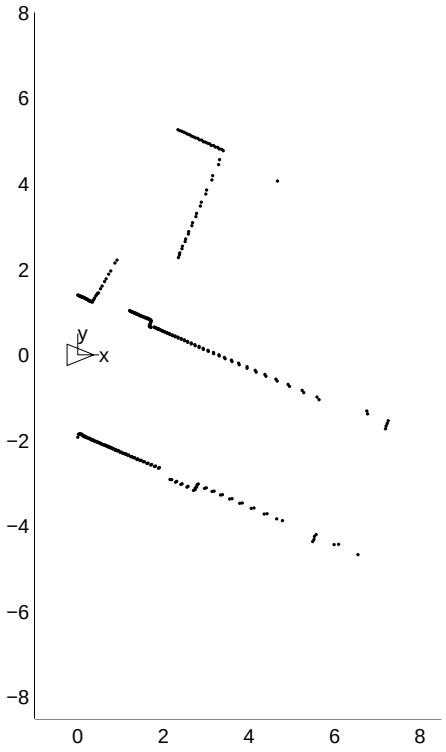

Figure 6: Captured Data of one Laserscan

Obviously points don't change their coordinates whereas their indices have to be changed in certain cases.

This occurs e.g. after pre filtering of a scan when points, which aren't to be considered correct measurements, are eliminated from the previous data. On one hand this leads to different numbers of the points on which the following segmentation algorithm has to be applied and on the other hand the points have to be re-indexed in the correct order.

After filtering the points keep their indices for the further proceeding. The previous group will be divided into groups regarding the characteristics of the algorithm. After grouping the initial data the resulting groups are considered to define a line and will be used to estimate the line parameters.

In general one point can't belong to more than one line. In cases (if the algorithm allows it) one point can be considered as an end-point of a segment and thus can be used for two segments representing for each segment an end-

point. This normally is a desired feature due to the fact that segments in the real world normally have a closed connection i.e. corners and so a virtual model with closed segments can be considered to be closer to the real world model. But this characteristics as well can lead to significant errors since a considered "corner-point" only in the fewest cases was captured in real world in fact exactly in the corner. So the tuning of the algorithm parameters has to minimize such errors.

Line Representation
A line or straight line is represented like explained in the previous sections by two parameters either in polar or cartesian representation. A line is considered to be infinite which is essential e.g. for the calculation of the perpendicular distance to a point. A line can be specified as well by two points which in cases is used to calculate a perpendicular distance or to extract a covariance of a parameter matrix. In our case a line is always the basis for the later to build segment.

In the literature, related to the topic in this thesis, in cases a line also is referred as *"edge"*.

Segment Representation
A segment is part of an infinite line and has to be extracted from one line. It is specified by its two endpoints which trivially are part of the former line, so a segment is represented by 4 parameters:

$$S = \begin{pmatrix} x_{P1} \\ y_{P1} \\ x_{P2} \\ y_{P2} \end{pmatrix}$$

$$\text{with its endpoints: } P1 = \begin{pmatrix} x_{P1} \\ y_{P1} \end{pmatrix}, P2 = \begin{pmatrix} x_{P2} \\ y_{P2} \end{pmatrix}$$

3.2 Geometrical Relations between Objects

Distance Measurements A very important feature in line extraction is the distance between a point and a line. This characteristic normally decides if a point is accepted to be assimilated into the group of points which is defining the line. There are different possibilities to define this distance e.g.

the distance only measured in the specific direction of the given dimensions of the coordinate system (X-or Y-axis) or the perpendicular distance as well as the distances that takes into account the given uncertainty of the geometrical entities which includes a probabilistic consideration of the results. The normal geometrical distances are the so called *Euclidean Distances*.

3.2.1 Euclidean Distance

The euclidean distances in the 2-dimensional space are calculated in the trivial geometrical way.

Distance Point-to-Point
Given 2 points P_m and P_n the distance d results as follows:

$$d = \sqrt{(x_n - x_m)^2 + (y_n - y_m)^2} \tag{7}$$
$$\text{with } P_m = \begin{pmatrix} x_m \\ y_m \end{pmatrix}, P_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

Distance Point-to-Line
As above explained there are several ways to define a distance between a point and a line in the 2-dimensional space. In this project we normally refer to the *Perpendicular Distance* which describes the smallest distance between the two entities. For the different representations the definitions are as follows.

Cartesian representation
Assuming a line being specified by two points P_k and P_l and the distance is to measure to a given point P_0 it results:

$$d = \frac{|(x_l - x_k)(y_k - y_0) - (y_l - y_k)(x_k - x_0)|}{\sqrt{(x_l - x_k)^2 + (y_l - y_k)^2}} \tag{8}$$
$$\text{with } P_k = \begin{pmatrix} x_k \\ y_k \end{pmatrix}, P_l = \begin{pmatrix} x_l \\ y_l \end{pmatrix} \text{ and } P_0 = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$$

Polar representation

Given a line in polar representation (Figure 5) $L = \begin{pmatrix} \rho_L \\ \theta_L \end{pmatrix}$ the distance is calculated by:

$$d = y_P \sin(\theta_L) + x_P \cos(\theta_L) - \rho_L \tag{9}$$

$$\text{with } P = \begin{pmatrix} x_P \\ y_P \end{pmatrix} \text{ or}$$

$$d = \rho_P \sin(\theta_P)\sin(\theta_L) + \rho_P \cos(\theta_P)\cos(\theta_L) - \rho_L \tag{10}$$

$$\text{with } P = \begin{pmatrix} \rho_P \\ \theta_P \end{pmatrix}$$

Vector representation
A further form to calculate the Point-to-Line Distance is to use a special vector representation of a line. Distance d then is calculated as follows:

$$d = \alpha_L \cdot P_c - \beta_L \tag{11}$$

with

$$P_c = \begin{pmatrix} x \\ y \end{pmatrix}$$

α_L represents the *Unity Norm Vector*[6] to the line and β_L the norm of the Normal Vector to the line. "$\cdot$" specifies the *Inner product* also known as *Dot Product* or *Scalar Product* of two vectors and P_c represents a measurement point in its cartesian representation.

Given a line in polar representation $L = \begin{pmatrix} \rho_L \\ \theta_L \end{pmatrix}$ the parameters are given by:

$$\alpha_L = \begin{pmatrix} \cos(\theta_L) \\ \sin(\theta_L) \end{pmatrix} \quad \text{and} \quad \beta_L = \rho_L \tag{12}$$

3.3 Data Probability Distribution

Given sensor data always has to be considered to be affected by errors. In this chapter I want to describe the fundamentals of the handling with error afflicted sensor data as well as to show on an example how this paradigm was used in the implementation of the algorithms. [DuHa73] and [Cast98] provided the fundamentals.

[6] Unity Norm Vector: The normal vector from the origin to the line with the Norm $n = 1$

3.3.1 Squared Mahalanobis Distance

The *Squared Mahalanobis Distance* constitutes a unitless value which takes into account the statistical error distributions of two geometrical entities. It can be considered to describe a distance between geometrical objects by the means of the covariances and is generally specified in related literature like [DuHa73] by:

$$D^2 = [x - \mu]^T \Sigma^{-1} [x - \mu] \tag{13}$$

where μ describes a d-component *Mean Vector* and x describes a d-component column vector. $(x - \mu)^T$ specifies the transpose of $(x - \mu)$ and Σ is a d-by-d component *Covariance Matrix* with its inverse Σ^{-1}.

In our mono-dimensional case for the distance the above given formula reduces to

$$D^2 = \frac{(x - \mu)^2}{\sigma^2} \tag{14}$$

where x specifies the measured euclidean distance and μ the deviation from the mean. σ describes the uncertainty of the used function which in our case is the used distance function.

With the calculated value D^2 a *Hypothesis Test* can be applied. It is considered to be true whenever:

$$D^2 \leq \chi^2_{r,\alpha} \tag{15}$$

where the threshold $\chi^2_{r,\alpha}$ is obtained from the χ^2-distribution with $r = \text{rank}(x)$ and α the probability of rejecting a correct model.

3.3.2 The Normal Density of Sensor Data

The problem with sensor data is that their measurements can't be considered to be absolutely correctly representing the real world since various errors are influencing the measurements. Due to the fact that the sources of errors are multiple and each of these errors has an arbitrary probability distribution the *Central Limit Theorem*[7] says that the cumulative distribution approaches a *Normal Distribution* with the mean μ and a variance σ^2.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}} \tag{16}$$

[7] Also referred as "Moivre-Laplace Limit Theorem"

This distribution is specified entirely by the two parameters *mean* μ and the *variance* σ^2 so for simplicity the abbreviation for this distribution is denoted with

$$x \sim N(\mu, \sigma^2) \tag{17}$$

(Read: x is distributed normally with the mean μ and the variance σ^2.)

And

$$E[x] = \mu = \int_{-\infty}^{+\infty} xp(x)dx \tag{18}$$

$$E[(x-\mu)^2] = \sigma^2 = \int_{-\infty}^{+\infty} (x-\mu)^2 p(x)dx \tag{19}$$

In the multidimensional (d-dimensional) case the general multivariate normal density is specified by:

$$p(x) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{[(x-\mu)^T \Sigma^{-1}(x-\mu)]} \tag{20}$$

Here x is a d-component column vector and μ the d-component *Mean Vector*. $(x-\mu)^T$ specifies the transpose of $(x-\mu)$ and Σ is a d-by-d component *Covariance Matrix* with its inverse Σ^{-1} and the determinant $|\Sigma|$.

The abbreviation for this distribution is likely to the univariate case:

$$x \sim N(\mu, \Sigma) \tag{21}$$

and

$$\mu = E[x] \tag{22}$$

$$\Sigma = E[(x-\mu)(x-\mu)^T] \tag{23}$$

To be more specific: Let x_i be the ith component of the column vector x, μ_i the ith component of the mean vector μ and σ_{ij} the i-jth component of the covariance matrix Σ then μ_i and σ_{ij} are as follows:

$$\mu_i = E[x_i]$$

$$\sigma_{ij} = E[(x_i - \mu_i)(x_i - \mu_i)^T]$$

In case of statistical independence of x_i and x_j the covariance matrix

$$\Sigma = \begin{pmatrix} \sigma_i & \sigma_{ji} \\ \sigma_{ij} & \sigma_j \end{pmatrix}$$

reduces to the diagonal matrix with the values $\sigma_{ij}, \sigma_{ji} = 0$ and $p(x)$ reduces to the product of the mono-dimensional normal densities of each component x_i.

So the knowledge of the covariance matrix allows us to calculate the spread of the data in every direction of the d-dimensional space. Data tends to scatter into a cluster whose center is defined by the mean μ. The shape of the cluster depends on the Covariance matrix which in our 2-dimensional case shapes an ellipse. The *Eigenvectors* of the covariance matrix determine the direction of the axes and the *Eigenvalues* determine the length of the ellipse

Figure 7 illustrates the distribution of 2-dimensional data around the given mean μ and the elements of the covariance Σ.

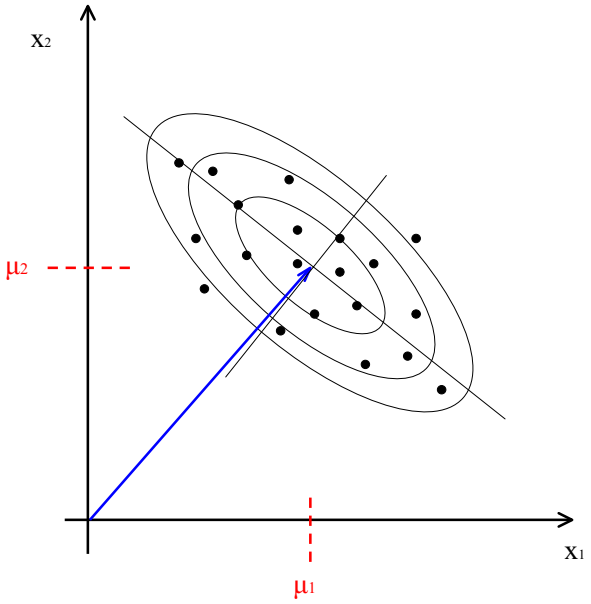

Figure 7: Scatter diagram of a distribution $p(x) \sim (\mu, \Sigma)$

The points of constant density form an ellipse around the mean where the term $((x-\mu)^T\Sigma^{-1}(x-\mu))^2$ is constant. This brings us back to the above described *Squared Mahalanobis Distance* (Chapter 3.3.1). So the ellipses represent a constant value of the density of the given distribution as well as

a constant value of the squared mahalanobis distance. Thus if we know the squared mahalanobis distance based on the given density we now can apply a hypothesis Test based on the χ^2-distribution and determine with a certain probability if our element contains to the cluster which is based on the given distributions.

3.3.3 Linear and Non-linear Transformations

In many cases we have to transform given data into another representation. The most trivial case is the transformation of data in polar representation into cartesian or vice versa. This transformation already was explained in section 3.1.1. The normal density function parameters μ and σ^2 therefore have to be transformed as well.

Linear Transformations
A linear transformation $F(u) = x$ normally is represented in matrix notation:

$$F(u) = x \tag{24}$$
$$= \alpha u + \beta \quad with\ \ given \quad u \sim (\mu_u, \Sigma_u) \tag{25}$$

α represents a $n \times m$-Matrix and β and n-dimensional column vector.

So the new density function $x \sim (\mu_x, \Sigma_x)$ has to be found.

From Equations 22 and 23 we can calculate the new parameters

$$\mu_x = E[x] \text{ and } \Sigma_x = E[(x - \mu_x)(x - \mu_x)^T]$$

$$\begin{aligned}\mu_x &= E[x] \\ &= E[F(u)] \\ &= E[\alpha u + \beta] \\ &= \alpha E[u] + \beta \\ &= \alpha\mu_u + \beta \end{aligned} \tag{26}$$

$$\begin{aligned}\Sigma_x &= E[(x - \mu_x)(x - \mu_x)^T] \\ &= E[(\alpha u + \beta - \alpha\mu_u - \beta)(\alpha u + \beta - \alpha\mu_u - \beta)^T] \\ &= E[(\alpha u - \alpha\mu_u)(\alpha u - \alpha\mu_u)^T] \\ &= E[(\alpha(u - \mu_u))(\alpha(u - \mu_u))^T] \\ &= E[\alpha(u - \mu_u)(u - \mu_u)^T\alpha^T] \\ &= \alpha\, E[(u - \mu_u)(u - \mu_u)^T]\, \alpha^T \\ &= \alpha\, \Sigma_u\, \alpha^T \end{aligned} \tag{27}$$

Non-linear Transformations

In the non linear case of a transformation we can't use the same transformation over the whole function. Thus we want to linearize the function only at one location a so called *Sampling Point* which in our case should be the mean vector μ_u. Therefore we use the approximation of a *Taylor Series* which consists of adding the derivations of the function at a sample point. For simplification it's common only to use the first derivation. So the transformation of a non-linear function can be written formally as follows:

$$\begin{aligned}F(u) &= x \\ &\simeq F(\mu_u) + F(\mu_u)'(u - \mu_u) \end{aligned} \tag{28}$$

$F(\mu_u)'$ specifies the partial derivation of $F(\mu_u)$ which is called *Gradient* or *Jacobian Matrix*. Formally it is written as

$$\begin{aligned}F(\mu_u)' &= \frac{\partial F}{\partial u}(\mu_u) \\ &= \nabla F(\mu_u) \end{aligned} \tag{29}$$

and consists of a m× n - Matrix with the dimensions depending on the number of variables of the function $F(u)$.

Thus α corresponds to the gradient $\nabla F(\mu_u)$ so

$$\alpha = \nabla F(\mu_u)$$

$$\begin{aligned} F(\mu_u) &= \nabla F(\mu_u)\mu_u + \beta \qquad \Rightarrow \\ b &= F(\mu_u) - \nabla F(\mu_u)\mu_u \end{aligned} \tag{30}$$

applied on the mean and the covariance we get the new parameters μ_x and Σ_x as follows:

$$\begin{aligned} \mu_x &= E[x] \\ &= \alpha\mu_u + \beta \\ &= [\nabla F(\mu_u)\mu_u] + F(\mu_u) - [\nabla F(\mu_u)\mu_u] \\ &= F(\mu_u) \end{aligned} \tag{31}$$

and

$$\begin{aligned} \Sigma_x &= \alpha \ \Sigma_u^{-1} \ \alpha^T \\ &= [\nabla F(\mu_u)] \ \Sigma_u^{-1} \ [\nabla F(\mu_u)]^T \end{aligned} \tag{32}$$

3.3.4 The Extended Information Filter

Another problem we have is the estimation of a state vector by a given set of noisy measurements. This occurs e.g. if we want to estimate a line by a given set of measured points. We not only want to have the two line parameters ρ and θ but as well an estimation of the covariance. This can be achieved by *The Extended Information Filter* ([Cast98], [Neira93]).

The requirement therefore is the availability of a relationship between our line (which will be called in the following the *State Vector*) and the set of measurements. In our case this would be given by a distance function between a point and the line. The therefore given equation 9 takes into account the two parameters of the point and the two parameters of the line. The result is a mono-dimensional value respective the perpendicular distance.

Another requirement is an initial state of the parameters due to the fact, that this kind of filtering is an actualisation algorithm which needs an initial state to actualise.

So first I want to give the fundamentals of the algorithm: Let x be a state vector whose value is to be estimated, and let there be n independent measurements y_k with $k \in \{i, \ldots, n\}$ each with a normal error distribution:

$$\hat{y}_k = y_k + u_k \quad ; \quad u_k \sim N(0, S_k) \tag{33}$$

so S_k specifies the covariance of a measurement.

Between the state vector and the measurements exists a non-linear function of the form: $f_k(x, y_k) = 0$. The linear approximation explained in the previous chapter is given as well as:

$$f_k(x, y_k) \simeq h_k + H_k(x - \hat{x}) + G_k(y - \hat{y}_k) \tag{34}$$

with

$$h_k = f_k(\hat{x}, \hat{y}_k) \quad ; \quad H_k = \left.\frac{\partial f_k}{\partial x}\right|_{(\hat{x}, \hat{y}_k)} \quad ; \quad G_k = \left.\frac{\partial f_k}{\partial y_k}\right|_{(\hat{x}, \hat{y}_k)} \tag{35}$$

Here h_k specifies the function with the given parameters x and the measurement y_k which has to be integrated. H_k and G_k are parts of the jacobian matrix with respect to the parameters of the state vector (H_k) and with respect to the measurement (G_k).

A new state vector and a new covariance for the state vector can be calculated by the given formulas:

$$\hat{x}_n = P_n M_n \quad and \quad P_n = Q_n^{-1} \tag{36}$$

where

$$Q_n = \sum_{k=1}^{n} F_k \quad ; \quad M_n = -\sum_{k=1}^{n} N_k \tag{37}$$

by calculating for each given measurement k

$$F_k = H_k^T (G_k S_k G_k^T)^{-1} H_k \quad ; \quad N_k = H_k^T (G_k S_k G_k^T)^{-1} h_k \tag{38}$$

This algorithm is a so called *Batch Algorithm* which integrates a set of n measurements at one time to a state vector. Thus the complexity of the Extended Information Filter is directly related to the number of measurements to integrate n.

To integrate a new measurement m the procedure is quite simple: Calculating F_m and N_m add F_m to the inverse covariance P_n and re-invert it, as well as to add to M_n $-N_m$ and recalculate $\hat{x}_m$.

3.3.5 The Extended Kalman Filter

Another algorithm to actualize state vectors is the *The Extended Kalman Filter* ([Cast98] [WeBi02]). In contrary to the Extended Information Filter the Kalman Filter works completely recursive i.e. that for each integrated measurement a new state vector is calculated as base for the proximate integration.

Given an state vector x_{m-1} and its Covariance P_{m-1} a recursive actualisation x_m and P_m can be obtained as follows:

Be

$$H_{k,m} = \left.\frac{\partial f_k}{\partial x_m}\right|_{(x_m,\hat{y}_k)} \quad ; \quad G_{k,m} = \left.\frac{\partial f_k}{\partial y_k}\right|_{(x_m,\hat{y}_k)} \tag{39}$$

with the geometrical relation function $h_{k,m} = f_{k,m}(x_m, \hat{y}_k)$ with the embraced actual state vector x_m, the measurements to integrate $\hat{y}_k$ and its covariance S_k. An actualisation factor the so called actual *Kalman Gain* $K_{k,m}$ results from:

$$K_{k,m} = P_{m-1}H_{k,m}^T(H_{k,m}P_{m-1}H_{k,m}^T + G_{k,m}S_kG_{k,m}^T)^{-1} \tag{40}$$

and the actualization of the state vector and its covariance is obtained by:

$$x_m = x_{m-1} + K_{k,m}(h_{k,m}) \tag{41}$$

$$P_m = (I - K_{k,m}H_{k,m})P_{m-1}^{-1} \tag{42}$$

where I specifies the *Identity Matrix* with the according dimensions.

The requirement for the integration of the first measurement is the existence of an initial state vector x_0 and covariance P_0 whose quality is essential for the quality of the results. The farther the initial solution from the correct state vector the worse the following estimation and therefore the final result. For each integration of a new measurement and actualisation of the state vector and its covariance a constant computational effort is necessary.

Application of the Mahalanobis Distance Now we have the tools to transform elements into a distinguished representation, extract Covariances and applying e.g. the Mahalanobis distance. Let's contemplate the distance between a measurements and a line given by two points and pose the question if the point could pertain to the line taking into account their uncertainties.

Our given function to use is the distance function from a point to a line whereas the point is given in polar coordinates.

$$f_k(L, P_k) = \rho_{P_k} \cdot \sin(\theta_{P_k}) \cdot \sin(\theta_L) + \rho_{P_k} \cdot \cos(\theta_{P_k}) \cdot \cos(\theta_L) - \rho_L$$

$$with \qquad L = \begin{pmatrix} \rho_L \\ \theta_L \end{pmatrix} \qquad and \qquad P_k = \begin{pmatrix} \rho_{P_k} \\ \theta_{P_k} \end{pmatrix}$$

The line parameters have to be calculated in the normal analytical way using the two line points.

The non-linear function has to be approximated by:

$$f_k(L, P_k) \approx f(\hat{L}, \hat{P}_k) + \nabla f(\hat{L}, \hat{P}_k)(L - \hat{L})$$

where $f(L, P_k)$ is the desired value and $f(\hat{L}, \hat{P}_k)$ is the value given by the measurements.

The transposed Jacobian of the function f is specified as follows[8]:

$$\nabla f_k^T = \begin{pmatrix} \frac{\partial f}{\partial \rho_{P_k}} \\ \frac{\partial f}{\partial \theta_{P_k}} \\ \frac{\partial f}{\partial \rho_L} \\ \frac{\partial f}{\partial \theta_L} \end{pmatrix} = \begin{pmatrix} \sin(\theta_{P_k}) \cdot \sin(\theta_L) + \cos(\theta_{P_k}) \cdot \cos(\theta_L) \\ \rho_{P_k} \cdot \cos(\theta_{P_k}) \cdot \sin(\theta_L) + \rho_{P_k} \cdot -\sin(\theta_{P_k}) \cdot \cos(\theta_L) \\ -1 \\ \rho_{P_k} \cdot \sin(\theta_{P_k}) \cdot \cos(\theta_L) + \rho_{P_k} \cdot \cos(\theta_{P_k}) \cdot -\sin(\theta_L) \end{pmatrix} \tag{43}$$

For the point we assume statistical independence between the distance and the angular error. In our case a line is specified by two points so in effect the error of the line parameters are *not* independent so the covariance for the line parameters is not diagonal.

The covariance matrices for a point or the line result as follows:

$$Cov_{P_k} = \begin{pmatrix} \sigma_{\rho_{P_k}} & 0 \\ 0 & \sigma_{\theta_{P_k}} \end{pmatrix} \qquad Cov_L = \begin{pmatrix} \sigma_{\rho_L} & \sigma_{\rho_L \theta_L} \\ \sigma_{\theta_L \rho_L} & \sigma_{\theta_L} \end{pmatrix} \tag{44}$$

Assuming as well statistical independency between a point and line (obviously this is only the case if the point still isn't "integrated" into the line) we get:

$$Cov_{f_k} = \begin{pmatrix} Cov_{P_k} & 0 \\ 0 & Cov_L \end{pmatrix} = \begin{pmatrix} \begin{pmatrix} \sigma_{\rho_{P_k}} & 0 \\ 0 & \sigma_{\theta_{P_k}} \end{pmatrix} & 0 \\ 0 & \begin{pmatrix} \sigma_{\rho_L} & \sigma_{\rho_L \theta_L} \\ \sigma_{\theta_L \rho_L} & \sigma_{\theta_L} \end{pmatrix} \end{pmatrix} \tag{45}$$

[8]The Transpose was used to produce a 4×1 matrix due to the document width which would be exceeded by a 1×4 matrix

This covariance matrix now can be used to calculate the covariance of the function at the sample point by using:

$$\sigma_f^2 = \nabla f_k Cov_{f_k}^{-1} \nabla f_k^T$$

This signifies a matrix calculation of:

$$[1 \times 4] \cdot [4 \times 4] \cdot [4 \times 1]$$

Due to the fact that the matrix is a diagonal block matrix (and therefore most of the values are 0) we are able to split it back into its blocks and using only the relevant parts of the Jacobian (given in equation 35):

$$\sigma_f^2 = H^T Cov_L H + G^T Cov_P G \tag{46}$$

This reduces the calculation of former $[4 \times 4]$ matrices into calculations with easier to handle $[2 \times 2]$ matrices.

In this equation everything is known except the covariance of the line Cov_L. It is obtained by applying the Extended Information Filter on the inital state vector i.e. the line parameters and integrating the two line points. This will produce a covariance for the line but will not change the line parameters itself due to the fact, that the actualisation of the vector $\hat{x}$, given in equation 36, will be zero. It is easy to see that the distance function from two points to the line defined by them will be 0, so N and respectively M (equation 38 and 37) will be zero. But the covariance given by P is not-zero and thus specifies the covariance of the line defined by two points.

Now all the parameters are given to calculate the Mahalanobis distance from the point to the line. The equation

$$D^2 = \frac{(d-\mu)^2}{\sigma_f^2}$$

with a $\mu = 0$ provides a value which can be considered to be the "distance from point to line measured in σ's". With the given value we want to apply a hypothesis test e.g. with the condition of a probability of at least $p = 95\%$ certainty that if the point belongs to the line we will accept it or respectively that with a maximum probability of $p_f = 5\%$ the point will be rejected although it belongs to the line[9]. The threshold is given by the χ^2-distribution

[9]This is the case of *"False Negatives"*

with the significance level of $1 - p = p_f = 0.05$ and a rank $r = rank(f) = 1$. From a χ^2-distribution table in normal statistical literature the value is given with $\chi^2_{0.05,1} = 3.84$. So if

$$D^2 \leq 3.84$$

the point will be accepted regarding the hypothesis test.

Figure 8 illustrates the squared mahalanobis distance between a Point and a Line. The used covariance of a point was set[10]:

$$Cov_P = \begin{pmatrix} 0.005^2 & 0 \\ 0 & deg2rad(0.0125°)^2 \end{pmatrix}$$

With the given standard deviation of $5mm$ in direction of ρ and $0.0125°$ for

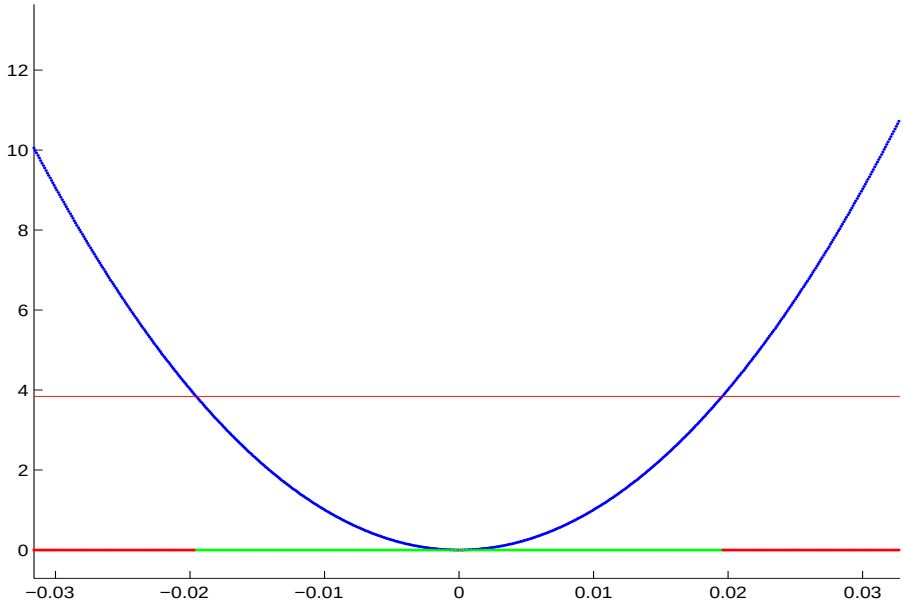

Figure 8: Mahalanobis Distance between a Point and a Line varying the euclidean distance

θ the margin for acceptance lies little below $20mm$.

[10]Values are given by the SICK Laser sensor

3.4 Defining Algorithm Parameters

The goal of this project is to extract features, respectively segments, of sensor captured data which should represent the real world as close as possible. The algorithms themselves have completely different modes of operations though the goal always is the same: extracting segments which are defined by groups of points. Like mentioned before the consideration if a segment is "good or not" also depends on the subjective evaluation of the observer. But to give a possibility to estimate the quality of the results regarding the tasks and the requirements it is necessary to define parameters which have to be taken into account during the procedure of segmentation. These parameters define the requirements which the results have to accomplish. They are chosen to "tune" and to optimize the algorithms and in generally were used for all the algorithms in the same way. They should help to characterise the algorithms and the produced results.

A parameter which is normally applied on a point is:

Point-to-Line distance Like explained in chapter 3.2 a very important and often used characterisation is the distance from a Point to a Line. This distance normally is calculated for a threshold test where the answer should be found, if a point is *sufficiently close* to a line. This threshold should be parameterised and will be called throughout this thesis *Point-to-Line* -parameter or criteria.

The parameters which are applied on already defined segments are the following:

Distance between segments Different segments which are parts of one specified line aren't compelled to be connected or defining one large segment, if the distance between two adjacent endpoints is too large. Thus there is considered to exist a gap between two segments. The used parameter for this threshold will be called *Inter-Segment distance*. This occurs e.g. with a door thus representing an *Inter-Segment Gap* between two segments of the same wall.

Minimum number of Points A segment only should be considered to exist if it is specified by enough measurements. This defined parameter will be called *Minimum Number of Points*.

Minimum length of Segment A segment only should be considered to exist if its length is larger than a defined parameter called *Minimum Length of Segment*.

Minimum Density of a Segment A combination of the parameters *Minimum Number of Points* and *Minimum Length of Segments* leads to a characterisation *Density of a Segment*. This is specified by the mean distance of adjacent points of a segment which depends on the length and the number of points specifying a segment.

Maximum number of "invalid" points between two measurements It can occur that a distance gap in a segment is smaller than the predefined threshold though the gap itself constitutes a significant interruption due to the fact of a large number of measured points in this gap. This occurs in cases of segments with different angle regarding to the laser where the projection of one segments onto the other is very small but should be considered to present a gap. The used parameter will be called *Number of Invalid Points*.

This parameter can only be used due to the fact that the scanned data is available in an ordered way and so information about adjacency is provided.

Figure 9 illustrates cases where the above parameters are applied and affect the results.

In this chapter we introduced the mathematical basics which were applied for the implementation and realization of the algorithms as well as the criteria which have to be fulfilled by the final results. Now we are able to introduce and describe the algorithms whereby we want to start with the introduction of the pre processing procedure.

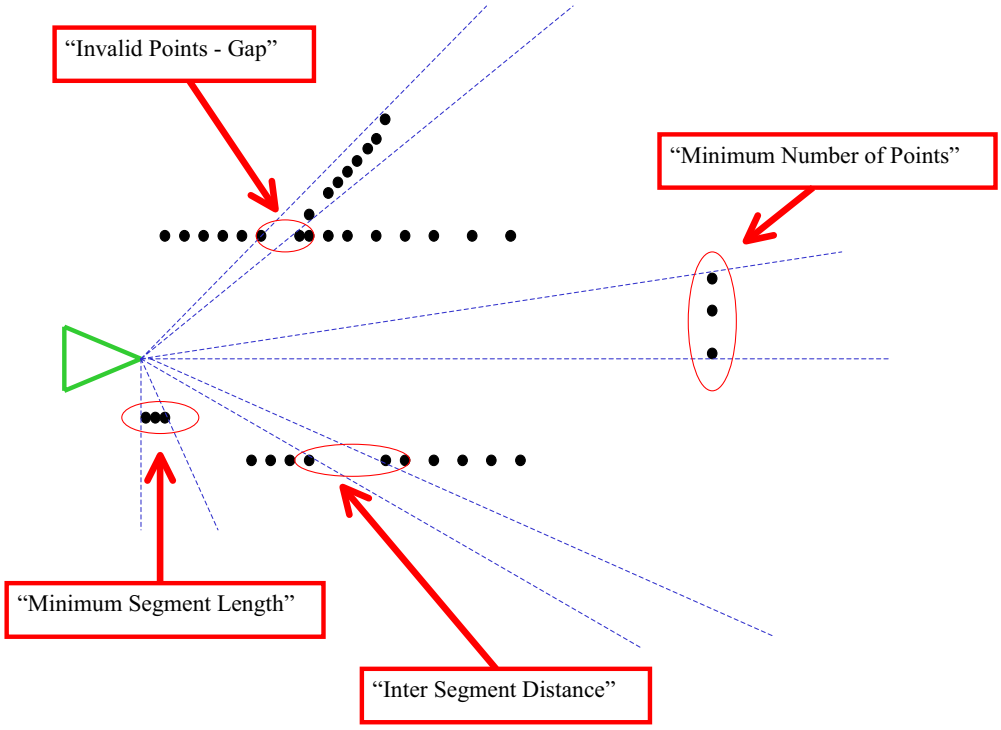

Figure 9: Application of predefined parameters on virtual scan data

4 Preprocessing

In the following chapter I want to explain the data processing before the application of the segmentation algorithms. The whole process of segmentation generally is the same for each of the algorithms:

- Preprocessing
- Segmentation
- Postprocessing

Firstly the data has to be filtered of invalid measurements and thus produces a group of measurement data which has to be used for the segmentation. This preparation of data is proceeded in the *Preprocessing-Step* which can be slightly varying for the different algorithms according to their characteristics.

On the resulting *one* group of measurements the segmentation algorithms have to be applied with the goal of producing *several* point groups where each group defines a potential segment according to the real world.

In the final *Postprocessing-Step* these potential point groups have to be checked on compliance regarding the predefined criteria which should specify a segment.

As already mentioned above, the Preprocessing is supposed to prepare the raw measurement data for the algorithms. It should be applied with the goal to obtain correct data and finally produce an optimisation on the whole segmentation procedure. The *Filtering of Invalid Scan points* has to be applied on all algorithms whereas the *Filtering of Outliers* only concerns algorithms which are sensible to them.

4.1 Filtering Invalid Scanpoints

Like explained in chapter 2 the used Laser sensor is used in an indoor mode with a maximum range of a 2^{13}-Bit [mm]-value. If the laser beam isn't reflected within this range by an obstacle it provides a filled data message. In our case this signifies a value of $8192mm$. Thus we have to define a *Maximum Range* value which should give us certainty about the validity of the distance value. Taking into account the hardware error and a sufficiently large margin the parameter for the maximum range was defined

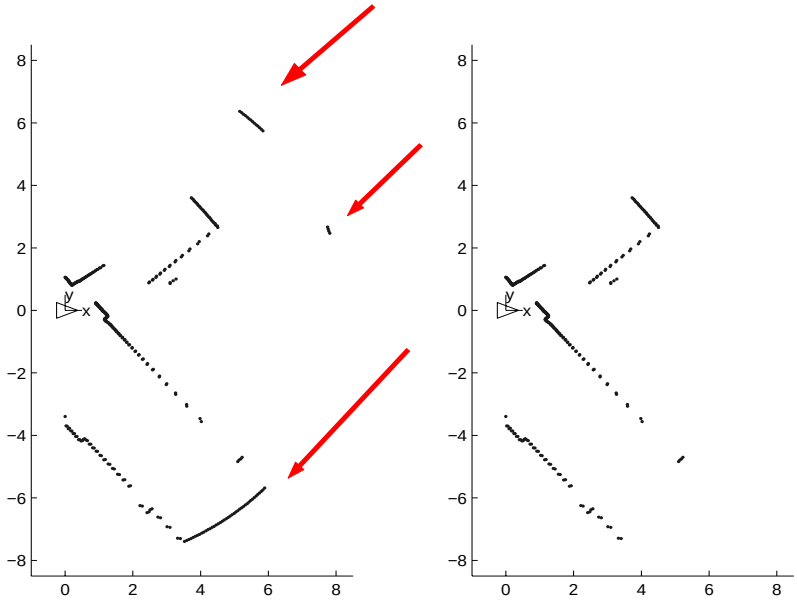

Figure 10: Scan data before and after the filtering of ignored measurements

with $range_{max} = 8100mm$. We don't know if the obtained maximum value is due to the exceeding of the range or results from a measurement which was measured exactly with this distance. To be certain only to proceed on points, which can be considered to be correct measurements, we ignore the maximum values and proceed only on data within the maximum laser range.

This step of filtering out the measurements which exceed the maximum range is applied for *all* the algorithms in advance and trivially has a complexity which is linear to the number of measurements $O(n)$ with $n =$ number of laser-provided points (our case n = 361).

Figure 10 shows an example of raw data and the filtered valid data. The robot was located in a corridor which is longer than $8100mm$ and had insight into a laboratory whose opposite wall as well is farther than the maximum range whereas the obstacles in the laboratory were situated within the max. range. The invalid measurements are indicated.

4.2 Filtering Outliers

Some algorithms react very sensible on so called *"Outliers"*. These measurements are captured accidentally and aren't belonging to a real world segment. This occurs e.g. as a result to bad reflection characteristics or non relevant discontinuances of segments (e.g. small but deep crevices in walls). An outlier can be specified by the characteristic of a big distance to his two adjacent neighbours. Thus a measurement, whose distance to his previous and to his proximate captured measurement exceeds a predefined threshold, is considered to be an outlier and will be erased from the initial point group. Obviously an erased outlier won't be regarded in the outlier-check of his proximate neighbour since an outlier is considered to be not existing and therefore shouldn't be included anymore.

For algorithms which are sensible on outliers the outlier-filtering will be proceeded in advance. The filter has a complexity directly depending on the number of valids and therefore has the complexity $O(n)$ with $n =$ number of valid points[11].

Due to the geometrical fact that measurements farther away from the laser will automatically have larger distances the threshold is specified *distance relative*. To derive an appropriate threshold we took into account the laser hardware error as well as an examination of the given test data compared to the real world. Good results were obtained with a threshold of $50 - 80\frac{mm}{m}$.

Figure 11 shows a virtual example of the Filtering of Outliers. The red measurements will be erased whereas the green will remain in the measurement data set.

In this chapter we described the procedure of preprocessing the raw input data as preparation for the pure segmentation algorithms. The input data now is available in a general and common way which provides an equal precondition for all of the algorithms. This should make a later comparison of the results more convenient. In the following chapter I will give an introduction to the procedures and the paradigms of the different segmentation algorithms.

[11] Hereinafter *"valid"* signifies all points within the maximum laser-range

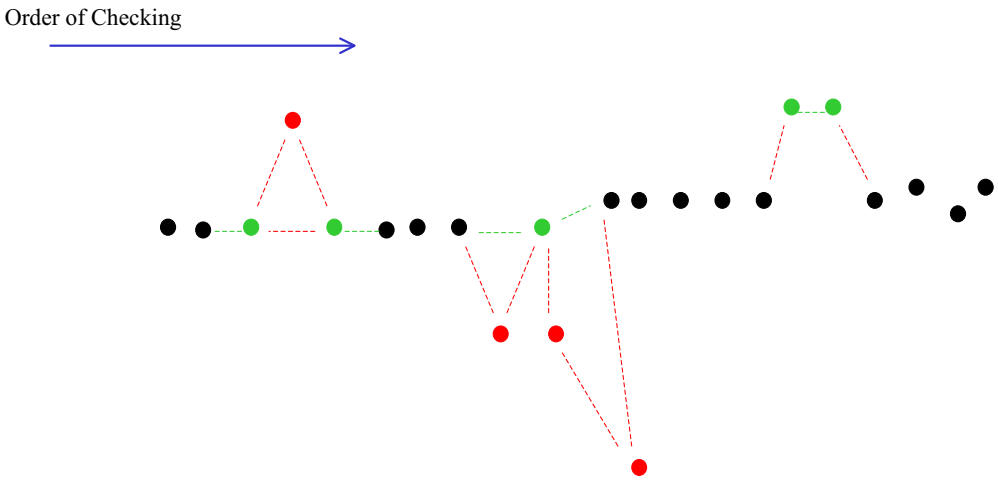

Figure 11: Filtering of Outliers

5 Segmentation Algorithms

In this chapter I will give an introduction to the implemented algorithms. First I will give the theoretical fundamentals: *How do they work?* and *What procedures sequences are applied?* to extract the required point groups from the initial data. The emphasis should lie on the paradigm of the algorithms and the differences in the modes of operation.

5.1 Choice of the algorithms

In this project we tried to implement different algorithms which are based on substantially different aspects of feature extraction but with the requirement of potentially equal results: A segment-model as close as possible to the real world. We want to show that different procedures can lead to the same results and will point out their characteristics, advantages and their disadvantages.

One one side we wanted to analyse algorithms which are commonly applied on this a kind of problem and therefore are thoroughly tested and confirmed on their mode of operation e.g. the *Split&Merge algorithm* or the *Hough-Transformation*. On the other side we wanted to apply algorithms which had less importance until now in this field of application but whose importance is increasing over the last couple of years.

The different modes of operation can be classified by a multiplicity of criteria. In the following I only want to point out some possibilities for a potential classification:

Sequence of involving the measurement into the resulting model:

Ordered:	Split & Merge
Unordered:	RANSAC , Hough, EM

Usage of Characteristics:

Pure Geometrical:	Split & Merge , Hough
Geometrical & Statistical:	RANSAC , EM

Information about Segment relevance:

Weighted:	Hough , EM
Unweighted:	Split & Merge , RANSAC

Iteration proceeding on set of data:

Complete Data :	Split & Merge , Hough[1] , EM
Reduced Data:	RANSAC, Hough[2]

So a classification of the algorithms depends on the point of view of the observer or the specified goal of the tasks. Thus we just want to show the potencies of the algorithms and compare them on the different requirements of the tasks.

A precondition for the choice of the algorithms was a reasonable complexity in comprehension and implementation so the possibilities on optimization and variation are easier to understand and demonstrated.

5.2 The Split & Merge - Algorithm

The *Split & Merge Algorithm* uses exclusively the geometrical relation between the points and lines, which would be potentially specified by existing points. It is based on the basic principles of the *Top-Down Polyline Splitting* Algorithm, also called *Recursive Subdivision*, and the *Bottom-Up Merging* Algorithm. The fundamentals are given by [JaScKa95]. It combines the two different algorithms iteratively to receive an optimal result.

In the following I will explain the two algorithms, their combination and how they are used to produce the required point groups.

Polyline Splitting

The Polyline Splitting searches the preliminary endpoints of all segments recursively. Its name arises from the fact that the result is a *Polyline* where the vertices specify the endpoints of the preliminary segments. The group of points, situated between the vertices, define with its two vertices the later segment.

The algorithm splits the initial line by searching the particular point in between with the largest perpendicular line distance. This point is set as a new vertex, respectively a new segment endpoint, and builds two new segments with the former given endpoints. For the thereby new generated segments the algorithm searches again the vertices. The algorithm stops the recursion in case that the distance to the farthest found point lies below a predefined threshold.

The initial line is specified by the first and the last captuerd scan point. These therefore build the first and respectively the last vertex of our resulting polyline.

Figure 12 illustrates the splitting procedure.

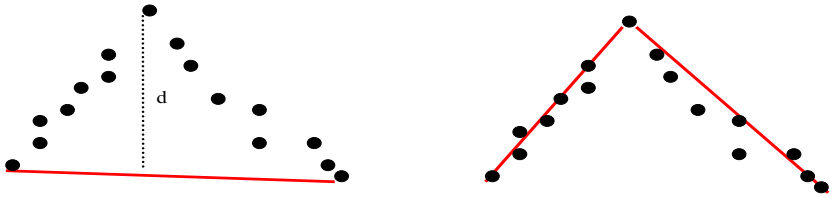

Figure 12: Split – Calculating all distances and inserting vertex

The algorithm of the split-step was implemented as follows:

```
=============================================================================
global Points
global Vertices
-----------------------------------------------------------------------------
Function: recursive split
Input:    Indices of first and last point of current line segment
-----------------------------------------------------------------------------
function: split( first_index, last_index )
   for ( i=first_index .. last_index )
        max_dist, max_index = findMaximum(first_index,last_index)
```

```
        if ( max_dist > DIST_THRESHOLD )
            addVertice( max_index )
            split(first_index, max_index)
            split(max_index, last_index)
        end_if
   end_for
end_function
=============================================================================
Function: finds maximum distance and index
Input:    Indices of first and last point of current line segment
Output:   maximum distance found, index of point with max. distance
-----------------------------------------------------------------------------
function: [max_dist , max_index] = findMaximum( first_index, last_index )
     line = generateLine( first_index, last_index )
     max_dist = 0
     max_index = first_index
     for ( i=first_index+1 .. last_index-1 )
         dist = pointLineDistance( Point(i), line )
         if ( dist > max_dist )
             max_dist  = dist
             max_index = i
         end_if
     end_for
     return [max_dist , max_index]
end_function
=============================================================================
```

As already explained, the split-algorithm works recursively i.e. the function calls itself. The initial function call is `split(P(first),P(last))`. In each loop the function `findMaximum()` is called. This function generates the line parameters in `generateLine()` for the `line` given by the two handed over points `first_index,last_index`. In a for-loop the perpendicular distance from each point `Point(i)` to the line is calculated (function `pointLineDistance()`). And continuously the maximum value `max_dist` and the index `max_index` is actualized. Finally these two values are returned. In the super-function `split()` the distance maximum value is used to decide if the line will be split recursively or not. If the distance value exceeds the predefined threshold `DIST_THRESHOLD` the index of the measurement with the largest distance is added to the global data structure `Vertices` by the function `addVertice()` and the function is called twice, once from the previous first point to the maximum and once from the maximum to the last point. The break condition for the recursion is constituted by the question of the distance.

This algorithm constitutes a typical *Divide-and-Conquer*-Algorithm and therefore has the complexity $O(n \log n)$ in the mean case and $O(n^2)$ in worst case where n specifies the number of points.

Merging

The general *Bottom-Up Merging* starts with the first two point and calculates the resulting line. It checks iteratively if the proximate adjacent point lies within a threshold of a relation to the line. This relation can be specified by a euclidean distance from the point to the line, which has to be defined in advance or it can be specified by the covariance of the line, which has to be calculated from the point group which is defining the line. If this threshold is exceeded the previous point group is extracted from the measurement set and the point which exceeded the desired requirement is taken to start a new point group.

In our case we just adapted the aspect of merging adjacent elements but applied it on the segments given by the split algorithm and not directly on the points. We use this procedure to "repair" bad estimated vertices by merging segments where the resulting, united segment fits better to the respective points than the two separated segments.

For the estimation, if one merged segment fits better than the two previous ones, we use the *Maximum Normalized Error* of each of the two separated segments and of the merged segment. If the Maximum Normalized Error (hereinafter called *MNE*) of at least one of the segments is bigger than the MNE of the merged one, the segments will be united, which is trivially obtained by erasing the respective vertex of the vertices given by the split.

The MNE is obtained by:

$$MNE = \frac{e}{D} \tag{47}$$

where D specifies the segment length and e the maximum perpendicular distances $d[i]$ obtained from all points to the respective line with $e = \{\text{argmax } d[i]\}$.

Figure 13 illustrates the case of merging a bad estimated vertex.

To avoid to privilege segments due to their location in the polyline we don't use a direction for the segment checking. Instead of testing from "left to right" or vice versa we calculate all triples of MNE's, i.e. for each adjacent segments we calculate their MNE and the MNE of the potentially merged counterpart. If exist more than one segment-pair which meets the conditions

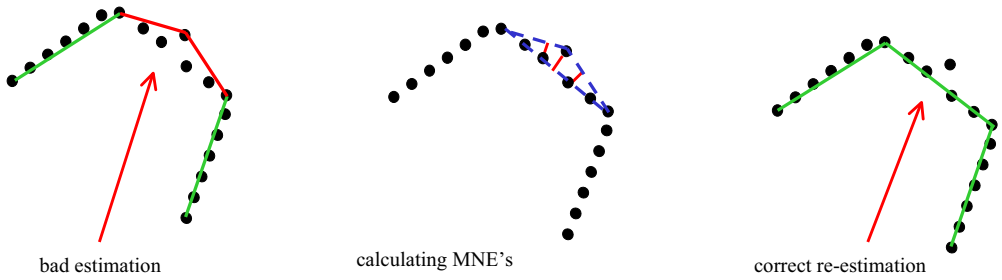

Figure 13: Merging of two segments

to be merged we merge the segments with the biggest difference compared to the merged one. After a merge obviously the MNE-differences of the adjacent segments have to be recalculated. This procedure gets repeated until no segments have to be merged anymore.

The algorithm of the merging-step was implemented as follows:

```
================================================================================
global Points
global s_MNE
global d_MNE
global vertices
--------------------------------------------------------------------------------
Function: iterative merge of segments
Input:    vertices of polyline
--------------------------------------------------------------------------------
function: merge(vertices)
    for(i=1..num_of_vertices - 1)
        s_MNE[i] = getSingleMNE(i)
    end_for
    for(i=1..num_of_vertices - 2)
        d_MNE[i] = getDoubleMNE(i)
    end_for
    merged = 0;
    while(merged == 0)
        d_max_index=getMaxMNEDiff( s_MNE[i] , d_MNE[i] )
        if (d_max_index > 0)
            eraseMaxVertice(d_max_index)
            merged = 1;
```

```
            recalcMNE(d_max_index)
        else
            merged = 0;
        end_if
    end_while
end_function
================================================================================
Function: calculates MNE for given Segments
Input:    number of first vertex of segment
Output:   respective s_MNE
--------------------------------------------------------------------------------
function: s_MNE = getSingleMNE(j)
    line = generateLine( j, j+1 )
    for(i=j..j+1)
        dist[i] = pointLineDistance( Point(i), line )
    end_for
    max_dist=getMaxDist(dist[i]);
    l = getSegmentLength(j,j+1)
    s_MNE = max_dist / l
    return s_MNE
end_function
================================================================================
Function: return index with maxium MNE difference
Input:    all pre-calculated MNE's
Output:   if exist: index with maximum difference if not exist: 0
--------------------------------------------------------------------------------
function: max_vertice=getMaxMNEDiff(s_MNE[i], d_MNE[i])
    for(i=1..num_of_d_MNE)
        if(s_MNE[i] >= s_MNE[i+1]
            diff[i] = s_MNE[i] - d_MNE[i]
        else
            diff[i] = s_MNE[i+1] - d_MNE[i]
        end_if
    end_for
    if ( max( diff[i] ) > 0 )
        return i
    else
        return 0
    end_if
end_function
================================================================================
```

The function `merge()` calculates for all single segments the maximum normalized error `getSingleMNE` as well as for all potenitally merged ones `getDoubleMNE()`. Therefore the line, specified by the two vertices, is gen-

erated `generateLine()` and the distance from each point `Point(i)` to line `line` is calculated and hold in the array `dist[i]`. After computing all distance values the maximum value is acquired by the function `getMaxDist()` and by calculating the length of the segment by `getSegmentLength()` the MNE can be calculated and returned to the super function.

In a while loop the segments with the maximum MNE are obtained by the called function `getMaxMNEDiff()` which uses the data structures `s_MNE[i]` and `d_MNE[i]`. Therefore it calculates for each potentially merged segment the differences to its two "sub-segments" and holds them in the structure `diff[i]`. The difference is calculated by `s_MNE[i] - d_MNE[i]` i.e. if the difference is positive there exist a sub-segment whose MNE is larger then the MNE of its super-segment, indicated by a index > 0, so the segments should be merged. If no sub-segment has a MNE larger than the MNE of its super-segment the function returns 0. So this return value is used for the decision of merging or not. If the decision is YES the segments are merged trivially by erasing the respective vertex `d_max_index` from the vertex data structure. Due to the change on the vertex structure the new MNE's have to be computed on the respective segments and its adjacent neighbour. This proceeds the function `recalcMNE()`.

The break condition for the while loop is the alteration of the vertex structure. If no change was applied the function breaks and exits.

None of the functions contains nested loops and the given loops depend on the number of points because they proceed on the segments whereas one point only belongs to one segment[12]. So the complexity of this algorithms is linear to the number of points n with $O(n)$.

Split & Merge

The above described procedures are applied iteratively. In case of a merge a further split is applied on the affected segments with the intention to optimize the previous results. The whole Split & Merge algorithm is executed iteratively until no modification of the polyline are obtained anymore.

Figure 14 shows the polyline which was obtained after the whole Split & Merge Algorithm.

[12]Except the two vertices

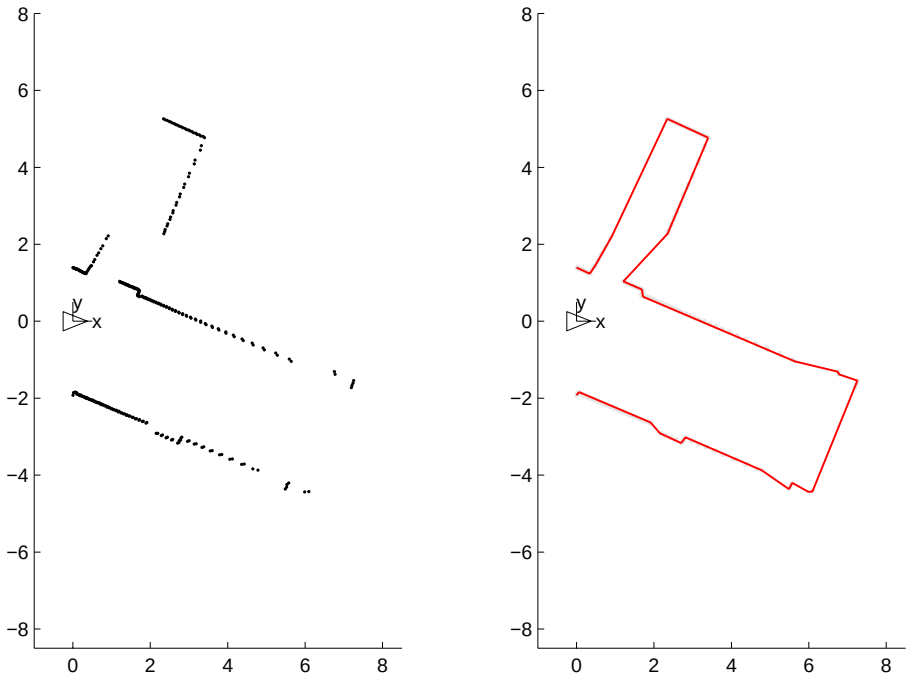

Figure 14: Scan data and generated Polyline after Split&Merge

5.3 The RANSAC - Algorithm

The *Random Sample Consensus* Algorithm, affiliated from [FiBo81], embarks another strategy as the previous described algorithm. It chooses randomly measurements to specify a line and searches the measurements, which are lying within a predefined threshold to this line. The important feature of RANSAC is the incorporation of certain probabilistic characteristics of a scan to determine the run-time length of the algorithm. By knowing the probabilistic characteristics of a scan we have a possibility to apply a kind of hypothesis test and therefore have a probabilistic certainty about the quality of the result.

Its basic principle is not based on a strategy of taking as many measurements as possible and eliminating the "negative compatible" entities but it starts with the smallest possible state vector, in this case a line specified by two points, and increases this state vector by searching and integrating the "positive compatible" observations or measurements.

In general it consists of two major steps:

- Estimation of the number of random tries to search a line, taking into account the predefined probabilities
- The iterative searching of potential segment point groups

Mode of Operation
RANSAC selects randomly two measurements from the given measurement set $\{P\}$. The two chosen points are considered to specify a potential line so we search the measurement subset $\{P_c\}$ which can be considered to be situated close enough to our potential line with $\{P_c\} \subseteq \{P\}$. Let's name this decision threshold *dist_eps*. This threshold should be equal for each randomly chosen line thus an estimation of a covariance given by a transformation of the covariances of the two random points isn't desired since this would lead for each line to a distinguished covariance. Due to the fact that variations of the error tolerances can be considered relatively small compared to gross errors a fixed threshold for each line and each measurement can be considered to be sufficiently appropriate so *dist_eps* can be set arbitrarily to one reasonable and task depending value. The subset $\{P_c\}$ generated by checking on *dist_eps* now is considered to build a potential segment point group if its number of elements $N = |\{P_c\}|$ is greater than a predefined number-threshold *min_num_points*. This threshold obviously should be set at least to the parameter *Minimum Number of Points* (see Chapter. 3.4) which determines the minimum number of measurements which have to define a segment. If N is large enough $\{P_c\}$ is extracted from the previous set $\{P\}$ and the procedure is repeated on the reduced set $\{P_n\} = \{P - P_c\}$. If N is too small $\{P_c\}$ is discarded and the procedure is repeated on the previous set $\{P\}$.

The algorithm has two exit conditions: Either the number of elements in a reduced set is smaller than *min_num_points* so $|\{P_n\}| < min\_num\_points$, or the number of unsuccessful tries exceeds a predefined parameter k.

Estimating Maximum Number of Tries
To decide the k =Maximum Number of Tries we have to introduce several new parameters [FiBo81].

In our case we need 2 points to define a line. Let ω define the probability that any chosen point belongs to a existing segment in the real world model. So the probability that both chosen points belong to a line is ω^2. Thus the

expected value of tries k is specified by $E(k) = \frac{1}{\omega^2}$. On the negative side we have a probability of $(1-\omega^2)$ that we won't find a correct line with one single try. The probability of not finding a line in k attempts is consequently

$$p_{fail} = (1 - \omega^2)^k$$

If we want to ensure with a probability of z that at least one of our randomly chosen point-set defines a existing line the probability of not finding a line is

$$p_{fail} = (1 - z)$$

Therefore we got:

$$(1 - \omega^2)^k = (1 - z)$$

with the number of attempts

$$k = \frac{\log(1 - z)}{\log(1 - \omega^2)} \tag{48}$$

For example if we know that at least the half of all points of the initial set belong to existing real world segments $\omega = 0.5$ and our requirement is a probability of $z = 95\%$ to find at least one existing line we get the following number of tries:

$$k = \frac{\log(1 - z)}{\log(1 - \omega^2)} = \frac{\log(0.05)}{\log(\frac{3}{4})} \approx 10.4$$

So we can assume with a probability of at least 95% that we will find an existing line if we make at least 11 attempts with the given probabilities of the points.

Table 1 provides some values of k depending on ω and z.

Fig. 15 illustrates the procedure of one successful RANSAC Iteration applying randomly chosen points, generating of line parameters, threshold check, extraction and erasing from original measurement set.

The algorithm of RANSAC was implemented as follows:

```
=========================================================================
global Points
global point_groups
-------------------------------------------------------------------------
Function: extracts point groups by RANSAC Algorithm
```

ω \ z	0.50	0.60	0.70	0.80	0.85	0.90	0.95
0.20	16.97	22.44	29.49	39.42	46.47	56.40	73.38
0.30	7.34	9.71	12.76	17.06	20.11	24.41	31.76
0.40	3.97	5.25	6.90	9.23	10.88	13.20	17.18
0.50	2.40	3.18	4.18	5.59	6.59	8.00	10.41
0.60	1.55	2.05	2.69	3.60	4.25	5.15	6.71
0.70	1.02	1.36	1.78	2.39	2.81	3.41	4.44

Table 1: Values for k dependent on ω and z

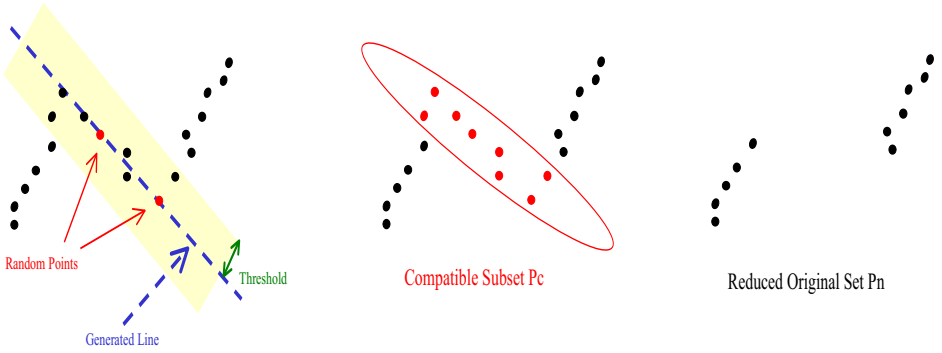

Figure 15: One Successful RANSAC Iteration

```
Input:    probability values
-------------------------------------------------------------------------
function RANSAC(omega, z)
    P_n = Points
    k = calcMaxTries( omega,z )
    cnt = 0;
    while cnt <= k
        line = generateRandomLine(P_n)
        P_c = getClosePoints( line,P_n,DIST_EPS )
        cnt = cnt + 1
        if ( getNum( P_c ) > MIN_NUM_POINTS )
            P_n = P_n - P_c
            addToPointGroups(P_c)
            cnt = 0
            if( getNum( P_n ) < MIN_NUM_POINTS )
```

```
                break
            end_if
        end_if
    end_while
end_function
================================================================================
```

With the subfunction:

```
================================================================================
Function: returns subset with compatible points
Input:    line, superset, threshold
Output:   subset of compatible points
--------------------------------------------------------------------------------
function P_c = getClosePoints( line,P_n,DIST_EPS )
    P_c = 0
    for( i=1..getNum( P_n ) )
        if ( pointLineDist( line,P_n[i] ) < DIST_EPS )
            addClosePoint( P_c,P_n[i] )
        end_if
    end_for
end_function
================================================================================
```

The RANSAC algorithm begins by calculating the maximum number of tries in `calcMaxTries()` specified by the two probabilistic parameters `omega` and `z`. The return value is used for the while loop as break condition. Inside the loop a line is generated by two randomly chosen points from the superset `P_n` by the function `generateRandomLine()` and the line parameters are returned in the data structure `line`. This line is used to obtain the point subset `P_c` which includes the "compatible" points to the line under constraint of the threshold `DIST_EPS`. In the subfunction `getClosePoints()` for each point `P_n(i)` the perpendicular distance is calculated (`pointLineDist()`) and if the distance lies below the threshold the current point is added to `P_c`. In the proceeding while loop now it has to be checked if the size of the compatible points, obtained by `getNum(P_c)`, is large enough to be accepted as potential segment specification. In this case the subset has to be erased from the previous superset of measurements. This is specified by the command `P_n = P_n-P_c`. This successful pass of RANSAC leads to a reset of the counter of unsuccessful tries `cnt`.

RANSAC only has to be proceeded further if the resulting superset of measurements is large enough to build a valid sized sub set at all. So the current size of `P_n` is calculated and compared to the threshold `MIN_NUM_POINTS`

which defines the required minimum number of points to define a line. This threshold as well as the threshold `DIST_EPS` is defined globally as a algorithm criteria.

The algorithm is in its complexity only depending on the runtime fixed factor k therefore the complexity is constant $O(1)$.

5.4 The Hough-Transformation

The *Hough-Transformation* (by [JaScKa95]) is a so called "Voting Algorithm". In this kind of algorithms an initial multiple model is offered and the single measurements vote for the, from their point of view, most probable single model or they vote in a weighted manner where the most probable single model gets the highest voting value. The resulting "Voting histogram" leads to the resulting filtering of the initial model depending on the histogram values.

In the Hough-Transformation the initial model is generated by a parameter transformation from the given representation space into another virtual one, the so called *Hough Space* which serves succeedingly as initial model.

The Hough Space
In our case the measurement data are represented in the 2-dimensional cartesian space with the 2 polar- or cartesian point coordinates. The Hough Transformation transforms this *Point-Coordinate-Space* into the two dimensional *Line-Parameter-Space*.

The Transformation itself is quite simple. With the knowledge that a point appertains to a line with given parameters we have a point function which assigns a y-value to a particular x-value under constraint of the line parameters:

$$y = \frac{\rho - \cos(\theta) \cdot x}{\sin(\theta)}$$

By the transformation from this coordinate function into the line parameter space we get trivially the already known function:

$$\rho = \sin(\theta) \cdot y + \cos(\theta) \cdot x$$

which assigns a ρ-value to a particular θ-value under the constraint of the point parameters.

The similar transformation in the cartesian space results as:

$$y = m \cdot x + c_0$$

and gets transformed into the parameter space by:

$$c_0 = y - m \cdot x$$

where to one slope-value m one y-axis-offset-value c_0 is assigned.

Using the polar line parameter space we assign to a given angle θ a norm ρ. Due to the fact that θ-values of a line contain to a closed set $\theta \in \left]-\frac{\pi}{2}; +\frac{\pi}{2}\right]$ we partition the given set into regular steps $\Delta\theta$ and use the given discrete θ-values with each of the point parameters to calculate the particular ρ-values. In effect to realize a voting algorithm we have to divide the second dimension of the line parameters into discrete values as well. Given the maximum range by the laser with $8100\,mm$ the possible ρ-values are restricted to the set $\rho \in [-8100\,mm; +8100\,mm]$. This set has to be partitioned by a regular ρ-step $\Delta\rho$. Partitioning the two dimensions of parameter space into discrete values we now have a countable number of possible line specifications which can be used as closed Hough-space. To a given constraint parameter pair of a point and one, in our Hough-space specified, θ-value exactly one specified ρ-value is assigned by rounding the resulting parameter accordingly to the defined Δ's. This characteristic builds the base of our Hough-Transformation Voting Algorithm.

Figure 16 shows the discrete assignments of ρ to the given θ_k under constraint of particular point parameters and the resulting lines in cartesian space. The dimension of θ was divided into 4 values with a regular step-width of $\Delta\theta = \frac{\pi}{4}$ with $\theta_{1..4} = \{\, \frac{\pi}{2}\,;\, \frac{\pi}{4}\,;\, 0\,;\, -\frac{\pi}{4}\,\}$. The constraint parameters were given by two points $P_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$ and $P_m = \begin{pmatrix} x_m \\ y_m \end{pmatrix}$.

The Accumulator

Initially a data structure with the dimensional magnitudes according to our discrete Hough-space is created and entirely initialized with 0. This data structure is called *Accumulator* or *Accu*. The transformation is calculated for each constraint parameter-pair P_p with $p \in \{1 \cdots q\}$ and q =number of points and for each possible value of θ_k with $k \in \{1 \cdots l\}$ with $l = \frac{16200\,mm}{\Delta\theta} + 1$. The

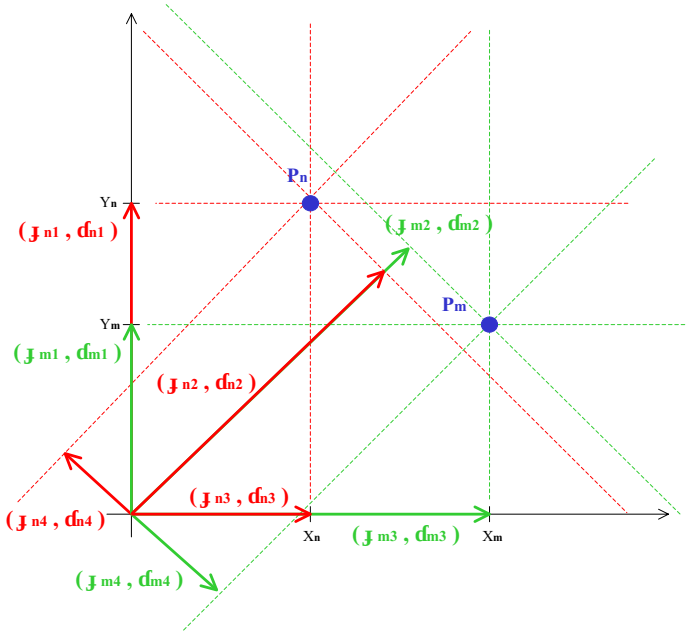

Figure 16: Hough Transformation with resulting Lines and Parameter

resulting transformed parameter pair (ρ_{pk} ; θ_{pk}) specifies a discrete coordinate in the Hough space and the according value in the Accu is increased by one. This can be considered as a "Voting" of one point for one line. After the complete transformation the sum of the entire accu adds up to $q \cdot l$. The coordinate in the Accu with the highest value specifies the line which was voted mostly from all points so this line can be considered to be the most possible line which is specified by the given measurements. Other local maxima as well specify lines with a high possibility to exist.

The definition of $\Delta\rho$ and $\Delta\theta$ depends on the required task. They constitute the "resolution" of the results i.e. the larger the steps are chosen the rougher the transformation will assign points to specified lines since the step-width define the distance between adjacent lines respective angle and distance. Small Δ's make it possible to find separated close situated lines as well as a variation between segments with a small angular difference.

Fig.18 shows the Accumulator after the voting on the original captured scan data displayed in Fig.17.

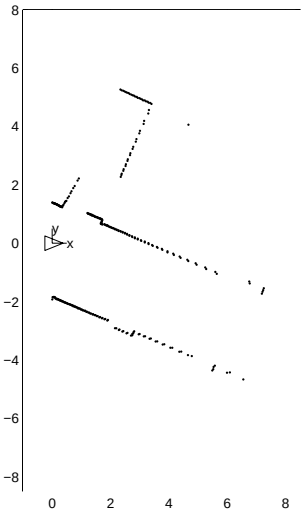

Figure 17: Original data points

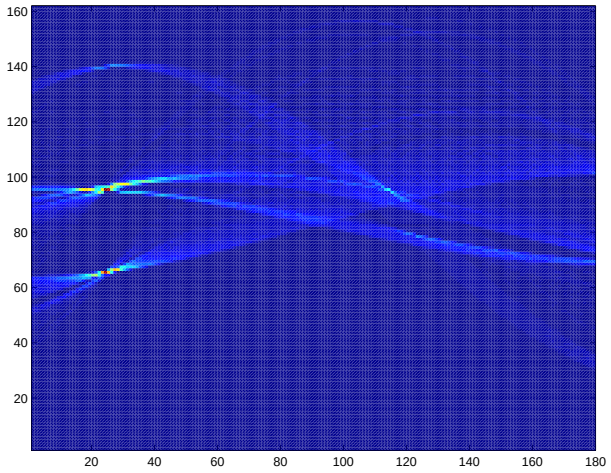

Figure 18: Filled Accumulator

5.4.1 Variation in Maximum Extraction

The maxima should specify the potential lines with a high probability to exist. The problem on this characteristic now is to find the relevant maxima which specify a real existing line because not all local maxima really mirror the real world model.

Therefore we applied two different possibilities to find relevant maxima.

Hough by Revoting

One possibility is only to give relevance to the global maximum due to the fact that the global maximum specifies the line with the highest probability of all the lines found. The goal is to avoid the points which were voting for the maxima voting as well for others non relevant lines. So the line, specified by the global maximum, is generated and all the points which are close enough are extracted and build a resulting point group. This set of point has to be erased from the original measurements and the reduced data is taken for a new Hough transformation. This procedure is repeated until the number of votes of the global maxima is smaller than a given threshold or if the number of left over points is smaller than this threshold. The way of extracting point groups and proceeding on reduced data is equal to the

previous explained RANSAC Algorithm.

The threshold which is used to find close enough points obviously should be chosen accordingly to $\Delta\rho$. If we want to exploit the results of the transformation it should be defined with $\frac{\Delta\rho}{2}$ to keep consistency in our algorithm. Would it be chosen smaller not all points which were voting for this line would be found. In case of a larger threshold points which were voting for other lines would be extracted as well. The threshold for the number of votes of the maximum is defined accordingly to the parameter Minimum Number of Points used to specify a line.

The algorithm of Hough by Revoting was implemented as follows:

```
================================================================================
global Points
global point_groups
--------------------------------------------------------------------------------
Function: extracts point groups by Hough Revoting Algorithm
Input:    predefined deltas of rho and theta
--------------------------------------------------------------------------------
function Hough Revote(delta_rho, delta_theta)
    P_n = Points
    while( getNum(P_n) > MIN_NUM_POINTS )
        accu = calcAccu( P_n, delta_rho, delta_theta );
        [line, num_votes] = getMaxVote(accu);
        if ( num_votes >= MIN_NUM_POINTS )
            P_c = getClosePoints( line, P_n , delta_rho / 2 );
            P_n = P_n - P_c
            addToPointGroups( P_c )
        else
            break
        end_if
    end_while
end_function
================================================================================
```

The Hough-by-Revoting algorithm consists in essence of a loop with the break condition of a check on the superset size obtained by the function `getNum()` compared to the global variable `MIN_NUM_POINTS`. If this would be given a further pass obviously would be useless. The two parameters `delta_rho` and `delta_theta` specify the step width for the respective hough-space parameters. Firstly the Accumulator is calculated in `calcAccu()` under constraint of the given parameters and the current superset of measurements `P_n` and returns the filled data structure `accu`.

```
================================================================================
Function: proceeds hough transformation and
          fills accordingly the Accumulator
Input:    set of Points, deltas of rho and theta
Output:   filled Accumulator
--------------------------------------------------------------------------------
function accu = calcAccu( P_n, delta_rho, delta_theta )
    num_of_theta = (PI / delta_theta)
    for( i=1..getNum( P_n ) )
        for( j=1..num_of_theta )
            theta = PI - (j-1)*delta_theta
            rho = P_n(1)*cos(theta) + P_n(2)*sin(theta)
            index_rho = roundRho(rho, delta_rho)
            accu(i,index_rho) = accu(i,index_rho) +1
        end_for
    end_for
    return accu
end_function
================================================================================
```

Inside the function `calcAccu()` firstly the number of thetas is calculated by dividing the available θ-range. The currently to use θ as well is calculated by using the current index. For calculating the values of `cos(theta)` and `sin(theta)` a lookup table is calculated in advance and used through the programm due to the fact that the exact values of theta are known and the sine and cosine calculation is proceeded many times. The function `roundRho(rho, delta_rho)` takes the calculated, not-discrete ρ-value and rounds it accordingly to `delta_rho` to the appropriate discrete ρ-value.

After returning the filled accumulator the function `getMaxVote()` seeks the coordinate with the maximum voting value and returns the according generated line parameters as well as the number of respective votes. This variable `num_votes` is the basis on the decision of Hough will be proceeded further or not by comparing it to `MIN_NUM_POINTS`. In case of success the compatible points to the current global maximum in the Hough space would be searched. This function is exactly the same as already used in the RANSAC procedure. The constraint parameter in this case is reasonably defined by `delta_rho/2`. As well equally to the RANSAC procedure the measurement subset is erased from the superset `P_n = P_n-P_c` and the subset is added to the global point groups `point_groups` by `addToPointGroups( P_c )`. In case of a too small potential subset i.e. the global maximum obtained too few votes, a break is initiated and the revoting algorithm is finished.

The function to calculate the accu contains a nested for-loop where the numbers of loops depends on the number of currently to use points and the number of θ's. So the complexity is $O(n \times m)$ where n = number of points and m =number of θ's.

Hough by Neighbourship
Another version for the maximum extraction from the accumulator is to calculate it only once and searching all the local maxima. So we're able to avoid the repeatedly calculation of the accu. To find the local maxima a recursive clustering algorithm was implemented which searches all neighbours and the "neighbours of the neighbours" of a maxima. Firstly all the votes of the accu were held in a data structure with the number of votes and the indices, ordered by the number of votes. All indices whose votes are lying below a predefined threshold are erased in advance i.e. their number of votes were set to zero. The current global maxima thus is situated on the top of the structure. Downwards all neighbours are searched recursively, firstly marked and at last erased. The actualized global maxima now stands on top and the procedure is repeated until the data structure is empty and all clusters with a local maxima is found.

The algorithm of Hough by Revoting was implemented as follows:

```
=============================================================================
global Points
global maxima
global point_groups
-----------------------------------------------------------------------------
Function: extracts local maxima of accu by clustering
Input:    predefined deltas of rho and theta
-----------------------------------------------------------------------------
function Hough Neighbourship(delta_rho, delta_theta)
    P_n = Points
    accu = calcAccu( P_n, delta_rho, delta_theta )
    list = generateOrderedIndexList(accu, MIN_NUM_POINTS)
    while ( getNum( list ) > 0 )
        list( 1 ) = marked                 //top element mark 1
        addToMaxima( list (1) )
        markList( 1, list )
        eraseMarked( list )
    end_while
    generatePointgroups()
```

```
end_function
=============================================================================
```

The Hough-by-Neighbourship-Relations algorithm holds the estimated maxima in a global data structure `maxima` with the indices of the accu respectively the line parameters. It starts with a unique calculation of the acumulator `calcAccu()` which is equal to the calculation in the previous described algorithm. Further `generateOrderedIndexList(accu)` generates a list with all elements of the accu with a voting value bigger than `MIN_NUM_POINTS`. In this list the indices and the number of votes are held as well as a marking-column for each entry. The command `list(i) = marked` sets the value in the marking column of element `i` to 1. This signifies the membership of this element to the cluster. The first element is marked automatically by `list(1)=marked` due to the fact that it specifies the local maxima of the cluster which is to build. Hence the first element in the list has to be added to the global list of maxima by `addToMaxima(list(1))`. In the next step the cluster around the first element has to be built.

```
=============================================================================
Function: marks recursively list by checking on neighbourship
          to element elem
Input:    element to check on neighbours, list
-----------------------------------------------------------------------------
function markList(elem , list)
    for(i=2..getNum(list))
        if ( isNeighbourTo(list(i), elem) AND list(i) !=marked )
            list( i ) = marked
            markList(i,list);
        end_if
    end_for
end_function
=============================================================================
```

The function `markList(elem,list)` searches the list from top to bottom and checks on neighbourship with the element `elem`. If a neighbour to `elem` is found the function is called again to search the list on neighbourship on the found element. Obviously the first element definitely is element of the cluster so this element hasn't to be checked. To avoid an endless loop it has to be checked if the found neighbour already is marked. The function `isNeighbourTo(list(i),elem)` checks the indices of the two elements `list(i),elem` and returns 1 if they are neighbours. If all elements

are marked, respectively the cluster is found, the elements have to be erased from the list. This is proceeded in the function `eraseMarked()`. This whole procedure of adding the maximum, marking the cluster and erasing the list elements is executed until the list is empty. Thus all voted coordinates are clustered.

```
============================================================================
Function: extracts point groups according to given line parameter
----------------------------------------------------------------------------
function generatePointgroups()
    for(i=1..getNum(maxima))
        P = getClosePoints( maxima[i],Points,DIST_EPS )
        addToPointGroups( P )
    end_for
end_function
============================================================================
```

In the end the function `generatePointgroups()` extracts the points groups from the entire measurement data `Points` by means of the guarded line parameters in `maxima`. The procedure is equally to the already presented point group extraction in previous algorithms with the difference that the extracted point groups aren't erased from the superset. So the possibility of measurements to contain preliminary to more the one point group is given.

The recursive clustering algorithm is called for almost each element of the list and proceeds a check on each other element of the list. The list represents all coordinates which were voted more times than a specific threshold. The major variable which specifies the number of coordinates is the number of θ's. So the complexity of the clustering algorithm is quadratic $O(n^2)$ where n specifies the number of θ's.

5.5 The EM-Algorithm

The *EM-* or *Expectation Maximization*-Algorithm uses an iterative approach to improve a pre-defined virtual model. We are using a variation of the EM-Algorithm whose fundamentals are given by [DeLaRu70].

The basic principle is to use a iterative procedure to change a given model by probabilistic relations between the given measurement data and the model which in the end is used to extract the resulting segments. The algorithms consists of two major steps:

- **The Expectation-Step** (hereinafter the E-Step) and
- **The Maximization-Step** (hereinafter the M-Step)

The E-Step calculates the expectation of the measurement data to the model[13] and the M-Step maximizes the expectations by changing the model parameters under constraint of the previously calculated expectations. [LCBT01] introduces a 3D approach of a similar problem.

In the following I want to introduce the fundamentals of the algorithm referring to our problem.

The virtual Line Model

The model we are using is a finite collection of 2D lines which in the end is used for the segment extraction. Our used model is denoted Θ and it consists of J lines so each single model line is called Θ_j with $j \in \{1 \cdots J\}$, so our model is specified by

$$\Theta = \{\Theta_1 \cdots \Theta_J\} \tag{49}$$

Each single model specifies a line in vector representation so the single model parameters are $(\alpha_j\,,\,\beta_j)$ where α_j specifies the *Unity Normal Vector* to the line Θ_j and β_j the *Norm* of the *Normal Vector* from the origin of the coordinate system to Θ_j. Hence each line is defined by

$$\Theta_j = (\alpha_j\,,\,\beta_j) \quad \in \quad \mathbb{R}^3 \times \mathbb{R} \tag{50}$$

Due to the fact that α_j is a orthogonal unit vector to the line its given: $\alpha_j \cdot \alpha_j = 1$ where "$\cdot$" signifies the *Inner Product*[14].

This representation gives us the distance relation function:

$$|\alpha_j \cdot z - \beta_j| = d \tag{51}$$

where z denotes a measurement given by 2 parameters in cartesian coordinates. Thus the points z which are elements of the line Θ_j are specified by

$$\alpha_j \cdot z = \beta_j \tag{52}$$

[13] *Model* specifies a set of *Lines* which hereinafter also will be referred to with *Single Models*

[14] Also called *Dot-Product* or *Scalar Product*

The given Measurement Model
Our measurement data are represented in 2 dimensional space so each measurement is specified by $z_i \in \mathbb{R}^2$. The complete set of measurement data will be denoted with Z and the number of obtained measurements I so:

$$Z = \{z_i\} \qquad \text{with} \qquad i \in \{1 \cdots I\} \tag{53}$$

Contemplating a probabilistic relation between the measurements and the line model we have a probability of $p(z_i|\Theta_j)$ for each measurement z_i to the single model line Θ_j. Based on the assumption of *Gaussian Measurement Noise* the error distribution from a measurement z_i to its closest Θ_j is given by the normal distribution

$$p(z_i|\Theta_j) = \frac{1}{\sqrt{2\pi\sigma^2}}\ e^{-\frac{1}{2}\frac{(\alpha_j \cdot z_i - \beta_j)^2}{\sigma^2}} \tag{54}$$

where σ specifies the variance parameter.

The above given equation only is relevant for measurements which in reality can be assigned to our virtual model. We have to introduce a model to which measurements are assigned if they don't belong to a existing model, e.g. outliers. The goal is to assign each measurement to a single model line though in reality this isn't the case. Therefore we introduce a *Phantom Model* to which such un-assignable measurements are assigned. Let's denote this model with Θ_*. We give this phantom model a uniform error distribution with $p(z_i|\Theta_*) = 1/z_{max}$ where z_{max} specifies the maximum range of our laser sensor. Assuming that $z_i \in [0; z_{max}]$, which here always is the case, we can rewrite the phantom normal distribution as

$$p(z_i|\Theta_*) = \frac{1}{\sqrt{2\pi\sigma^2}}\ e^{-\frac{1}{2} ln \frac{z_{max}^2}{2\pi\sigma^2}} \tag{55}$$

This specifies the above chosen uniform distribution due to the constant exponent.

These distributions are the basis for the E-Step of the EM-Algorithm.

The Log-Likelyhood Function
The Log-Likelyhood function specifies a description of likelyhood between the measurements and the model. This function is the basis of optimization because it describes how close the measurements and the model are lying to

each other estimated by the squared perpendicular distance from a point to a line[15].

To define the likelyhood function we have to introduce a new set of variables the so called the *Correspondence*. The correspondence exists for each measurement to each single model and the phantom model. Let's denote it with c_{ij} and c_{i*}. In effect the correspondences c_{ij} are binary variables which adopt the value 1 if the measurement z_i corresponds to the j'th single model Θ_j. If this is not the case it adopts the value 0. One measurement only can be assigned to *one* single model that is to say to the model for which its probability to correspond is the highest. If the measurement is not caused by any of the given single models the phantom model correspondence is set to 1 . The given correspondence vector of all correspondences of one measurement i therefore is denoted by

$$C_i = \{c_{i*}, c_{i1}, c_{i2}, \cdots, c_{iJ}\} \tag{56}$$

So the correspondence vector for one measurement sums up to exactly 1 since each measurement is caused by exactly one single model θ_j.

Assuming the knowledge of the correspondences we can rewrite the general probability of one measurement under constraint of its correspondence and the model as follows:

$$p(z_i|C_i,\Theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left[c_{(i*)} ln\frac{z_{max}^2}{2\pi\sigma^2} + \sum_{j=1}^{J} c_{(ij)} \frac{(\alpha_j \cdot z_i - \beta_j)^2}{\sigma^2}\right]} \tag{57}$$

Obviously this formula is partially redundant due to the fact that only one correspondence value is 1 the rest will be 0 and these parts in the exponent then are redundant.

By making the correspondence explicit in the measurement model we are now able to calculate the *Joint Probability* of one particular measurement z_i along with its correspondences C_i. Assuming that all correspondences of all $J+1$ single models are equally probable in absence of measurements we have:

$$p(z_i, C_i|\Theta) = \frac{1}{(J+1)\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left[c_{(i*)} ln\frac{z_{max}^2}{2\pi\sigma^2} + \sum_{j=1}^{J} c_{(ij)} \frac{(\alpha_j \cdot z_i - \beta_j)^2}{\sigma^2}\right]} \tag{58}$$

Under the assumption of independence in measurement noise we can compute the likelyhood of *all* measurements Z and their correspondences

[15]Evaluated in the exponent of the normal error distributions

$C = \{C_i\}$ simply by multiplying them:

$$p(Z, C|\Theta) = \prod_i \frac{1}{(J+1)\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left[c_{(i*)} ln \frac{z_{max}^2}{2\pi\sigma^2} + \sum_{j=1}^{J} c_{(ij)} \frac{(\alpha_j \cdot z_i - \beta_j)^2}{\sigma^2}\right]} \tag{59}$$

This function happens to be maximized. Due to the fact of the inconvenient product in this formula a common practice is to maximize the *Log-Likelyhood* instead:

$$\ln p(Z, C|\Theta) = \sum_i ln \frac{1}{(J+1)\sqrt{2\pi\sigma^2}} - \frac{1}{2} c_{(i*)} ln \frac{z_{max}^2}{2\pi\sigma^2} - \sum_{j=1}^{J} c_{(ij)} \frac{(\alpha_j \cdot z_i - \beta_j)^2}{\sigma^2} \tag{60}$$

The logarithm is strictly monotonic hence the maximization of the log-likelyhood corresponds to the maximization of the likelyhood though the maximization of the log-likelyhood is more convenient due to the sum in the equation.

All the above given equation compute a joint over the model *and* the correspondences. All we are interested in are the model parameters and since the correspondences only are interesting regarding the determination of the most likely model Θ all we want to achieve is to estimate the *Expectation values* of the log-likelyhood taking into account all correspondences C. So the expectation of the log-likelyhood is given with:

$$E_C\left[\ln p(Z, C|\Theta)\right] =$$
$$= E_C\left[\sum_i ln \frac{1}{(J+1)\sqrt{2\pi\sigma^2}} - \frac{1}{2} c_{(i*)} ln \frac{z_{max}^2}{2\pi\sigma^2} - \sum_{j=1}^{J} c_{(ij)} \frac{(\alpha_j \cdot z_i - \beta_j)^2}{\sigma^2}\right] \tag{61}$$

Finally we want to factor in the expectation of the correspondences and since the expectation is linear the log-likelyhood function under the constraint of the expectation values of the correspondences results with:

$$E_C\left[\ln p(Z, C|\Theta)\right] =$$
$$= \sum_i ln \frac{1}{(J+1)\sqrt{2\pi\sigma^2}} - \frac{1}{2} E[c_{i*}] ln \frac{z_{max}^2}{2\pi\sigma^2} - \sum_{j=1}^{J} E[c_{ij}] \frac{(\alpha_j \cdot z_i - \beta_j)^2}{\sigma^2} \tag{62}$$

Now we have the log-likelyhood function of *all* measurements to *all* single models and the phantom model for unassignable measurements under constraint of the expectations of the correspondences. This is the basis for the later maximization step.

Expectation-Maximization
The Expectation Maximization consist of the two mentioned steps. The calculation of the expectation values of the correspondences $E[c_{ij}]$ and $E[c_{c*}]$ of the fixed measurements to a given model $\Theta^{[n]}$ and the maximization of the log-likelyhood function with the calculated expectations regarding the model parameters. This optimizes the model iteratively until the point of maximum likelyhood and therefore convergence in the optimization of the model parameters. The algorithm starts on an initial model which can be generated randomly or defined systematically.

Let's first introduce the E-Step.

The Expectation Step
To the given model $\Theta^{[n]}_{j=1\cdots J} = \{\Theta^{[n]}_1, \cdots, \Theta^{[n]}_J\}$ and the measurements $Z_{i=1\cdots I} = \{z_1, \cdots, z_I\}$ we search the expectations $E[c_{ij}]$ and $E[c_{c*}]$ for all i, j.

Assuming a uniform prior over the correspondences *Bayes-Rule* gives us directly the expectations by:

$$\begin{aligned} E[c_{ij}] &= p(c_{ij}|\Theta^{[n]}, z_i) \\ &= \frac{p(z_i|\Theta^{[n]}, c_{ij})\ \ p(c_{ij}|\Theta^{[n]})}{p(z_i|\Theta^{[n]})} \\ &= \frac{e^{-\frac{1}{2}\frac{(\alpha_j \cdot z_i - \beta_j)^2}{\sigma^2}}}{e^{-\frac{1}{2}\ln\frac{z_{max}^2}{2\pi\sigma^2}} + \sum_{k=1}^{j} e^{-\frac{1}{2}\frac{(\alpha_k \cdot z_i - \beta_k)^2}{\sigma^2}}} \end{aligned} \tag{63}$$

Similarly the expectation for the phantom model is given by:

$$E[c_{i*}] = \frac{e^{-\frac{1}{2}\ln\frac{z_{max}^2}{\sigma^2}}}{e^{-\frac{1}{2}\ln\frac{z_{max}^2}{2\pi\sigma^2}} + \sum_{k=1}^{j} e^{-\frac{1}{2}\frac{(\alpha_k \cdot z_i - \beta_k)^2}{\sigma^2}}} \tag{64}$$

As already mentioned before, the probability and thus the expectation for measurement i and model j depends directly on the term $\frac{(\alpha_k \cdot z_i - \beta_k)^2}{\sigma^2}$ which specifies the squared mahalanobis distance (chap. 3.3.1) from point i to line j under constraint of the variance parameter σ.

Thus we have the expectation values and are able to use them for the optimization of the model $\Theta^{[n]}$.

The Maximization Step

The given expected log likelyhood in equation 62 is supposed to be maximized and to extract optimized model parameters (α_j, β_j) to build model $\Theta^{[n+1]}$. Obviously only one part of (62) is dependent on the model parameters so we are able to extract the particular part for the maximization. The term to maximize results with

$$\sum_i -\frac{1}{2} \sum_j E[c_{ij}](\alpha_i \cdot z_i - \beta_i)$$

so the required task can be achieved by *minimizing*

$$\sum_i \sum_j E[c_{ij}](\alpha_i \cdot z_i - \beta_i) \tag{65}$$

A given constraint for the minimization of (65) is that $\alpha_j \cdot \alpha_j = 1$ since only then the resulting vector will be a normal vector to a line and thus only then a correct result will be obtained. So the M-Step can be characterized as a *quadratic optimization problem* under *equality constraints* of some variables, in our case α_j.

The solution for this problem is given ([LCBT01]) by the introduction of the *Lagrange-Multiplier* λ_j with $j = \{1 \cdots J\}$. So the Lagrange function results as:

$$L = \sum_i \sum_j E[c_{ij}](\alpha_i \cdot z_i - \beta_i)^2 + \sum_j \lambda_j \alpha_j * \alpha_j \tag{66}$$

Minimization is obtained in case that the derivation of the variables are equal to 0.

$$\frac{\partial L}{\partial \alpha_j} = 0 \qquad \text{and} \qquad \frac{\partial L}{\partial \beta_j} = 0 \tag{67}$$

So deriving the Lagrange function and using the normal-constraint of α_j we obtain a linear equation system *for each single model* Θ_j:

$$\sum_i E[c_{ij}](\alpha_i \cdot z_i - \beta_i) z_i + \lambda_j \alpha_j = 0 \tag{68}$$

$$\sum_i E[c_{ij}](\alpha_i \cdot z_i - \beta_i) = 0 \tag{69}$$

$$\alpha_i \cdot \alpha_i = 1 \tag{70}$$

From (69) we obtain for β_j:

$$\beta_j = \frac{\sum_k E[c_{kj}]\alpha_j * \cdot z_k}{\sum_k E[c_{kj}]\alpha_j} \tag{71}$$

and substituted back into (68) we get:

$$\sum_i E[c_{ij}] \left(\alpha_j \cdot z_i - \frac{\sum_k E[c_{kj}]\alpha_j \cdot z_k}{\sum_k E[c_{kj}]\alpha_j} \right) z_i - \lambda_j \alpha_j = 0 \tag{72}$$

By integrating out α_j we get the form:

$$\alpha_j \sum_i E[c_{ij}] \left(\cdot z_i - \frac{\sum_k E[c_{kj}] \cdot z_k}{\sum_k E[c_{kj}]} \right) z_i = \lambda_j \alpha_j \tag{73}$$

So this results as a linear equation of the type:

$$A_j \cdot \alpha_j = \lambda_j \alpha_j \tag{74}$$

where each A_j is a 2×2 matrix with the solution elements:

$$a_{st} = \left(\sum_i E[c_{ij}] z_{is} z_{it} \right) - \left(\frac{\sum_i (E[c_{ij}] z_{it} \sum_k E[c_{kj}] z_{ks})}{E[c_{kj}]} \right) \tag{75}$$

for $s, t = 1, 2$.

By solving this matrix we get two *Eigenvalues* $\lambda_{1,2}$. The 2 solution vectors obviously have to be *Eigenvectors* where the eigenvector with the bigger eigenvalue constitutes a vector defining the line. The eigenvector with the smaller eigenvalue therefore defines the Normal Vector to the solution line. Applying the smaller eigenvalue into the solution matrix we get the appropriate values for α_j and substituting them back into equation 71 we obtain the norm of the Normal Vector to the line. So we now have a solution to our maximization problem and the Expectation Maximization Algorithm is completed.

This procedure of calculating once the current expectations and calculating once the optimized log likelyhood function, and consequently the new parameters of the model, constitutes one closed EM-pass.

The model Θ
The problem with the model is the alleged knowledge of the model magnitude. EM assumes prior knowledge of the correct model but only in the fewest cases this may be the case. So it's necessary to implement as well possibilities to change the model complexity. On one side we have to be able to introduce new models on the other side we have to be able to erase models which are considered to be unsupported by the measurements.

Terminating unsupported single models
There are several cases when it's desired to delete single models.

One trivial case occurs when the solution of a vector returns the zero vector. This can happen e.g. when a model initially is too far from any point to be considered to correspond to any measurement. So expectations to this model are converging to zero and therefore the solution matrix will be a zero matrix. The resulting solution vector as well will result a zero-vector. Obviously a zero vector can't be considered to be an optimization, so this single model has to be erased. This case only occurs if the model initially is defined without knowledge and dependence of the points. This case is avoided if the initial model only consists of lines which are specified by two existing measurements e.g. by initiating the starting model by choosing randomly points to define the model lines. If this is not the case i.e. the possibility of zero-solution vectors is existent each pass of EM the zero vector models have to be erased.

If two model lines converge towards the same values, EM would be proceeded for both single models though the result would be the same. The problem results in the fact that the two single models are "competing" for the same measurements and therefore an optimal result would be avoided. Let's assume a segment which is specified by several noisy points. It would be desired that though they are noise affected they should "vote" for the same model line. In case of two lines competing from two directions to these points the segment points would be "split" into the points which in particular are slightly closer to one or respective to the other line. Figure 19 shows a virtual example of two lines competing for one segment. The segment consists of 8 Points and each line is assigned to 4 points. Thus in the end both 4-point groups would be erased if the minimum number of lines is specified with 5.

To avoid this we define a threshold which is applied on the model parameters

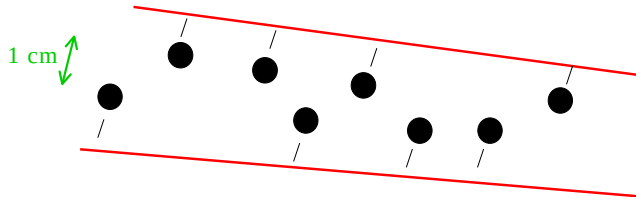

Figure 19: Two model lines competing for the same segment

and specifies the status of two equal lines. In case of two too likely models one of them will be erased. This procedure has to be applied during the iterative EM-Step i.e. after each pass the models are checked on likelyhood.

Another case of erasing a model line can be applied after convergence of the whole model. In case that one line is voted as preferred line from too few points it will be erased. In this case the previously often applied threshold *Minimum Number of Points* is applied again. It's obvious to see, that if the number of points which would vote for this line is lower than this threshold it will be erased definitely afterward in the postprocessing during the check of number of points. So to avoid a measurement voting accidentally for a line e.g. by a bad initialization we check after convergence of the model on the number of "line member points". Therefore we check the numbers of maximal expectation values for each single model line. In case of a lower number than the given threshold it's going to be erased.

Adding single models

Our approach adds after each detected convergence a specified number of randomly generated model lines. This procedure assumes that a model after convergence is *not* entirely defined and can be improved continuously. The procedure is to select randomly two measurements form the current data set and build the respective line. This line is added to the current model and the EM algorithm is applied repeatedly. In case that a added model line is very likely to a already existing one, the newly added single model will be erased quickly due to similarity, respectively a bad estimated line will be erased quickly due to absence of significant votes, except the votes of the two points it was generated of. Hence only relevantly added lines will affect the results.

Another approach would be a systematic adding of single models i.e. seek-

ing significant lines to add, respectively to the result, though this procedure would use a lot of time to find the according points to build them e.g. by a complete measurement-model distance check. Due to the effective procedure of terminating unsupported lines the approach of randomly generated lines was found superior.

Exit Condition for EM

The general condition to stop EM is to reach a predefined number of iteration steps for EM itself or to reach a specified number of found convergence.

An additional condition for a previous algorithm break can be a check on result quality. The expectation values for the phantom model may serve to decide if a model is defined well or not. This results from the fact that measurements will vote more heavily for the phantom model if they don't correspond to our current line model.

One possibility is to use the expectation values of all measurements to the phantom model and computing the mean. If this falls below a predefined threshold the current model can be considered to be describing the measurement very well. The expectation mean for the phantom model constitutes a characterization of correspondence of all measurements to the model.

Another possibility is to check on the number of points which are voting for the phantom model. Therefore we search the measurements with the maximum expectation value for the phantom model and count them. If this number falls below a predefined threshold the number of well described measurements of the whole set can be considered to be sufficiently high so a break can be initiated.

The above described possibilities to estimate the quality of the model can't be applied exclusively due to the fact that the data scans are affected by noise and the measurements obviously are unpredictable. Though e.g. the mean of the phantom expectation will converge to a lower value as at the start of the algorithm, a predefined fixed "quality threshold" can't be considered to be signifying a general quality description since the measurement sets and the models are varying considerably. So for one scan a threshold could be specifying an adequate result characteristic whereas for another scan it could be inapplicable are could lead to a extensive number of EM iterations.

So the thresholds were implemented and defined sufficiently low to make sure their application only in case of very good results. The general operated exit condition constitutes the number of iterations.

The algorithm of Expectation-Maximization was implemented as follows:

```
================================================================================
global Points
global point_groups
global model
--------------------------------------------------------------------------------
Function: Applies EM algorithm
--------------------------------------------------------------------------------
function EM()
    model = initializeModel( MODE, NUM )
    em_cnt = 0;
    conv_cnt = 0;
    while (1)
        em_cnt = em_cnt + 1
        old_model = model
        [ E , Ep ] = E_step( model )
        model = M_step( E )
        model = terminateEquals( model , EQU_DIFF )
        if ( checkConvergence( model , old_model, CONV_DIFF ) )
            conv_cnt = conv_cnt +1;
            model = terminateMinVoted( model , NUM_MIN_VOTES)
            Ep_mean = calcEpMean(Ep)
            num_Ep_max = calcNumEpMax(Ep)
            if ( ( Ep_mean < MIN_EP_MEAN)            OR
                 ( num_Ep_max < MIN_EP_NUM_MAX)      OR
                 ( em_cnt <  MAX_EM_ITERATIONS)      OR
                 ( conv_cnt <  MAX_EM_CONVERGENCES)      )
                break
            else
                model = addRandomLine(model, NUM_NEW_LINES)
            end_if
        end_if
    end_while
    generatePointgroups(model)
end_function
================================================================================
```

EM starts with the initialization of the model `initializeModel()`. The parameter `MODE` specifies the mode how the model should be generated. There are two modes: `RAND` and `SYS`. In RAND-mode from the measurement two randomly chosen points are used to build the line. This line is added to the model data structure `model`. The parameter `NUM` specifies the number

of lines which have to be generated on initialization. In the mode `SYS` a systematic model is generated where the whole scan area is covered with systematic situated lines. Entering into the loop we have to save the model `old_model` parameters which are neccessary afterwards to check with the new generated model on convergence. The function `E_step(model)` calculates the expectation values by means of the current model and returns the model expectations in `E` and the expectation for the phantom model in `Ep`. The function `M_step(E)` calculates the optimized model parameters by means of the expectation and returns the new model parameters in `model`. The algorithm for these two functions is illustrated in pseudo code afterwards. After optimization the model lines which are equal are searched and erased in `terminateEquals()` where equality holds if the differences between the model parameters are falling below a predefined threshold `EQU_DIFF`. `checkConvergence()` returns `1` if the previous model `old_model` and the currently maximized model `model` are considered to be equal. This consideration depends on the parameter differences and the decision is specified my means of the threshold `CONV_DIFF`. If convergence is detected we eliminate the models which are voted by too few measurements `terminateMinVoted()` determined by the threshold `NUM_MIN_VOTES`. `calcEpMean(Ep)` and `calcNumEpMax(Ep)` compute the values for the later break-condition check where they are used with the counters for convergence and iterations and the predefined thresholds. If no break condition is reached the function `addRandomLine()` adds to the model `NUM_NEW_LINES` randomly generated lines and the next iteration is proceeded. If the break condition is reached the loop is quitted and the current model is used to extract from the entire measurement data `Points` the respective point groups. This happens in the function `generatePointgroups(model)`. This procedure already was described in the above presented algorithms. The general algorithm consists of a while-loop which is proceeded maximum `MAX_EM_ITERATIONS` or `MAX_EM_CONVERGENCES` times and doesn't depend directly on input data so the complexity is constant $O(1)$.

```
========================================================================
Function: calculates Expectations for points to given model
Input:    model
Output:   Expectation values for model E,
          Expectation values for phantom model Ep
------------------------------------------------------------------------
function [ E , Ep ] = E_step(model)
```

```
    for ( i = 1..getNum( Points ) )
        Ep(i) = calcE( Point(i) , phantomModel );
        for( j = 1..getNum( Model) )
            E(i,j) = calcE(Point(i), model(j) )
        end_for
    end_for
    return [ E , Ep ]
end_function
============================================================================
Function: calculates maximization for each model
          under constraint of given expectations (E)
Input:    expectation values E
Output:   optimized model
----------------------------------------------------------------------------
function  model = M_step( E )
    for( j = 1..getNum( Model) )
        model(j) = calcMaximization ( model(j) )
    end_for
    return model
end_function
============================================================================
```

In the E-Step for each measurement and for each model the expectation value (equation 63) is calculated and for each point the expectation for the phantom model (equation 64). The complexity depends on a nested loop where n specifies the number of points and m the number of models so $O(n \times m)$.

In the M-Step for each model the parameters are optimized as defined in equation 75. So the complexity is linear $O(m)$ depending on the number of models m.

Figure 20 shows an example of the optimization of an initial systematic model with 4 lines and the resulting parameters after the first expectation and maximization step on real scan data. The arrows indicate the assignment from starting lines and their respective optimizations.

In this chapter the algorithms were presented. We now know the basic principles and the general modes of operation. For each algorithm the major parameters were introduced and the influence their variation can have on the results. The results are available for all the algorithms in the same format: as a set of point groups. In the post processing step these point groups will be used to generate the appropriate segments.

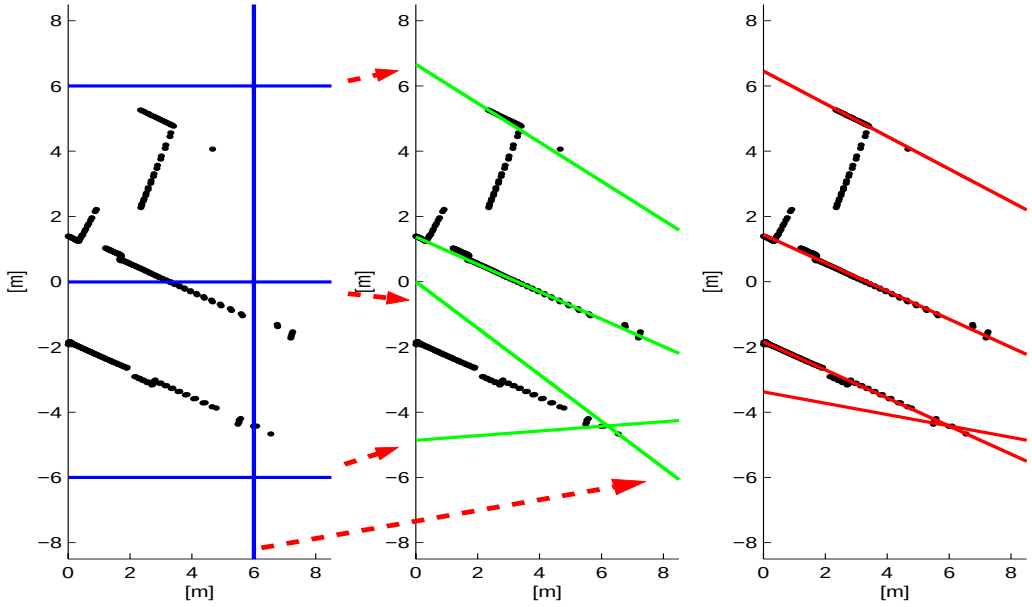

Figure 20: Initial systematic model , first optimization and results after first convergence by Expectation Maximization

6 Postprocessing

As explained before the segmentation algorithms are designed to find point groups which specify potential segments. All the algorithms provide the same resulting format which consist of point groups found by the different algorithm procedures. Thus the extraction of the resulting segments from the given point groups are equal for all algorithms.

In this chapter I want to describe the procedures which are applied to transfer the given point groups into initially potential output segments. These segments are checked on compliance of the criteria given by the predefined segment parameters presented in chapter 3.4. On complete compliance the potential segments are accepted as resulting segments. After a last check on overlapping or on partial or entire duplication and respective erasure the finally resulting segment-set is created. This whole procedure can be summarized as *Segment Postprocessing*.

6.1 Segmentation

The given point groups were built regarding the geometrical or probabilistic relation between points and virtual lines. This gives us a certainty about perpendicular deviation from a collective line. So far we don't have any information about the geometrical relation between the points themselves. One criteria we require to define a continuous segment is the distance between two adjacent[16] points. Therefore the segment criteria *Inter-Segment Distance* has to be applied. To build a continuous segment points only are allowed to have a certain maximum mutual distance. If this is not the case, e.g. a door in a wall, the segment is split between this two adjacent points. Therefore we check the distance on each point to his adjacent neighbour where adjacent signifies the posterior captured measurement. In case of a split the whole set of already checked points of this segment is separated and specifies a smaller, independent segment. The procedure of sequential point-to-point distance check gets further applied for the rest of the segment. So a multiple separation of one segment is possible. The used distance parameter was defined *distance relative* due to the fact that afar captured adjacent points automatically have larger mutual distances. By comparison of the real world, the captured data and the final virtual model we came to the conclusion that

[16] *Adjacent:* hereinafter will denote two sequentially captured measurements

good results were obtained with a Inter-Segment Distance $\approx 0.05\frac{m}{m}$.

Another criteria, a continuous point group has to comply, is the number of measurements between two adjacent points which aren't belonging to the appropriate point group. The associated parameter was denoted with *Number of Invalid Points*. This characteristic can be used due to the indexed measurements. If the difference of indices of two adjacent points is larger than the predefined threshold the segment is split between this two points. A very small value e.g. 1 would signify that every outlier which wasn't filtered in advance, would cause a perhaps unintentional separation. A value chosen too large would ignore a possible existing segment. Reasonable results were obtained with a threshold *Maximum Number of Invalid Points* between 2 and 4.

The corresponding algorithm was implemented as follows:

```
================================================================================
Function: checks one segment on parameters Inter-Segment Distance
          and Maximum Number of Invalid Points
Input:    point group
Output:   vertices where the segment has to be split eventually
--------------------------------------------------------------------------------
function vertices = segment( point_group )
    for ( i = 1..getNum( point_group )-1 )
        dist = getPointPointDistance( point_group[i] , point_group[i+1] )
        if ( dist > INTER_SEG_DIST )
            addVertice(vertices , i)
            continue
        end_if

        ind_dist = index.point_group[i+1] - index.point_group[i];
        if ( ind_dist <  MAX_NUM_INV )
            addVertice(vertices , i)
        end_if

    end_for
end_function
================================================================================
```

The function gets the point group which holds in `index.point_group[i]` the corresponding index of point `i`. `getPointPointDistance()` obviously calculates the distance between the two input points. Exceeds the computed distance the threshold `INTER_SEG_DIST` the current index is added to the

structure `vertices` which holds the positions where the segment has to be split eventually. This structure afterward has to be applied to split the point groups on the appropriate positions. If the distance exceeds the threshold no further check on invalids has to be applied between this two points therefore the `continue`. The check on invalids is applied trivially on the difference of the indices and the same procedure is applied as previously described.

The algorithm has a linear complexity $O(n)$ where n describes the number of points.

A third criteria we have to apply is the *Minimum Number of Points*. As already explained before, we consider a segments only to be defined by points if the number of points is big enough. This is achieved quite trivially by counting the points of each point group after the above explained split. If the size of a point group is smaller than the predefined threshold the segments is erased completely from our set of point groups. The application of this parameter ensures us of the fact that a segment has to be evaluated by a certain number of measurements. If this is not the case we can't be sure if the segment wasn't "captured" accidentally by noisy sensor data. Thus we ignore the respective measurements.

Satisfactory results were obtained with a minimum number of points between 5 and 10 points.

6.2 Line Generation

The so far obtained point groups can be considered to be "complete" i.e. they won't be split anymore. The proximate step is to generate the line which would be specified by all points of one group. In the literature (e.g. [Cast98], [JaScKa95] or [LCBT01]) this step also is referred to as *Line Smoothing*.

Therefore we realized two implementations:

Total Regression The Total Regression calculates a "best fit straight line" by estimating the least squared error from all given points to the line.

Extended Information Filter The *EIF* calculates the actual state vector (the line) by integrating all given points simultaneously.

6.2.1 Total Regression

The Total Regression "estimates" the line with the minimized square errors from the points to the line[17] in x and y-direction.

The line parameters are obtained by:

$$\theta = \frac{\arctan \frac{a}{b}}{2} \qquad \rho = (\bar{y} \cdot \cos\theta) - (\bar{x} \cdot \sin\theta) \tag{76}$$

using the **Arithmetic Mean**,

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \qquad \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{77}$$

the **Sum of the squared Errors**,

$$s_{xx} = \sum_{i=1}^{N} (x_i - \bar{x})^2 \qquad s_{yy} = \sum_{i=1}^{N} (y_i - \bar{y})^2 \tag{78}$$

the **Product of the summed Errors**,

$$s_{xy} = \sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y}) \tag{79}$$

and the resulting **Regression Parameters**

$$a = 2 \cdot s_{xy} \qquad b = s_{xx} - s_{yy} \tag{80}$$

The Total Regression has to be computed in two passes due to the fact that for the calculation of the sum of the squared error or the product of the summed error the arithmetic means have to be known. So we have to calculate in one closed loop the mean values and in another loop the furthermore parameters.

6.2.2 Extended Information Filter

Contrary to the total regression the application of the *EIF* can be done directly in one loop (see 3.3.4). However the EIF needs a distance calculation for each measurement to a initial line and requires as well the additional calculation of the covariance of the resulting line.

[17] Thus Total Regression is also known as *Method of the minimal Squares*

Comparison
Both of the two above described algorithms for line smoothing were implemented.

The Total Regression can be considered to be easy to comprehend and to implement. It is a fast an effective algorithm to calculate a line of best fit whereas the EIF-algorithm calculates additionally the covariance directly as well as it provides an easy integration of a further point (though in our case this doesn't happen). Figure 21 shows the differences between a line estimated by the Total Regression and the Extended Information Filter on virtual test data.

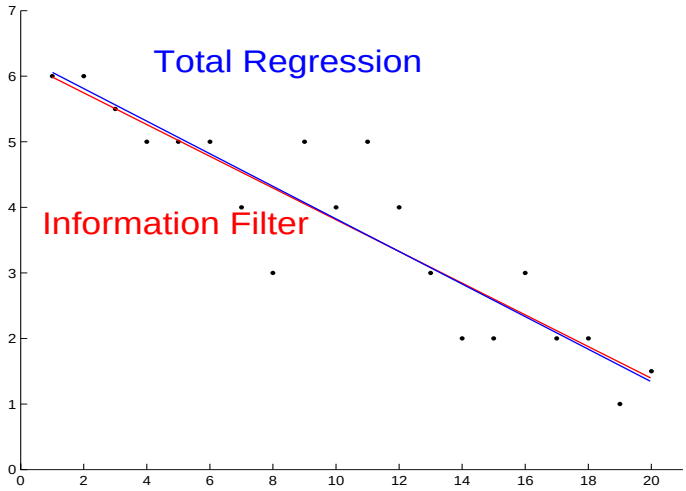

Figure 21: Differences between a line estimated by Total Regression (blue) and the EIF (red)

The differences in this case aren't very big. In this case the EIF was initialized by the line built by the the first and the last point of the measurement array and only was calculated once, thus no iteration was proceeded. If the obtained results would be used iteratively to integrate more further elements, the Information Filter and the Total Regression would converge towards the same results.

A comparison of the elapsed time neither produced a measurable difference so the application of each algorithm is equivalent.

6.3 Avoiding Overlappings

Due to algorithms characteristics or noisy sensor data in some cases it can occur overlappings of segments or even that one small segment lies "embedded" within a another bigger one. From the geometrical point of view this doesn't make sense since a sensor isn't able to detect segments which are lying behind another so the result definitely should be considered as wrong. A virtual example of this case is illustrated in Figure 22.

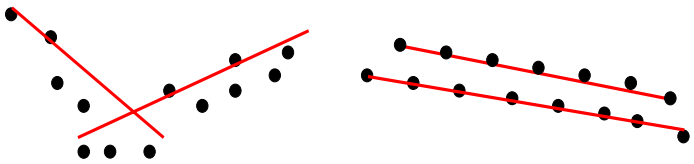

Figure 22: Overlapping or embedded positioned Segments

Obviously this cases have to be avoided or filtered.

To detect overlappings the indices of the point groups are used. If the index of an endpoint of one segment lies within the index set of another segment point groups an overlapping is detected. There are three possible procedure to patch overlapping segments:

Merging It's tried to merge the point groups under constraint of the given segment criteria.

Cutting Parts of point groups have to be erased with the objective to loose as few information as possible.

Erasing Segments have to be erased if no other procedure is applicable

The first and mostly desired possibility is to *Merge* overlapping point groups. This is desired since there won't be loss of information, respectively measurements, regarding the results. This possibility is only applicable if the resulting point groups comply the given segment criteria. Therefore the point groups are combined and a regression line is generated (see chap. 6.2) . If

the perpendicular distance of all points to the line lies below the specified threshold *Point-Line Distance* we consider the merged segment as valid and add it to the set of correct point groups.

If a merge is not possible due to the above explained reasons we try to *Cut* the segments in a way to obtain valid resulting segments. Again the objective is to loose as few points as possible.

Thus we seek the overlapping index position of two segments and estimate which part of which segment has to be erased. Evidently each segment contains one part which lies "embedded" within the other so one of these two parts have to be erased. The potential results have to be contemplated in both cases of partial segment erasure. The first objective is *not* to loose one of the segments completely. This can happen if the number of total points in a point group would be falling below the threshold *Minimum Number of Points* after deleting parts of the segment. If this only would happen to one segment only the affected part of the other segment will be erased.

If none of both segments would be erased the smaller of the two affected segment parts will be deleted.

If none of the above given possibilities are applicable one of the segments has to be *erased* completely. In this case the smaller segment will be deleted.

The general algorithm results as follows:

```
=============================================================================
Function: checks on overlaps and clears them out in case
Input:    point groups
Output:   adjusted point groups
-----------------------------------------------------------------------------
function point_groups = clearOverlappings( point_groups )
    num = getNum(point_group)
    for ( i = 1..num - 1 )
        for ( j = i+1..num )
        if ( overlap( i , j ) )
            if ( merge( i , j , DIST_EPS ) )
                num = num - 1
                j = j - 1
            else_if ( cut( i, i+1,MIN_NUM_OF_POINTS ) )
                num = num - 1;
                j = j - 1;
            else
                erase ( i , j )
```

```
                num = num - 1
                j = j -1
            end_if
        end_for
    end_for
    return point_groups
end_function
================================================================================
```

The major part of the algorithm `clearOverlappings()` consists of a nested for loop where each point group `i` is checked on overlappings with each other point group `j`. The initial number of point groups is obtained by `getNum(point_group)`. If an overlapping is detected firstly it tries to merge them by `merge()`. If a merge isn't possible regarding the global criteria `DIST_EPS` the function returns `0`. The function returns `1` if the segments could be merged. The merge consists of deleting the two point groups from the data structure `point_groups` and adding the merged point group at the same position `i`. Afterwards the number of point groups has to be decreased `num-1`. To detect multiple overlappings the current index has to be decreased as well `j-1` to check again on the same segment and the remaining segments.

A similar procedure is applied with a the cut function `cut()`. The cut-function checks the segments internally on `MIN_NUM_OF_POINTS` and seeks the optimal solution in case of a cut. Again after a cut the number of point groups have to be decreased as well as the loop index to carry on with the correct indices.

The function `erase()` follows the same procedure with the difference that it seeks the appropriate segment to erase.

Finally the adjusted data structure `point_groups` is returned.

The algorithms consists of two nested for-loops so the complexity is squared with $O(n^2)$ where n specifies the number of previous given point groups.

6.4 Endpoint Acquirement

After estimating the line and clearing overlappings we have to generate the segment by calculating the segment endpoints. The segment endpoints have to be acquired depending on the given group of points. The first and last point in the given ordered group are not in every case the required endpoints since they are not always the "farthest end" of the segment. So we have to

find the optimal points of the group which should be used to calculate the segment endpoints.

Therefore we calculate a reference point by taking the middle of the first and the last point in the list and search the farthest points in each of the two direction of the segment. The points with the largest distances are taken to acquire the perpendicular line points which are now defining the segment.

The procedure of the Endpoint Acquirement is illustrated in Figure 23.

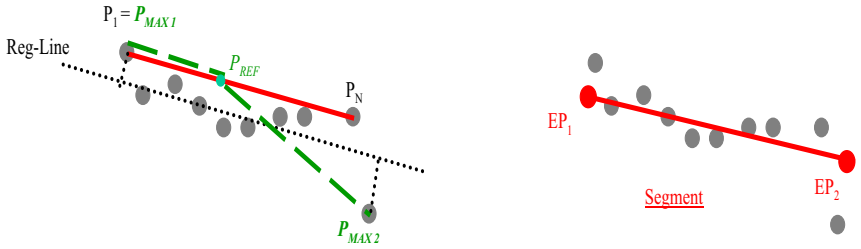

Figure 23: Endpoint Optimization and Endpoint Acquirement

In most of the cases the above described *Endpoint Optimization* doesn't provide big deviations on the segment length (in general in dimensions of mm to a few cm). In case of computational time problems it can be turned off by parameter.

6.5 Length Check

The final parameter on which the segments are checked is the Segment Length. The above calculated segment endpoints are used to calculate for each segment its specific length. If this value falls below the predefined threshold *Minimum Segment Length* the segment is ignored completely. This value depends on the required "resolution" of the segments. With a value chosen too big, small segments which could be specified by many measurements would be erased, whereas a value too small in cases leads to many eventually undesired segment particles. In our case it was set to value around $0.1m$ so we're able to detect e.g. the depth of a door.

Figure 24 shows the finally obtained result on real sensor data (on the left)

and the resulting preprocessed, segmented and post processed segments (on the right).

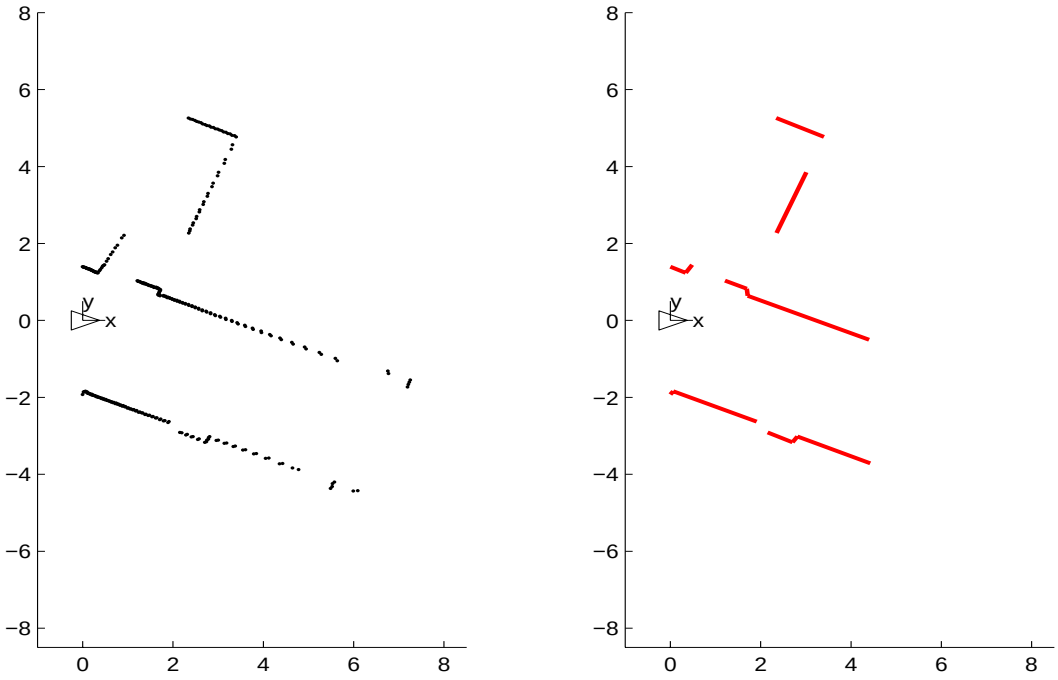

Figure 24: Original Scan data and the resulting Segments

In this chapter the complete postprocessing procedure was presented. Thus the extracting of segments from 2 dimensional sensor data is complete. We started on the raw sensor data and proceeded a preprocessing with filtering to prepare the data. Afterwards we applied the different algorithms for the point group extraction and finally we post processed the data and generated the resulting segments.

In the next chapter I want to point out variations in the algorithm parameters as well as modifications in the algorithms themselves. First of all we will compare the differences and results of the specific algorithm variations. A comparison between the different algorithms will follow in the chapter afterwards.

7 Analysis and Algorithm-Variations

In this Chapter we will present the results of the algorithms with variations in the specific and relevant algorithm parameters and with different modes of operations.

The objective is to show which possibilities exist to tune and optimize the algorithms as well as to show where are the limits of the algorithms. In some cases the parameter choice will depend on the task, the algorithm has to accomplish, so we will present the different possibilities of regarding the requirements.

On one side we will try to optimize the algorithms regarding computational time and on the other side towards the quality of the results. The final objective should be to find a reasonable compromise between velocity and quality.

All time measurements were obtained on a PC System with a 1,7 GHz CPU and 1GB RAM under usage of a Windows 2000© Operation System.

7.1 Usage of uncertainty characteristics

With the means of the described procedures in chapter 3.3 we have the possibility of integrating the uncertainty characteristics of measurement data. By applying data uncertainty in distance measurements the question is formulated as: "Is it possible that calculated distances of geometrical entities are lying below a specified threshold taking into account the uncertainty of the entities?". A very trivial problem is e.g. if we want to know if a point belongs to a line though the calculated euclidean is $\neq 0$ regarding the uncertainty of the point and the line. The specified threshold in this case is $= 0$. The procedure consists in obtaining the uncertainties of the point and the line (if it exists), calculating the Mahalanobis distance and applying a hypothesis test on the result under constraint of a desired probability.

Such calculations could be applied on point-line distances as well as on point-point distances (Fig. 25) so in our case we could apply it on procedures like the outlier filtering, calculation of inter segments distances or on point-line distances used e.g. in the Split & Merge algorithm, therefore this algorithm was used to obtain comparable time measurements.

The measurements were obtained by the MATLAB Profiler Module so the results will be presented in percentage values to avoid confusions to the

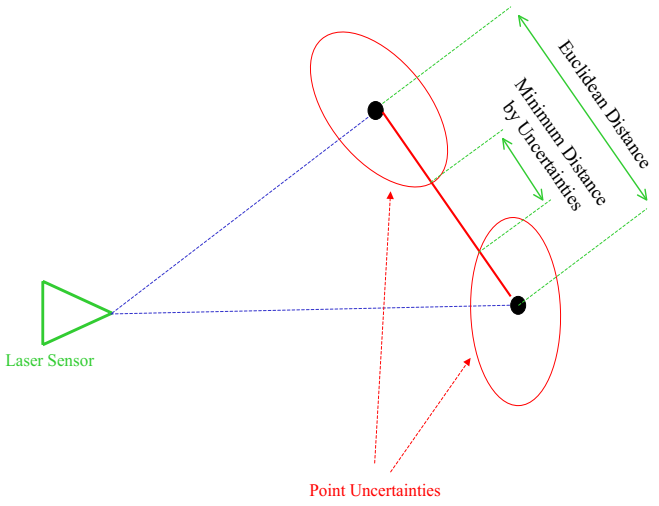

Figure 25: Application Mahalanobis Distance

later in this chapter presented time measurements obtained under C/C++.

The outlier filtering by using the Mahalanobis distance produced an average augmentation of 17.83%, whilst the additional needed time for the inter segment check resulted in average with 8.62%. In this cases the mahalanobis distance was used with the respective thresholds. The application of the Mahalanobis distance on the point-line distance in the Split & Merge algorithm produced for the distance checks an augmentation mean of $\approx 46\%$.

In effect the usage of the Mahalanobis distance led to an average additional time for the complete algorithm of 6.82%.

The differences in the results were insignificantly small.

The changes for the point-point distance consisted in additional points, which normally would have been erased, or in a not-separation of a segment due to a accepted distance. The number of additional points varied between 0 and 4 whereby the average number was smaller than 1.5. A segment which wasn't separated due to usage of the Mahalanobis distance almost didn't occur.

The usage for the point-line distance in the splitting step neither produces significant differences in the results for the generated polyline.

Alltogether the usage of the uncertainties of the geometrical entities only produced insignificant differences in the results but led to an augmentation of computational time. Hence we only used the Mahalanobis distance for the above given procedures in the MATLAB Prototypes where they were off turnable by parameter.

In our case the characteristic of uncertainty of sensor data and respectively the statistical and systematical error of the laser was bore in mind for a reasonable choice of the algorithm parameters. In this thesis I will use the term *reasonable choice* of parameters with the signification of taking into account the required task (e.g. separating a door from a wall), general environment a priory knowledge (e.g. that walls normally can be considered to be straight) and the given laser error tolerance.

7.2 Split & Merge

The complete procedure of Segmentation by using the *Split & Merge* algorithm is illustrated in Figure 26.

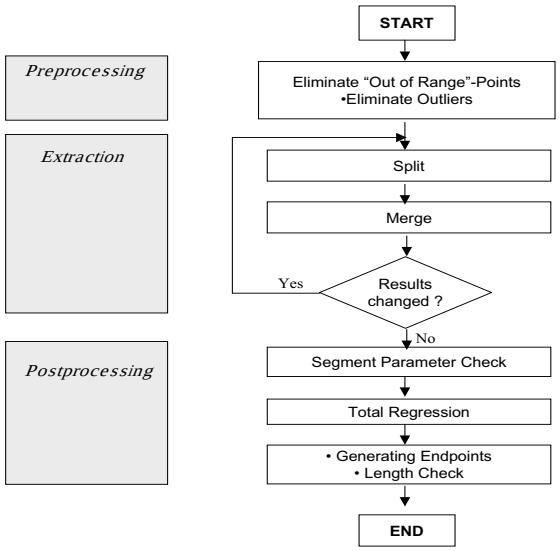

Figure 26: Flow Diagram of the Split&Merge Algorithm

Split & Merge uses all points of the given measurement data so it's necessary to proceed an outlier filtering (chap. 4.2). But due to the fact that it uses each point once (except the vertices) and groups the points systematically we can be sure that no segment overlapping (chap. 6.3) will occur.

Variation of the Distance Threshold

The most important parameter constitutes the used threshold for the split-decision. The smaller the threshold the deeper the recursion will proceed and respectively the more segments will be estimated. Figure 27 shows the split polyline and the resulting segments with a distance threshold of $0.08m$ (case 1).

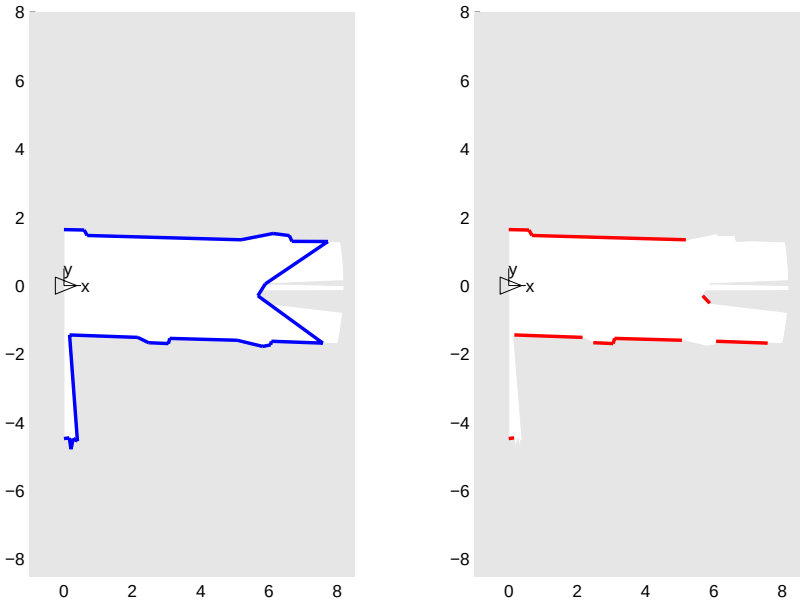

Figure 27: Split & Merge with a distance threshold of $0.08m$

It's easy to see that the "resolution" is very high where even the depth of a door is detected. As well the application of the segment parameters is good to identify. For example the small segments on the left of the doors on the bottom are erased due to the number of points. The wall segments on the outer right on the top were erased due to the inter segment distance and the minimum segment length.

The polyline after splitting and merging consists of 24 segments with a resulting final number of 10 segments.

Figure 28 shows the results with a threshold of $0.5m$ (case 2). The resolution

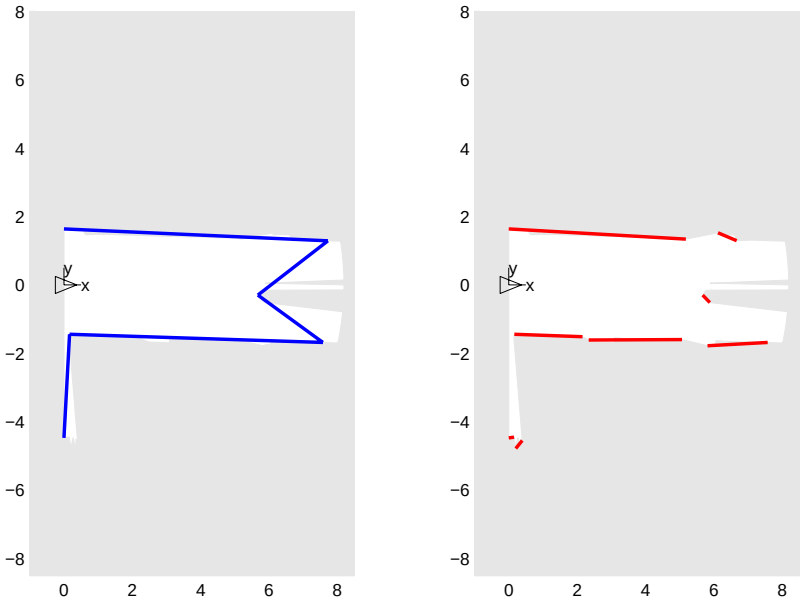

Figure 28: Split & Merge with a distance threshold of $0.5m$

now is very "rough". On the top and on the bottom the doors and walls were estimated as one segment. The polyline was separated into 5 segments with a resulting number of 8 segments.

In both cases the segment criteria were set equally:

- Inter Segment Distance: $0.05m$ (distance relative)
- Minimum Number of Points: 5
- Maximum Invalid Gap: 2
- Minimum Segment Length: $0.1m$

The measured computational time, listed in Table 2, present the absolutely

	EPS = $0.08m$		EPS = $0.5m$	
	time [sec]	$\% - value$	time [sec]	$\% - value$
Total mean time	0.01372	100.00%	0.00455	100.00%
Split & Merge	0.01261	91.93%	0.00365	80.22%
Preprocessing	0.00106	7.80%	0.0009	19.72%
Postprocessing	0.00002	0.15%		

Table 2: Mean computational times of different Split & Merge algorithms

measured elapsed time for the different parts of the algorithm with the relative percentage values regarding the complete elapsed time. Figure 29 shows the elapsed times for the different Split & Merge passes in a chart.

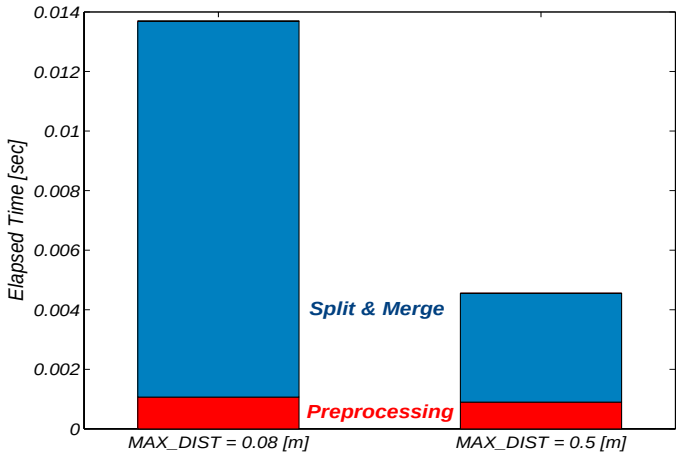

Figure 29: Calculational Time Chart of different Split & Merge passes

The much faster pass for a bigger distance threshold, due to the flatter recursion, is easy to recognize. The preprocessing obviously is equally for the two cases and needs a constant computational time whilst the postprocessing in both cases only plays a redundant role with 0.15% and 0.0%[18].

[18]Measurements obtained by C-function clock(). In case of a very short computational time the resolution of clock() may be too low thus 0.0 seconds measurements as output are possible and point to a very short procedure time

Altogether the whole algorithm (case 2) only needs 33% compared to the first case.

The varying threshold values against the time (Figure 30) show that from a distance of $\approx 0.2m$ and above the relative time savings are getting smaller and smaller. This signifies that the depth of recursion stays more or less the same whereas the depth differences between a threshold value $0.06m$ and $0.2m$ are comparatively immense. This gives us a estimation of the general distance distribution in the scans. Whilst the differences above $0.2m$ won't cause a great difference in time yet small variation beneath $0.2m$ may lead to big differences in the algorithm running time.

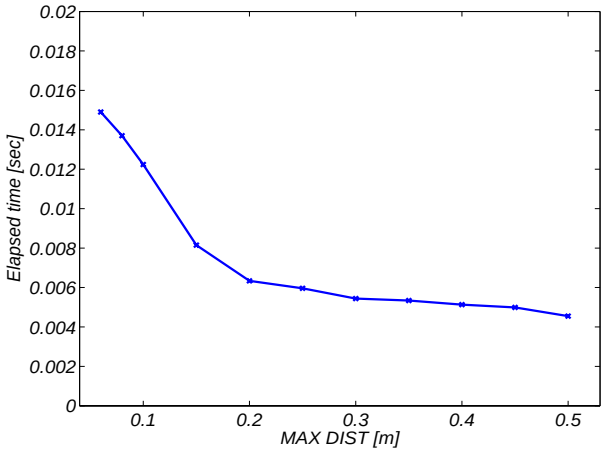

Figure 30: Elapsed time over varying thresholds

Variation of Merging Procedure

As a variation of the merging procedure we tried different criteria for the decision: merging or not. Beside the already described *Maximum Normalized Error* we implemented as well a merging due to angular deviation of adjacent segments. If the angular difference between two adjacent segments fell below a certain threshold the segments were merged.

However this approach doesn't cause large effects due to the fact that large segments with a small angular difference nevertheless lead to relatively large distant deviations. In an iterative splitting step this deviation would cause the algorithm to split the merged segment repeatedly. So the only

positive effect were obtained for a very small threshold ($\approx 1°$). This didn't lead to a big optimization thus the angular merging was implemented with a parameter switch and was "turned off" for the tests.

7.3 RANSAC

A Flow Diagram in Figure 31 illustrates the sequential parts of the complete RANSAC-Algorithm.

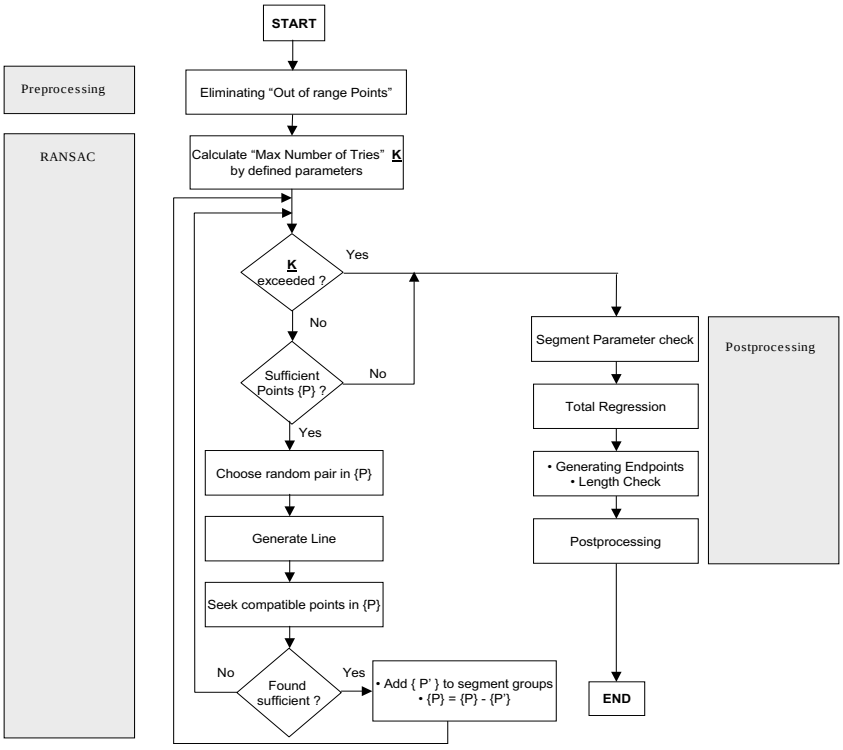

Figure 31: Flow Diagram of RANSAC Algorithm

RANSAC doesn't proceed an outlier filtering in advance whereas due to his characteristic of generating lines randomly and the respective group extraction an overlapping check has to be applied.

The specification of the criteria now has an important role to play because RANSAC yet uses the segment criteria during the algorithm and not only

in the end for the segment determination. Thus an inconvenient definition wouldn't only affect the quality of the results but also the proceeding of the algorithm.

The probability-parameters have to be defined more or less arbitrary due to the difficulties of measuring the quality of the results. For a comparison the results have to be estimated reasonably into "better or worse" and this always leaves a margin of subjective interpretation.

For the determination of the distance threshold the interpretation should be considered as the "resolution" of the results. If e.g. the detection of a door as a door is required, and the "depth" of a door lies around $0.1m$, a value above this value doesn't make sense since the door and the wall would be detected as one segment. On the other side a value deep below the error tolerance of the sensor neither is reasonable due to the noise affected sensor data. Thus the value of the distance threshold normally was set between $0.05m$ and $0.1m$.

The number of points which are found for the determination if a subset can be considered as a segment as well has to be chosen arbitrary. Obviously it should be at least equal or higher as the defined minimum number of points in a segment. For example a 5 point segment with 1 meter distance to the sensor can have a minimum length of $0.034m$ but this would certainly be erased due to the minimum length of segments whereas a 5 point segment with a distance of 8 meters would have a minimum length of $0.28m$ and would be considered as a normal segment. So with a too small value close segments with few points would be erase whilst small segments further away wouldn't be detected. Would it be too high smaller segments lying close to the sensor would be ignored.

[FiBo81] provides another point of view. The number has to be determined sufficiently high to avoid the detection of a segment which doesn't exist in the real world model. Let y be the probability of a point belonging to a not-existing segment, t the defined threshold and n the number of points to define a subset (in our case n = 2) so y^{t-n} should be minimized. Obviously y can be considered as smaller than ω because there should exist more points belonging to a segment than "Outliers". So with a $\omega = 0.5$ and therefore $y < 0.5$ and $t - 2 = 5$, hence $t = 7$, the possibility to find a segment which doesn't exist is smaller than 5%.

We want to ensure with a probability of z, that at least one of our randomly chosen point set defines a existing line so this value should be chosen as high as possible taking into account the computational time. In our case we varied this value normally between 70 and 95%.

In the tests the parameter ω was varied over a larger range to show the differences in results and computational time. Figure 32 shows an example of significant differences which can occur though in general the results are more likely.

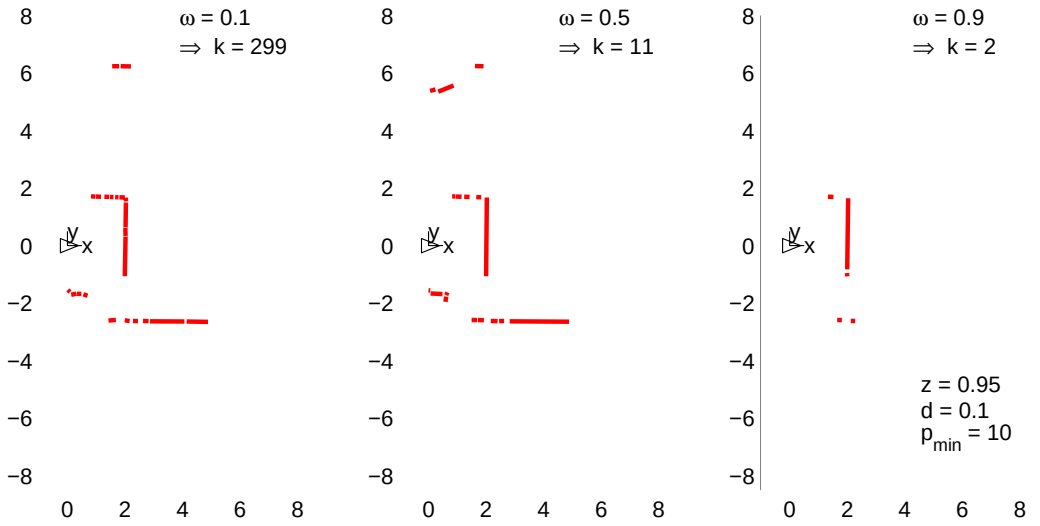

Figure 32: Different results on same data with varying ω's

The mean computational times which were obtained over several passes[19] with varied parameters are plotted in Figure 33. The plot confirms the expeced results. The trend shows that a higher z-value leads to a larger execution time due to the higher demanded probability of correct estimation. The differences in varying ω-values are even more obvious. In case of a high probability for a point being valid the maximum number of tries will decrease immensely and accordingly the used time.

[19]Each value is the acquired average of *Elapsed Time* obtained by ≈ 2000 passes

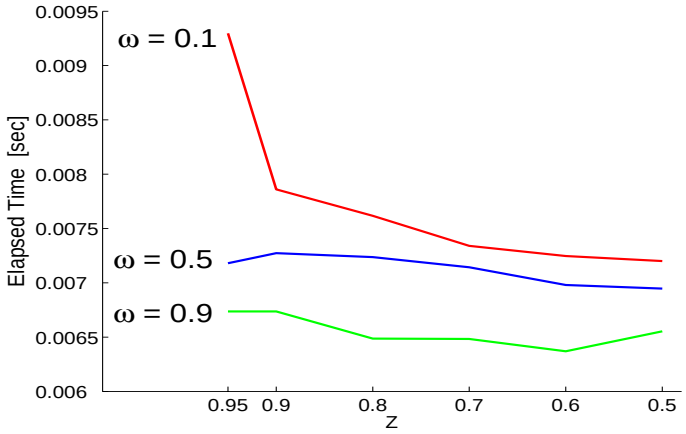

Figure 33: Average computaional time obtained with varying ω

The obtained maximum differences in time by varying probability parameters account for $\approx 29\%$ with values for the maximum number of tries with $1 \leq k \leq 299$. But though the relative differences seem to be large the maximum absolute difference accounted with ≈ 0.003 sec i.e. less than 3 milliseconds.

Characteristics of RANSAC

The big advantage of the RANSAC algorithm is its velocity. The number of maximum tries doesn't depend on the number of points but only on the probabilities which have to be defined in advance. The number of points only is relevant for finding the compatible points to a estimated line.

The quality of RANSAC of using randomly chosen points causes on the same scan varying results as well as different computational times. Figure 34 shows an example obtained in several executions with equal data and parameters.

The differences are easy to detect. Segments sometimes are separated into particles as can be seen in the outer left scan. The corner on top should be closed like in the middle scan. The door on the bottom wasn't even partially detected and the wall segment right to this door results very short.

In general the left scan can be considered to be coincidentally bad estimated whilst the middle and the right scan can be considered to be a standard

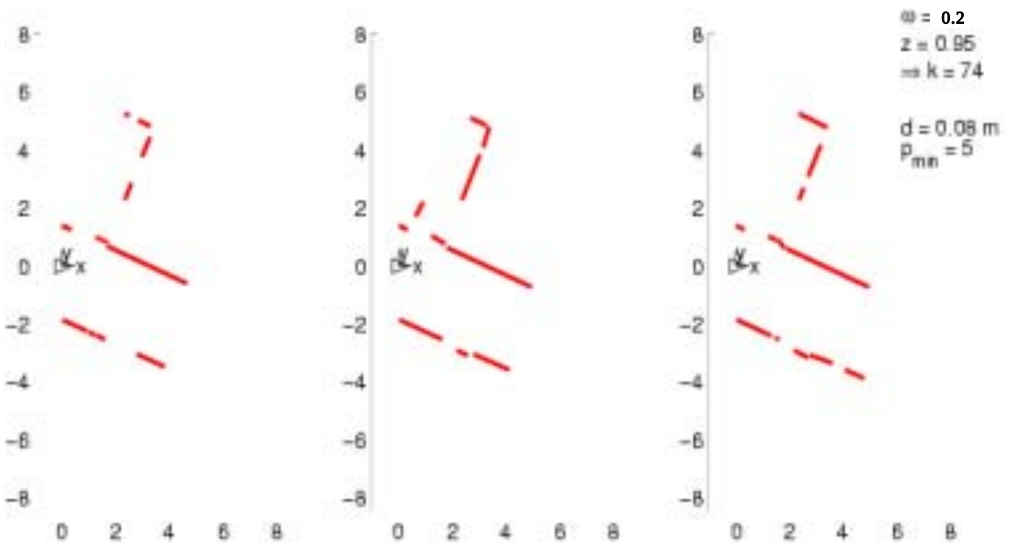

Figure 34: Varying results on same data obtained by RANSAC

result of RANSAC.

Proceeding 50 passes over a 289-Scan series provided the following time results:

- maximum difference: $\max \Delta = 0.00062\text{sec} \equiv 8.61\%$
- mean: $\bar{s} = 0.0072\text{sec}$
- sample variance: $\sigma^2 = 1.7516 \cdot 10^{-8}$

A further problem of RANSAC is the proceeding on a reduced data set after erasing an estimated subset. Figure 35 shows an example where RANSAC found enough compatible points to consider it being a line and erased the points from the data set. It's obvious that the erased points shouldn't build a segment with the set RANSAC estimated. The so produced gaps can cause a separation of the real world segment or even cause RANSAC not to find it at all. This characteristics also can cause the separation of one real world segment into many little segments which normally have slightly different angles after the total regression (segment particles).

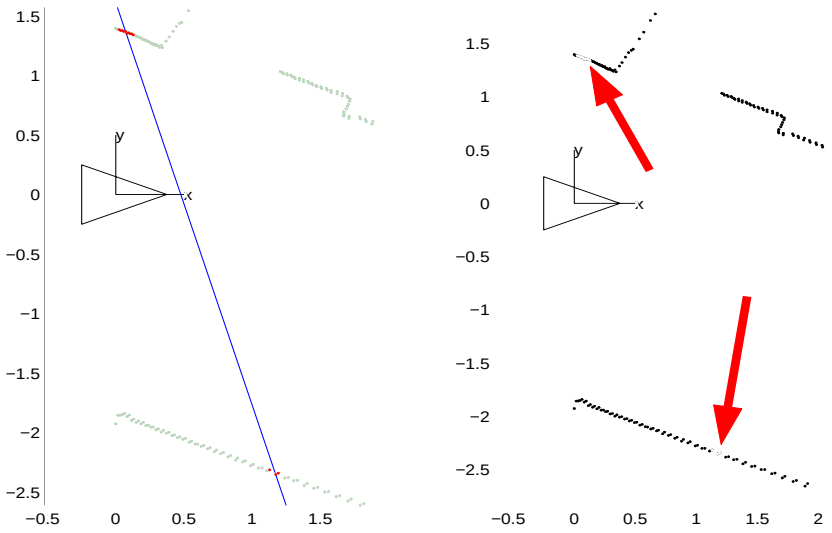

Figure 35: Gaps after erasing points from data set by RANSAC

This behaviour can be improved by increasing the minimum number of points but therewith small real world segments wouldn't be detected anymore. A better way is to increase the probability of finding the most relevant real world segments firstly. Therefore we implemented another version of RANSAC which proceeds *a split of the whole data set* in advance.

7.3.1 Extended RANSAC with Split

The goal was to generate point groups where the probability to find the more relevant segments first is very high. Thus we avoid e.g. to be two different, unlike walls of a room in one group. The splitting algorithm is similar to the algorithm explained in chapter 5.2. The only difference consists in the choice of the distance threshold which in this case should be chosen much bigger. Successively the RANSAC algorithm has to be applied on each group. The probability parameter ω then can be defined significantly higher so the maximum number of tries will be significantly lower whereby the additional computational time, caused by the splitting, can partly be compensated.

Figure 36 shows the split groups which were build on the given data with a threshold of $2\,m$.

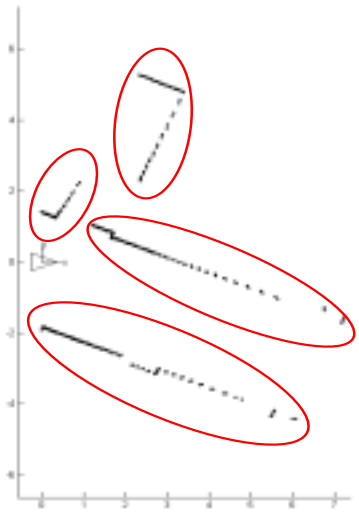

Figure 36: Split groups before proceeding RANSAC

Comparison of different RANSAC versions To compare the previous explained algorithms we tried to vary the parameters in a way to obtain more or less likely results. Figure 37 shows a series of equal data sets and the results once obtained by the original RANSAC algorithm (above) and the RANSAC algorithm with a split in advance (bottom).

It shows slightly better results for the second way. On the first two scans it's recognizable that the segments are less separated, corners are identifiable and objects like doors are detected.

Nevertheless in general the differences on the results are not very significant like the third scan shows.

Following parameters were used:

- **Common parameters**
  - Distance Threshold $d = 0.06\,cm$
  - Minimum Number of Points $t = 7$
- **Pure RANSAC**

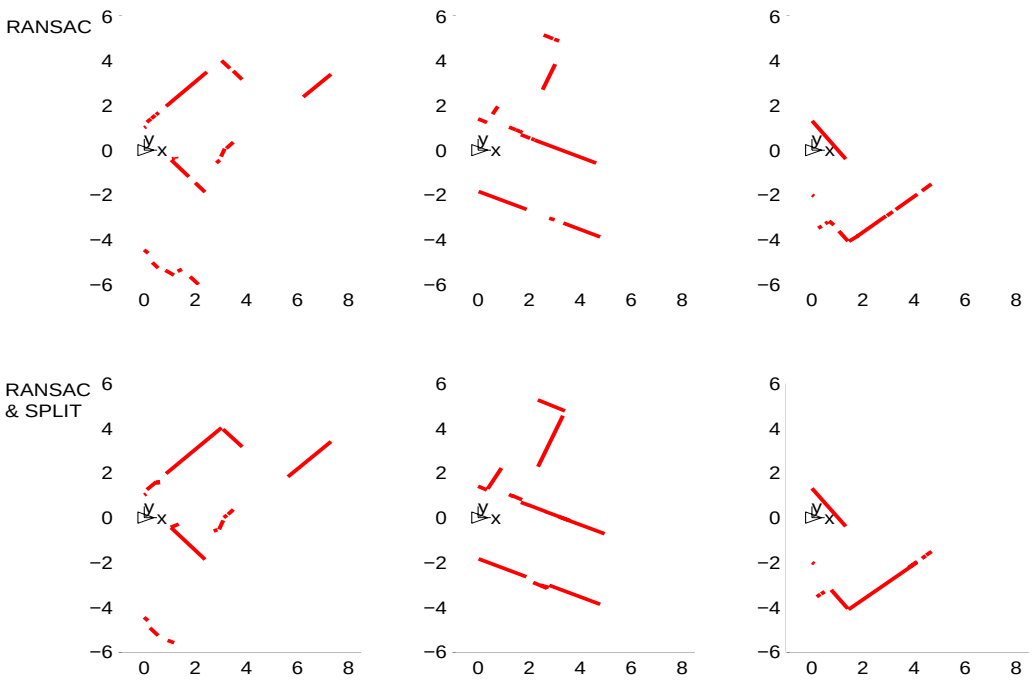

Figure 37: Comparison of the results of RANSAC without (above) or with split (below)

* $\omega = 0.2$
* $z = 0.8$
* $\Rightarrow k = 40$

- **Extended RANSAC with Split**
  * $\omega = 0.9$
  * $z = 0.9$
  * $\Rightarrow k = 2$
  * $split.d = 2\,m$

Figure 38 illustrates the average computational time measured of the different algorithm parts. The chart shows that the absolute difference in time is very small ($\approx$ 0.6 ms) though the relative difference accounts to 9.6%. As expected the procedure time for RANSAC decreases in the split case due to

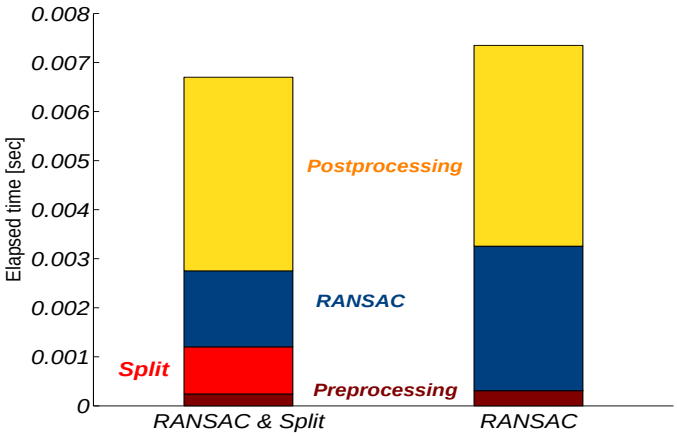

Figure 38: Calculational time for the different algorithm parts

the higher estimated probability parameters. The split step merely needs ≈ 1 ms what constitutes $\approx 15\%$ of the whole algorithm. The preprocessing plays again a very small role whereas the postprocessing the biggest part of both algorithm versions constitutes with each time more than the half of the time. This results on one side due to the normally large number of potential point groups which are generated through the randomly chosen points and on the other side on the required check on overlaps.

The measured absolute average and percentage values[20] are listed in Table 3.

The results show that the extension of the general RANSAC algorithm with a Splitting algorithm in advance generally improves the quality of the results and decreases the computational time.

7.4 Hough-Transformation

The mode of operation of the two different implemented versions of the Hough-Transformation already were described in chapter 5.4. In this chapter I want to describe the specific characteristics more detailed as well as to

[20] Values are rounded so a cumulated sum $\neq 1$ is possible

	RANSAC		SPLIT & RANSAC	
	time [sec]	$\% - value$	time [sec]	$\% - value$
Total mean time	0.0067	100.00%	0.007347	100.00%
Preprocessing	0.000243	3.63%	0.00031	4.22%
Split	0.00096	14.33%		
RANSAC	0.00155	23.13%	0.002947	40.11%
Postprocessing	0.003946	58.90%	0.004093	55.72%

Table 3: Average computational times and percentage values of different RANSAC parts

present the results.

7.4.1 Hough-Transformation by Revoting

The sequential parts of the *Hough by Revoting* (hereinafter also *HbR*) algorithm are illustrated in the flow diagram in Figure 39.
An Outlier filtering is not necessary due to the characteristic of a voting algorithm.

The iterative search for the most probable line in the current measurement set and the erasure of the respective points is illustrated sequentially in Figure 40.

The algorithm stops in the 8th iteration since the number of votes of the maximum is too low. Hence the number of potential point groups is 7 which are passed further to the postprocessing.

Each iteration the accumulator has to be recalculated so in general the number of votes v is specified by:

$$v = n + (n - n'_1) + (n - (n'_1 + n'_2)) + \cdots + (n - (n'_1 + \cdots + n'_m) \qquad (81)$$

where n specifies the number of the initial point sets, n'_i number of the recently found and erased point set P'_i and m is the number of valid sets.

In example of Fig.40 the subsets have the size $[105, 88, 37, 24, 21, 18, 8]$ the initial number of valid points is [310] and θ-step $= 2\,degrees$. So we have for this example a resulting number of votes v with

$$v = 90 * [310 + 205 + 117 + 80 + 56 + 35 + 17 + 9] = 90 * 829 = 74610$$

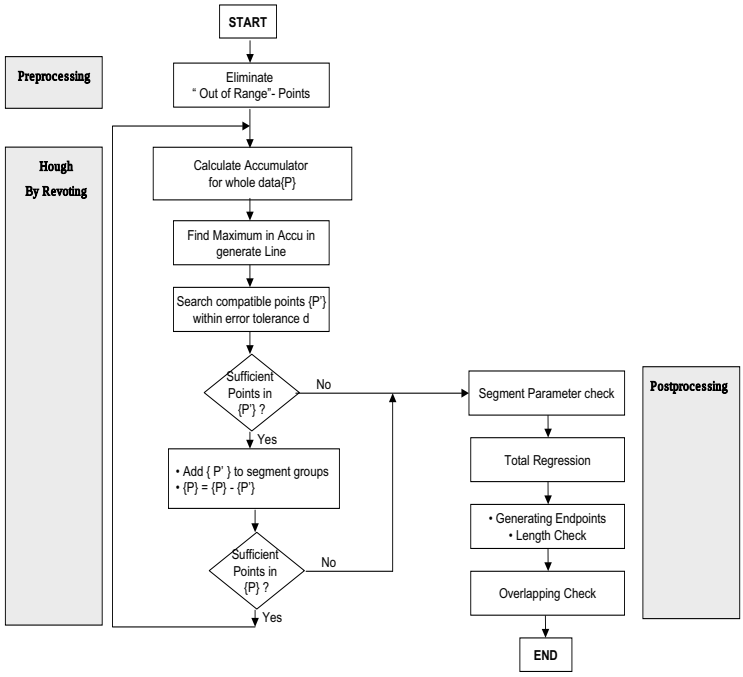

Figure 39: Flow diagram of Hough by Revoting Algorithm

As well we have to proceed a maximum search on the current accu 8 times.

The final result was obtained under constraint of the following parameters:

- ρ**-step:** $0.1\,m$
- θ**-step:** $2^\circ \equiv 0.0349\,rad$
- **Max. Distance:** $0.05\,m$
- **Min. Num. Points:** 5
- **Inter Segment Distance:** $0.05\frac{m}{m}$
- **Max. Invalids Gap:** 2

so the accu had the dimensions of a $[162 \times 90]$-matrix. The other parameters were chosen according to the previous considerations on the already described algorithms above.

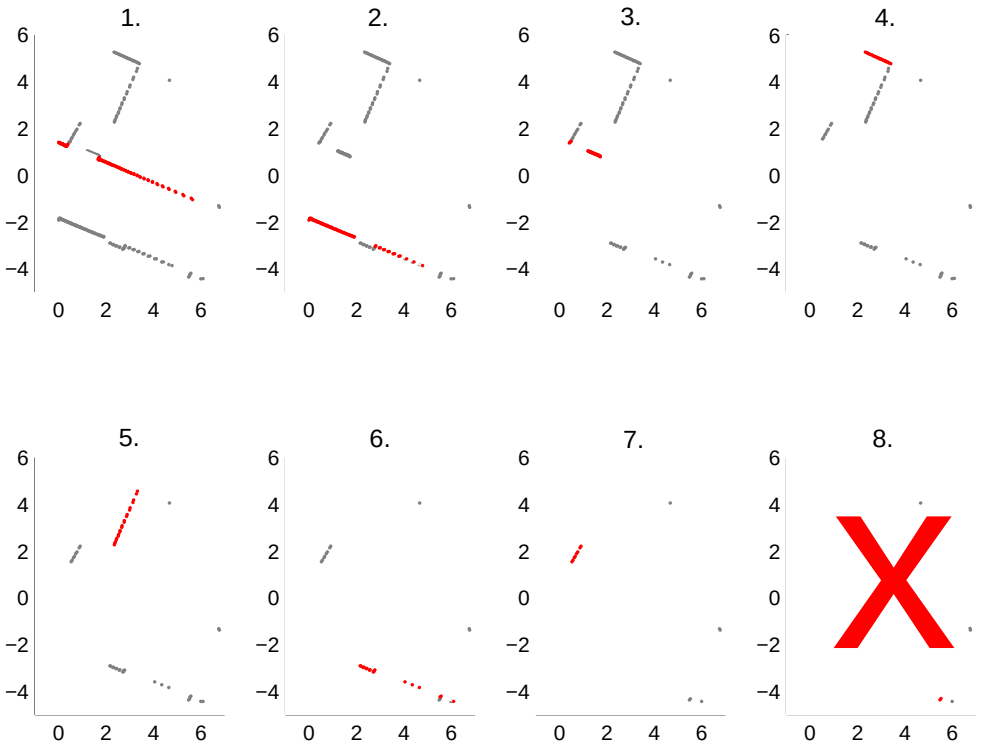

Figure 40: A HbR sequence of seeking and erasing points of one scan

The resulting segments are presented in Figure 41. The needed computational time was measured with 0.15 sec

Also in this case the application of the parameter Maximum Inter segment Distance can be demonstrated very well in the enlarged clipping shown in Figure 42. HbR estimated a line through the points on the outer left side of the scan since there exist definitely more points than the used minimum number of points threshold (5). Nevertheless between small groups of these points exist distances > 0.1 m. So the whole point group will be split into small ones since the measured points have a distance < 2m to the laser and the defined threshold accounted with $0.05\frac{m}{m}$. These split point groups have a size of $[2], [2], [2], [2], [4]$ thus each of them will be erased. Consequently a line which was estimated by HbR as the currently most probable one, due to the number of votes, was erased during the postprocessing process.

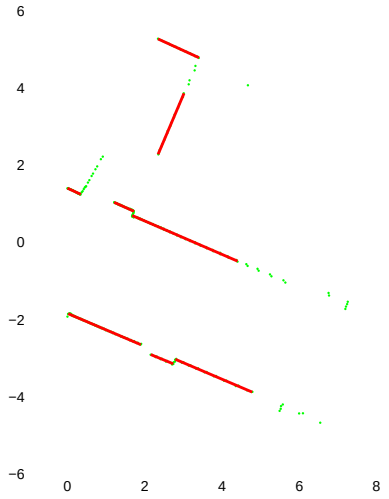

Figure 41: Result computed by Hough by Revoting

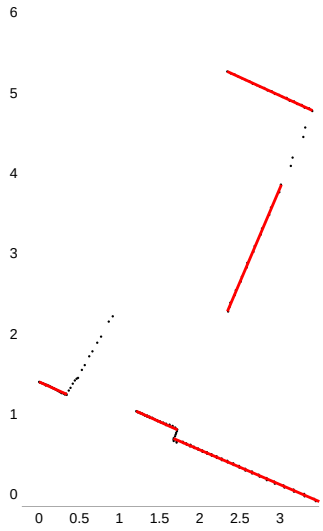

Figure 42: Application of the Inter-Segment Distance Parameter

A normally not desired characteristic of Hough by Revoting is the procedure on a reduced data set after point erasure as already explained in connection with RANSAC. A real example where it caused unintended results is presented in Figure 43.

The immense calculational effort and the loss of information due to proceeding on reduced data led us to another version of Hough.

7.4.2 Hough-Transformation by Neighbourship-Relation

By the implementation of *Hough by Neighbourship* (hereinafter also *HbN*) we tried to avoid the repeated calculations of the accu and the erasure of data from the set.

The complete Algorithm is illustrated in Figure 44.

The accumulator only has to be calculated once. From this accumulator we want to extract the local maxima by means of a clustering algorithm which considers the neighbourship relations between the accumulator values, re-

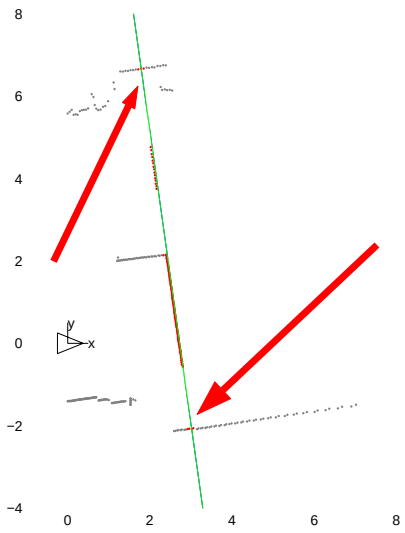

Figure 43: Separation caused due to erasure of points by HbR

spectively the Hough-Space coordinates. There are two different kinds of neighbourship: direct and indirect. The direct neighbours of a value are situated in the accu directly above, below, left or right. So one value has got 4 direct neighbours and the neighbours are defined by a deviation from the original value *either one θ-step OR one ρ-step* whereas the indirect neighbours are situated in the "diagonal" neighbourship of the original value. If you are using as well indirect neighbourship this signifies that a neighbour can deviate *one θ-step AND/OR one ρ-step*. In our implementation the usage of indirect neighbours was parameterized and thus can be turned off.

The problem with this algorithm is to find the limits of a cluster because the Hough Transformation in general doesn't provide separable clusters but a "value-cloud" where more or less all values are recursively adjacent (Fig.18 page 52). So we have to generate recognizable gaps between the clusters. First we set ρ to a lower value e.g. $0.05m$ to "stretch" the accu maximums further away and we eliminate in advance all values which fall below a particular threshold e.g. 10. Thus all lines which only are defined by less than 10 points will be ignored. This makes our results worse, but is necessary to find separate clusters at all.

Figure 45 shows the previous used accu filtered under constraint of a threshold of 7, 10 and 13 votes. The clusters in the second and third case

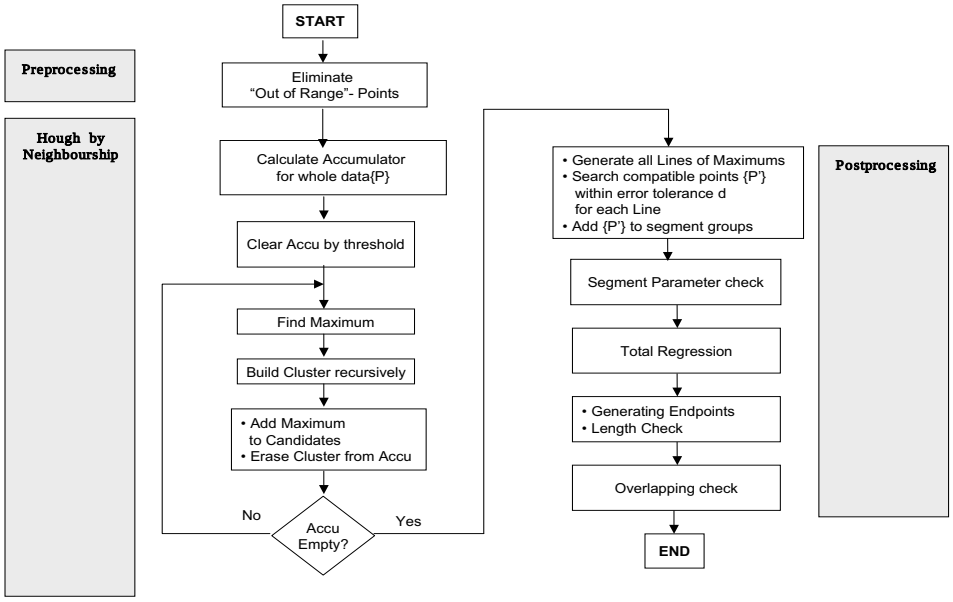

Figure 44: Flow Diagram of the *Hough by Neihgbourhsip* Algorithm

are easy to recognize.

This version of a Hough applying algorithm has the characteristic to provide much more candidate-lines than in the real world model really exist. Due to the error tolerance of the laser different points of one real world segment could vote for different parameters so if this parameters aren't adjacent they could cause different clusters which can be very close though not directly adjacent. The previously introduced example (Fig.17 & 18, page 52) with a chosen threshold of 10 votes produces 34 candidates. So on all these candidates the complete postprocessing has to be applied.

Figure 46 shows on the left the obtained candidate-lines and the result of the whole algorithm after the postprocessing.

Comparison of the two Hough versions

For an adequate comparison of the different versions it was essential to implement them under C/C++ due to the immense computational effort and

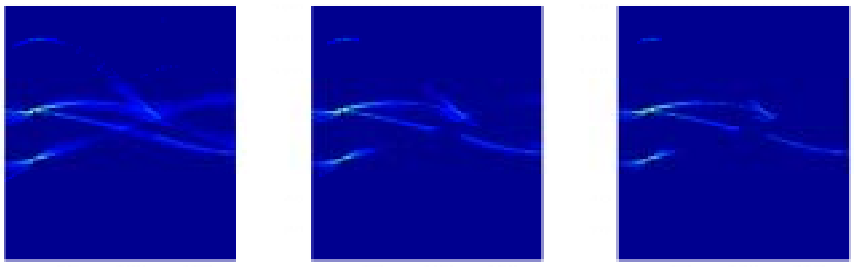

Figure 45: Filtered accus by a threshold of 7, 10 and 13 votes

the used recursion. MATLAB provided results which couldn't be considered to be close to reality if we want to bear in mind the later use on a real time robot.

The advantages and disadvantages of the two versions are obvious. The Revoting algorithm produces very good results considering the currently most probable line but this has to be "payed" with an immense computational effort whereas the Neighbourship algorithm reduces the computations but extracts an "unordered" set of potential lines from a pre-filtered data set that obviously contains less information than the unfiltered original measurement.

The segmentation results were obtained on equal scan data whereby the parameters were considered reasonably regarding the tasks and the advantages of the algorithms and were chosen as follows:

- **Hough by Revoting**
  - number-threshold $= 7$
  - ρ-step $= 0.1\,m$
  - θ-step $= 2^\circ$
  - line error tolerance $= 0.05\,m$
- **Hough by Neighbourship**
  - number-threshold $= 10$
  - ρ-step $= 0.05\,m$

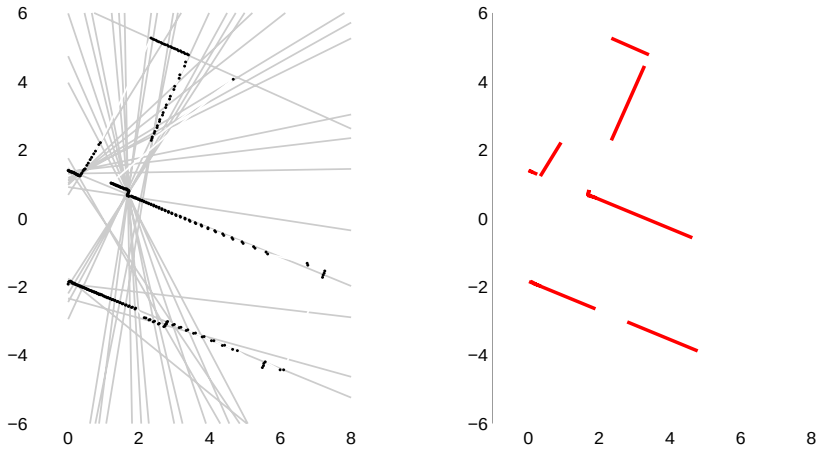

Figure 46: All candidate-lines of one HbN pass and the resulting segments

- θ-step $= 1°$
- line error tolerance $= 0.025\,m$

The results are given in Figure 47. The scales were partly changed to get better display results though the axes of two same scans are equal.

The figure shows above the obtained segments, provided by the *Hough by Neighbourship algorithm ($algo_1$)* and below provided by the *Hough by Revoting algorithm ($algo_2$)*. In general the results of $algo_2$ were at least the same quality as $algo_1$ or even better. The second and third scans show the disadvantage of $algo_1$. Segments with almost equal parameters are ignored because they are too close to be distinguished in clusters as well as ignored segments due to the recursive building of the clusters. Scan no.1 shows more or less equal results. There the advantage of $algo_1$ can be seen since the points aren't erased from the data set and so one point can be used for two segments e.g. in building a corner.

Figure 48 illustrates the chart of the time measurements average values and Table 4 contains the absolute values and the relative percentage values.

As expected, the Revoting version ($algo_2$) needs more time with an average computational time of $\approx$ 0.186 sec compared to HbN ($algo_1$) with $\approx$ 0.153 sec. In effect $algo_2$ proceeds $\approx$ 22% longer than $algo_1$. The postpro-

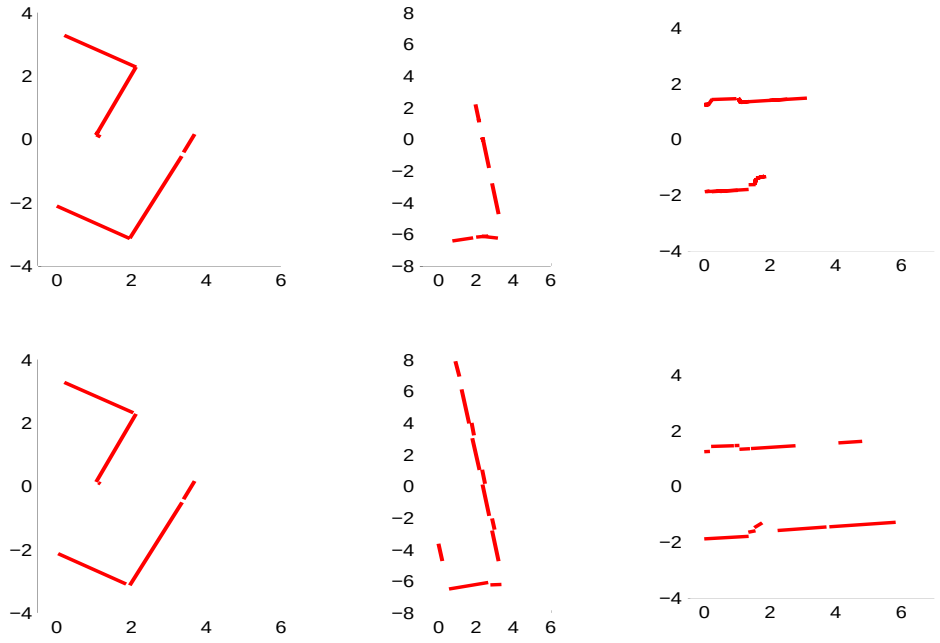

Figure 47: Different results on same data obtained by different versions of Hough

cessing of $algo_1$ is slightly longer compared to $algo_2$ due to the large number of candidate lines which have to be filtered whilst the preprocessing in both cases can be disregarded with $\approx 0.2\%$ of the entire algorithms. The recursive clustering algorithms consumes only $\approx 25\%$. The accumulator gets calculated *only once* in $algo_1$ whereas in $algo_2$ the accumulator is filled on average ≈ 9.6 times. Despite this fact the cumulated time of accu-calculation of $algo_1$ only needs $\approx 64\%$ more compared to the one-time calculation of $algo_2$. This is caused on one side by the larger dimensioned accu of $algo_2$ as well as the rapidly decreasing number of points which have to be transformed into the Hough space after erasing the first potential line points.

Parameter Variations

Improving the quality of the segments of the Neighbourship version by means of the used clustering algorithm arises as problematic. Trying to produce more convenient clusters signifies to enlarge the dimensions of the accumulator what in turn entails larger computational costs. By raising the threshold

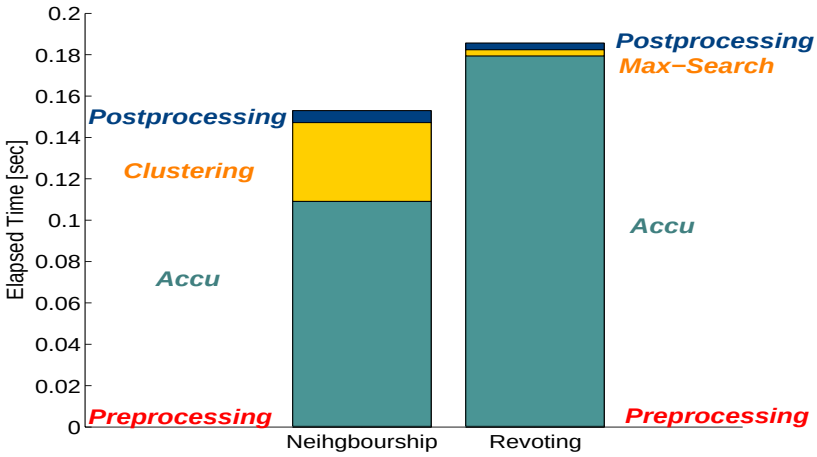

Figure 48: Time-comparison of algorithm parts of HbR and HbN

the loss of data, i.e. ignoring the small segments, gets immense and thus results counterproductive.

The above used parameters produced a reasonable compromise between quality and computational time.

The big disadvantage of the Hough by Revoting version consists in the used computational time which primarily depends on the accu dimensions. Hence the improvement approach leads to reduction of the accu. To influence directly the calculation dimensions we varied the parameter θ-step. The measured average times are given in Figure 49. It's easy to see that the used time decreases drastically. To compare the results a sequence of the same measurement data with a different θ-step can be seen in Figure 50 and Figure 51. The values for the sequential parts were chosen from left to right with: $\Delta\theta = [1°, 9°, 15°, 30°] \equiv [0.0175, 0.1571, 0.2618, 0.5236][radience]$.

Obviously the first segments on the left always are optimal due to the chosen θ-value. For this parameter an average time of ≈ 0.29 seconds was obtained. The first sequence (Fig. 50) shows an example where the results in all cases almost are equal since the angles of the major segments are exactly in the direction ("corridor") of the used angular parameters. This quality of results was obtained in few cases. The second sequence (Fig. 51) shows a standard case. The first results on the left are optimal whereas the last result is worse. With a increasing of $\Delta\theta$ the segments which aren't situated

	Hough by Neighbourship		Hough By Revoting	
	time [sec]	$\% - value$	time [sec]	$\% - value$
Total mean time	0.152969	100.00%	0.185596	100.00%
Preprocessing	0.000331	0.22%	0.00034	0.18%
Accu	0.108820	71.14%	0.179117	96.51%
Clustering	0.038029	24.86%		
Maxima-Search			0.00298	1.61%
Postprocessing	0.005786	3.78%	0.003158	1.70%

Table 4: Average computational times and percentage values of different Hough versions

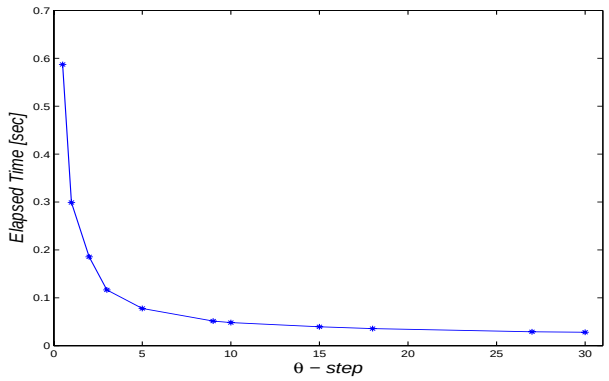

Figure 49: Average time with varying θ-step by Revoting Algorithmus

exactly in a respective angle direction will be partitioned or not even found at all. But this characteristics only were observed in this quantity and with such effect on a $\Delta\theta > 15°$. The results in the second and third example are still rather acceptable. Regarding the immense saving of time slightly worse results are absolutely acceptable.

A used $\Delta\theta$ of 9° consumed an average time of ≈ 0.0535 seconds and produced very good results though in few cases separations occurred.

Compared to the Hough by Neighbourship version the results of Hough by Revoting are equal or better but needs due to the above explained optimization less time for the procedure.

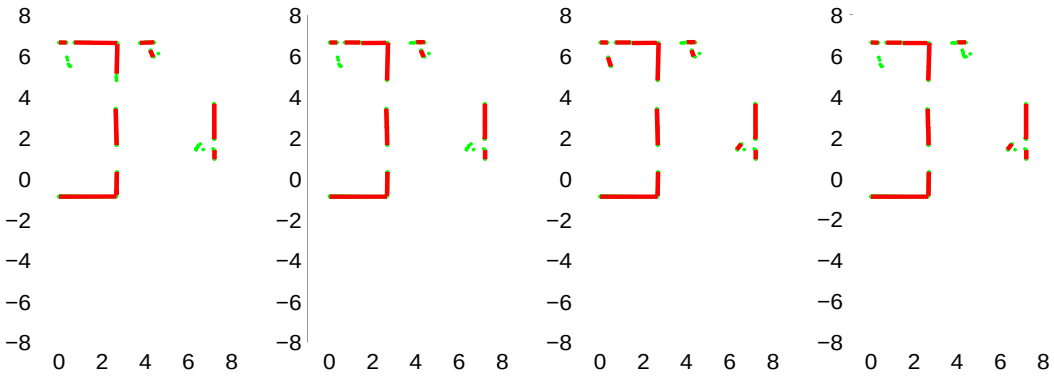

Figure 50: Results of equal data with varying θ-steps

7.5 EM-Algorithm

The complete *Expectation-Maximization*-Algorithm is illustrated in Figure 52.

EM proceeds an outlier filtering since outliers would influence the expectation values of the valid measurements in a considerable way.

Beside the normal used parameters which in this case were chosen in a reasonable way we have to specify several EM specific parameters.

Specifying σ
The parameter σ which is used in the expectation calculation specifies the "closeness" of the points to a line. It denotes the variance parameter of the distance function and so specifies the weighting a point gets with a certain distance to the line. σ is used as denominator in the exponent of the probability distribution function whilst the numerator has a constant value for one measurement z_i and one single model Θ_j. The smaller σ the bigger the value of the fraction hence the smaller the probability (due to negated exponent) of this point to pertain to this line. σ normally can be calculated by a transformation of the covariances of the line and the point. With a systematically initialized model we have no information about the uncertainty of the line so σ only would depend on the constant covariance of the measurement. In case of a line defined by two randomly chosen points a covariance for the line

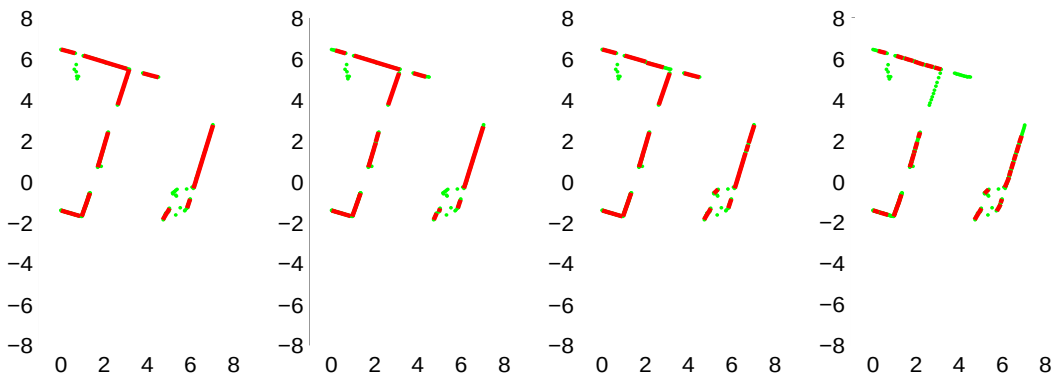

Figure 51: Results of equal data with varying θ-steps

can be calculated by a transformation of the two point covariances. This in turn can be used to calculate σ. This procedure would have to be calculated for each point to each model line (Complexity $O(n \times m)$) in each iteration. In general the covariances only have small differences due to the constant covariances of the points so a constant variance σ can be considered to be absolutely sufficient in this case.

A small σ (e.g. 0.05m) leads to a fast convergence since already small differences of distances of a measurement to different models leads to clearly weighted votes from the point to the lines with a significant preference to the closest. So the optimization produces relatively fast clearly defined models whilst a big value for σ would lead to slightly different expectation values for one point to all the model lines so the optimization will proceed small actualizations of the current model. On the other side a small σ leads quickly to a voting for the phantom model since the expectation for a far away line compared to σ will decrease quickly. Thus single model lines will be erased from model Θ rather quickly if no points are "close" to them.

Figure 53 shows a virtual test scenario. There exist two point groups with each 5 points. Each of the point groups are situated on a "virtual" line parallel to the x-axis and the groups have a mutual distance of 1 m. The initial model line (blue) is situated slightly closer (1 cm) to the below point group. On this situation EM is applied once with $\sigma = 0.25$ [1] and once $\sigma = 0.02$ [2]. In case [1] EM needs 7 iterations to converge in case [2] only 3. It's good to see how in case [1] the optimization are "advancing" slower

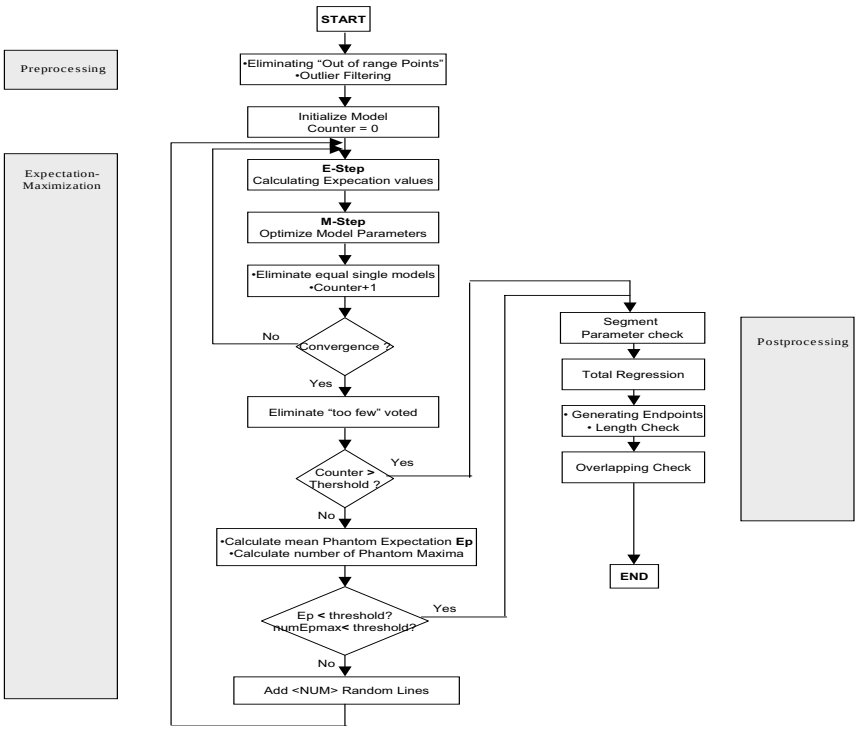

Figure 52: Flow Diagram of EM-Algorithm

than in case [2].

The limits for convenient σ-values were collected experimentally and were determined approximately with $0.018 < \sigma < 0.4 \quad (\equiv \frac{1}{\sqrt{2\pi}} < \sigma < \frac{1}{\sqrt{10\pi}})$. In case of $\sigma > 0.4$ in the above specified test scenario the model converges to the line exactly in between the two point groups. In case of $\sigma < 0.018$ the first optimization isn't parallel anymore and the model converges vertically to only two points of the groups.

After the above obtained results in tests and further examinations we used a σ between 0.02 and 0.1.

Model initialization
The initial model has essential influence regarding the quality of the results

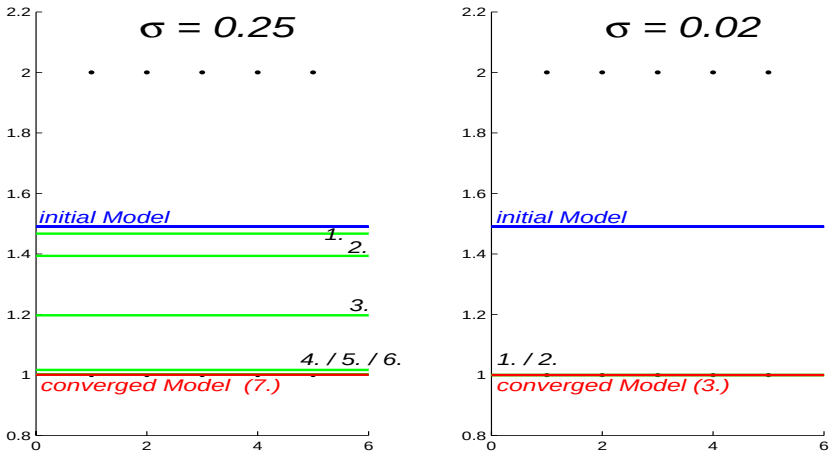

Figure 53: EM with different σ's on a virtual test scenario

of EM. The worse a model is initialized the worse are the results or the longer EM needs to obtain satisfactory results. Thus a flexible but complete initialization should be applied.

As described in the introduction of EM (chap. 5.5) the initialization of the model was implemented in two different modes: on one side with an initial systematic, pre-defined model and on the other side with a certain number of randomly generated lines.

Systematic Model Initialization
A systematic model consists of regularly distributed lines scattered systematically over the whole scan range. The primary intention was to achieve a initial distribution of lines to "reach" as many points as possible in the beginning without concrete a priori knowledge of the measurement distribution. The redundant lines which aren't relevant regarding the current measurements are optimized to the zero-vector and so are detectable and can be erased immediately i.e. each iteration. A check of the entire model on zero-vectors is realized rather fast regarding the computational effort (complexity $O(n)$ with n = numbers of current models), thus irrelevant models affect the computational costs insignificantly due to their early detection and erasure.

In our case we used three different kinds of systematical models with

different numbers of lines[21]. The lines were specified regularly (varying line parameter ρ) taking into account the "direction range" of the laser with four different values for θ with $\theta \in [+\frac{\pi}{2}, +\frac{\pi}{4}, 0, -\frac{\pi}{4}]$. A big initial model consists of 17 lines, a medium sized of 13 and a small one with 7.

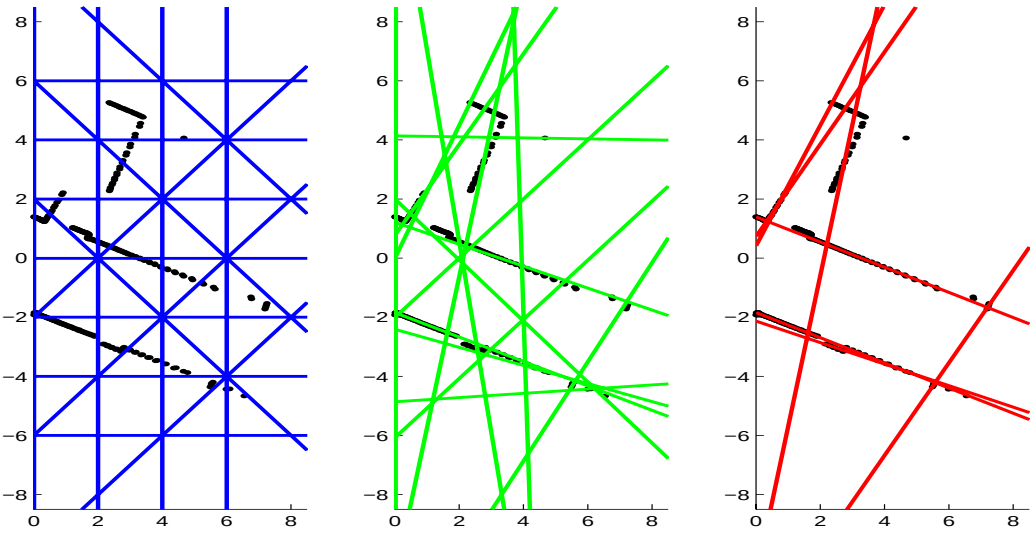

Figure 54: EM applying a big systematic initial model

Figure 54 and 55 show the sequence of EM applied on the same scan once with a big initial systematic model and once with the small one. The first images on the left shows the measurements (black) and the initial model (blue), the one in the middle shows the model after the first EM-Iteration (one optimization) and the left one shows the model after the first detected convergence. The used σ-value was 0.02.

The initial vertical and horizontal model lines have a distance of 2 meters and the diagonal each ≈ 2.8 meters ($\sqrt{(8)}$). The whole scan range can be considered to be covered quite completely. Obviously the probability of a model lying close to each potential point group is rather high and regarding the results they can be considered relatively good since after first convergence

[21] These were test models and were found to be reasonably and satisfactory. Obviously the model specification is absolutely arbitrary and can be chosen regarding prior environment knowledge and task requirements.

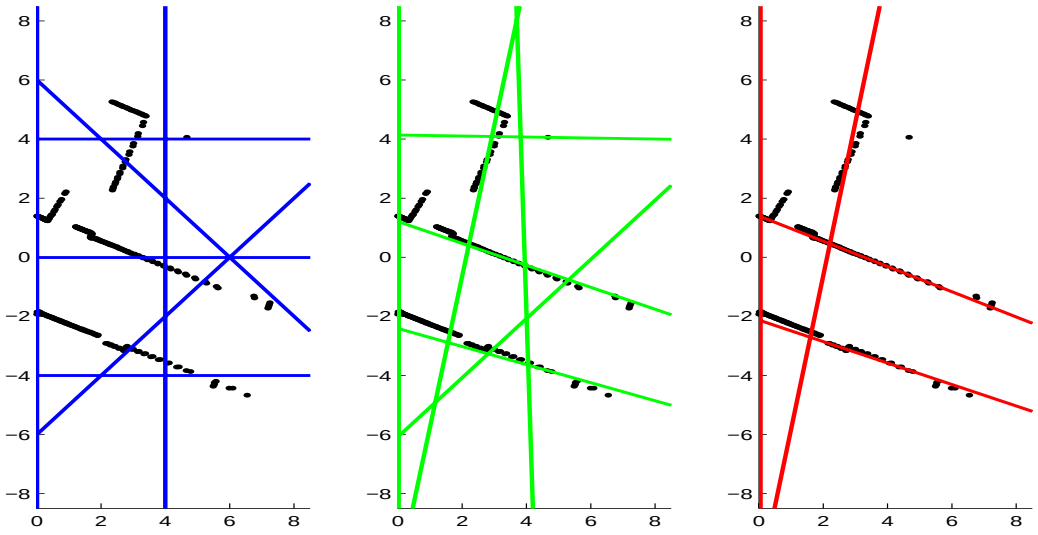

Figure 55: EM applying a small systematic initial model

almost each of the potential segments are covered, except the segment on the top.

In the small model the horizontal and vertical lines have a distance of 4 meters and in diagonal direction there exists only one single model. Evidentally the smaller number of initial lines reduce the possibility of covering as much as by using the big model so after the first iteration and first convergence it can be considered as well quite good though not as complete as in the previous case.

Table 5 presents the results on number of lines, iterations etc. of the above given cases as well as the additional case for the medium model. EM only was proceeded until the first convergence thus no lines were added neither a break condition was reached.

In the case [1] two single lines were erased immediately due to the 0-vector check whilst in the other cases no 0-vector was detected. The results regarding the used models are as expected since the bigger models having finally more resulting lines than the small one. Though here the arbitrary specification of the models leads to a slightly more resulting lines for case [2] than in case [1]. This shows that an ideal initial model doesn't exist without knowledge of the measurement distribution hence there is not a

	Big [1]	Medium [2]	Small [2]
Initial Lines	17	13	7
Lines after 1st iteration	15	13	7
Lines after 1st convergence	7	8	4
Detected 0-vectors	2	0	0
Iterations until convergence	8	10	6
Used time until convergence	≈ 0.34 sec	≈ 0.34 sec	≈ 0.14 sec

Table 5: Results on varying initial Models

"best" initialization though in general a larger model can be considered with a bigger probability to cover a major part of the points.

For the necessary number of iterations this neither is the case since the medium sized model needs more iterations until convergence than the large model though the trend to less iterations with a small model is easy to confirm.

Regarding the computational time the results are as expected. In case [1] the early eliminating of unnecessary model lines, the smaller number of resulting lines and the smaller number iterations compensate the primarily larger number of models compared to case [2]. The small initial model and few iterations in case [3] lead to the smallest elapsed time.

If EM would end with the first convergence the following results (Fig 56) would be obtained after the complete segmentation procedure.

The results are more or less equal and anything but satisfactory. So the procedure of EM proceeds until convergence iteratively and adds each convergence-iteration a specified number of randomly, by the points, generated lines. Due to the fact that the model after the first convergence already is associated to a major part of the entire measurement set an added line which isn't attached to previous minor assigned points will be erased very quickly due to the absence of heavy weighted votes. Therefore the number of lines to add normally was set larger than one. On the other side too much added lines would cause an enlargement of iterations until the next convergence.

Tests with an iteratively differing systematical model to add were applied as well, though didn't produce satisfactory results since in most of the cases a major part of the added model has been erased immediately.

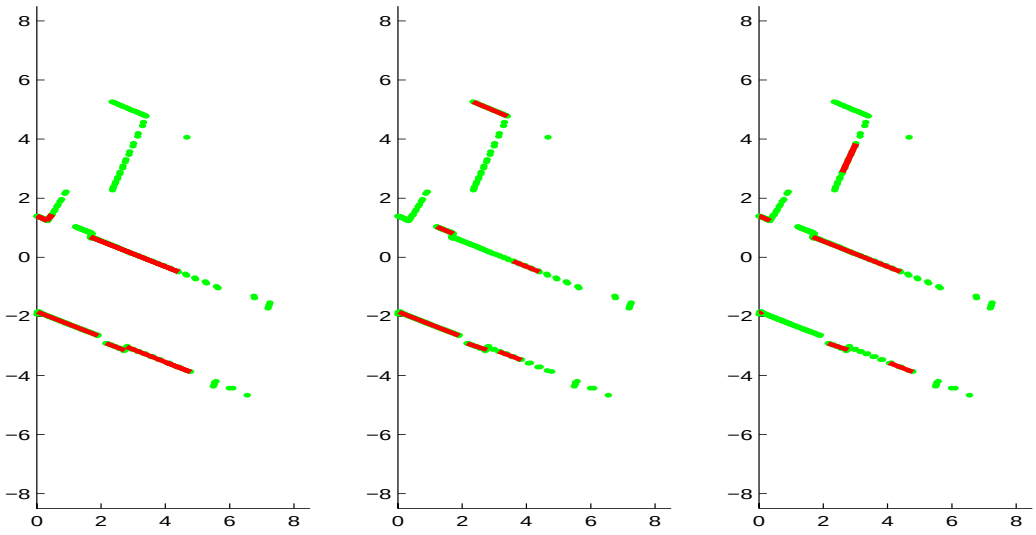

Figure 56: Potential results after 1st convergence with varying initial models

The initial model already defines the measurement model in a way that a added systematic model only would "overdefine" the current points. Only a rather different model compared to the initial model would lead to significant changes whereas such a model can be considered not very different to randomly generated lines so the random way can be considered as more flexible and normally leads faster to significant results.

Good results were obtained with a fixed value of randomly generated lines between 3 and 8.

Random Model Initialization

Another way to initialize the starting model is to generate a specific number of randomly chosen lines using existing measurements to specify them. In this case a single model always will be voted of at least two points (respectively the specification points) hence zero vectors won't appear.

The problem with this procedure is obvious: randomly chosen lines tend to equality in case of very probable potential point groups as well as to pile up to certain measurements (e.g. to data close to the laser since the density is much higher for data with small laser distances). So the point of view we had with a systematic model, to assign initially as much as data as possible, only can be achieved with the random version by creating a big number of

lines. This leads to many likely lines and to a preference in location.

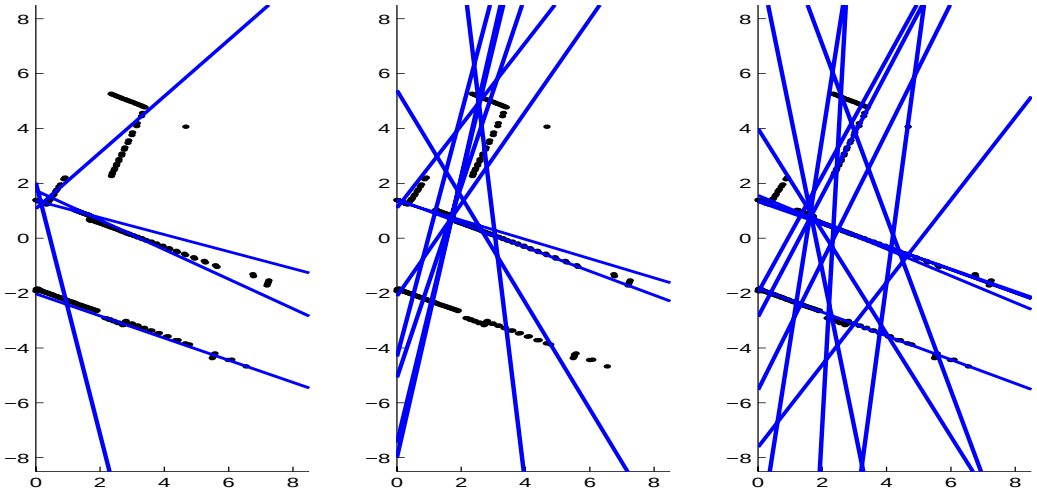

Figure 57: Generated random line models with varying numbers

Figure 57 shows three randomly generated models with a number of [5 , 10 , 15] lines. The more lines are generated the more lines seem likely as well as line "bunches" are growing bigger and bigger. It's obvious that quickly a lot of lines will be erased due to equality. The results led to the approach of integrating the initial model into the process of converging iteratively and adding randomly generated lines, i.e. the number of initial lines coincides with the number of lines to add on each detected convergence (between 3 and 8).

This approach can be considered as a "slow" approximation to the final result with an iterative improvement of the current model whereas the initialization with a systematic model tries to begin with a model as good and complete as possible and afterward to optimize it by adding few lines iteratively.

Maximum number of Iterations

The major break condition for EM is the *Maximum Number of Iterations*. EM only stops in case of convergence thus only if the number of iterations *after* convergence is higher than the specified threshold EM will stop. Occasionally it occurs that EM iterates a lot of times more than the defined

threshold though in general this was not the case due to the fast convergence characteristics of EM[22].

The specification of the maximum number of tries was achieved experimentally due to the fact that an analytical way to specify an appropriate value is very hard to achieve. Figure 58 shows a sequence of EM on equal scan data with different maximum number of iterations. From left to right the values were $z = [10, 20, 30, 50]$. A medium sized initial model was used and the number of additional random lines per convergence was specified with 3.

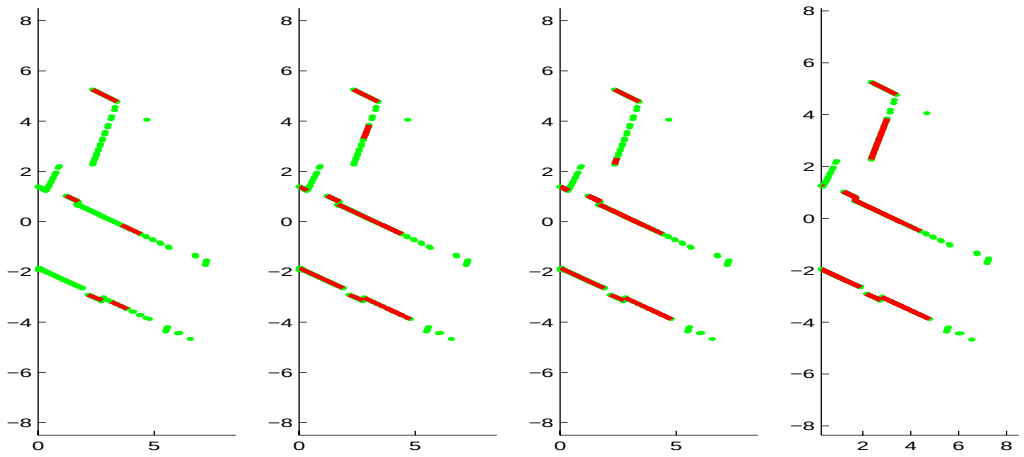

Figure 58: Results on equal scan data with varying *Maximum Iterations*

The trend of better results with more iterations should be obvious. In the test series a maximum number of iterations > 50 didn't improve the results significantly. The results with the value 50 in most cases are very good and only occasionally certain "important" segments aren't detected. A value < 20 produced only unsatisfactory results so the normally used value was defined between 20 and 50. A good compromise between computational time and results was found with 25.

[22]Taking into account the respective parameters σ and *Convergence threshold*

Defining Convergence Threshold
The threshold of convergence specifies the difference of the line parameters of two lines which they may have to be considered as equal. The problem with this definition lies in the fact that small differences of a short line vector can lead to completely different line direction whereby farther away situated lines aren't affected significantly by small differences.

This threshold may affect the number of iterations EM is proceeding significantly. In case of a too small chosen value lines which are changed (optimized) slightly but still aren't considered to converge won't improve the results significantly but only will enlarge the computational time. In the major part of the cases the lines aren't situated very close to the laser so we can more or less ignore this situation. Taking into account the laser error tolerance and examinations of further tests we specified the convergence threshold between 0.005 and 0.025 meter.

Defining max. $\bar{E}[\Theta_*]$-mean and number of $E[\Theta_*]$-max
Another break condition for EM was the quality of the result. We consider a current model as good if the mean value of the expectation values for the phantom model falls below a certain threshold or if the number of the maxima in phantom expectations are falling below a threshold.

These thresholds only should be considered to be used if we really can be certain that the defined thresholds are specifying a "quality"-model. The threshold for the phantom expectation mean value $\bar{E}[\Theta_*]$ was determined experimentally by estimating the quality of the results and comparison of the given values. In some cases yet a value of $\bar{E}[\Theta_*] = 0.2$ was extracted of a good model whereas in further cases only a value of 0.05 specified a good model. So this value normally was specified with a value between 0.05 and 0.02.

Points which are situated too far from a model will vote primarily for the phantom model. Thus a possibility to find the number of points which are not voting for a existing model is to find for each point the most probable (closest) single model. If the most probable line is the phantom model this particular point can be considered not to be defined by the current model. Obviously the number of such points should decrease through the optimization procedure. The threshold was defined with the required maximum number of points we want to be unassigned. For example if we want to have assigned at least 99% of a data set of 330 points the value would have to be chosen with 4. In our

case we specified the maximum number of such "Phantom Maxima" with a value between 2 and 5.

Figure 59 shows the plotted series of the mean $\bar{E}[\Theta_*]$ and the number of maxima $E[\Theta_*]$-max over 200 Iterations.

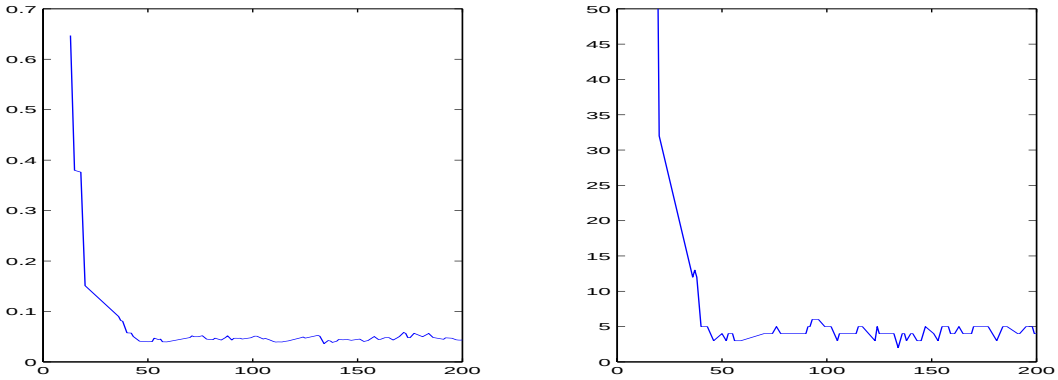

Figure 59: $\bar{E}[\Theta_*]$ and number of maxima of $E[\Theta_*]$ over 200 Iterations

In both cases the value converges. The mean value converges around a value ≈ 0.045 and the value for the phantom maxima around a value of 4. The results of the segmentation haven't changed significantly after 50 iterations. The final result is given in Figure 60.

Time measurement

The time measurements were made on one side with the randomly initialized model and on the other side with systematical models and were captured under C/C++.

Following Parameters were used:

- Maximum Iterations: 20
- σ: 0.02
- Convergence Threshold: 0.02 m
- Number of Lines to add per Convergence: 3
- $\bar{E}[\Theta_*]$: 0.03
- maximum number of $E[\Theta_*] - max$: 3

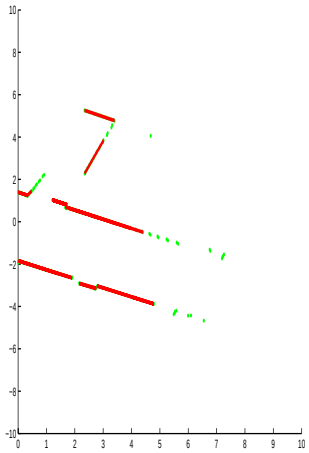

Figure 60: Results of EM after 200 Iterations

	Elapsed Time [sec]
Randomly initialized	1.76
Init with medium sys. model	1.829
Init with small sys. model	1.529

Table 6: Results on different EM variations

Table 6 shows the average measured times for the different versions of EM. Figure 61 and 62 shows some standard results for EM. On the left the final segments for the randomly initialized version of EM [1] with a number of 3 initial lines. In the middle a systematic initialization with a medium sized model (13 lines) [2] and on the right with a small systematical model (7 lines) [3].

In the first figure the results are acceptable but not very good since some major segments are missing in each case. The best results provides case [2] and as expected the results are better than in case [3]. Case [1] neither finds all major segments e.g. the segments of the corner on top.

The second figure shows more ore less equal results for all versions. All segments which can be considered to be important to be detected are detected except the door on the bottom in case [3].

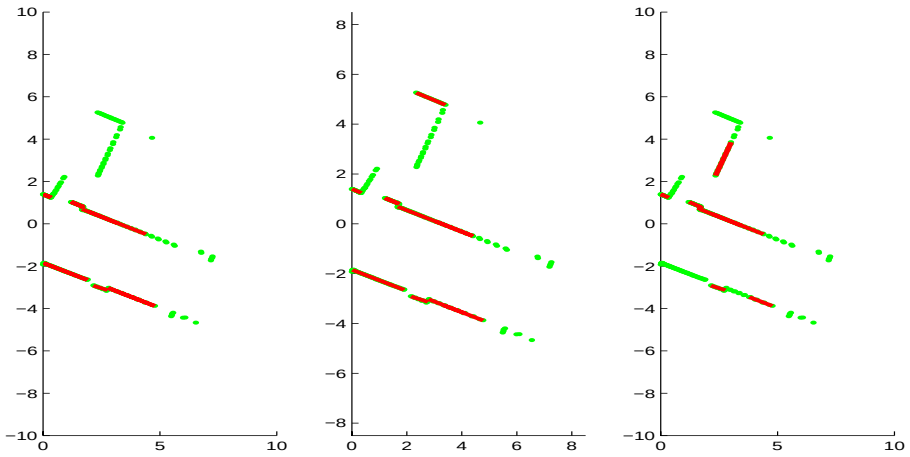

Figure 61: Final results of different EM versions after 20 Iterations [1]

In general the results are not as equal and are more similar to the results given in Fig. 61. The systematically initialized and afterwards randomly altered versions provide in general better results than the randomly initialized versions. The trend shows a preference for a big initial model and not more than 25 Iterations.

The big problem obviously is the computational time which is necessary for EM. An average time of ≈ 1.6 seconds ! can be considered as unbearable compared to the previously presented algorithms since the results neither are better. One advantage of EM consists in the possibility to find a very good result if it's only possible to give it sufficient iterations though thereby the computational cost would increase immensely.

In this chapter we specified the parameter and estimated reasonable compromises between quality of the results and calculation time for each algorithm. We tried to point out the limits of the algorithms and tried to find the best mode of operation regarding the required task to extract the potential segments from a set of measurement data captured in an indoor environment.

Now we have the results to compare the algorithms mutually.

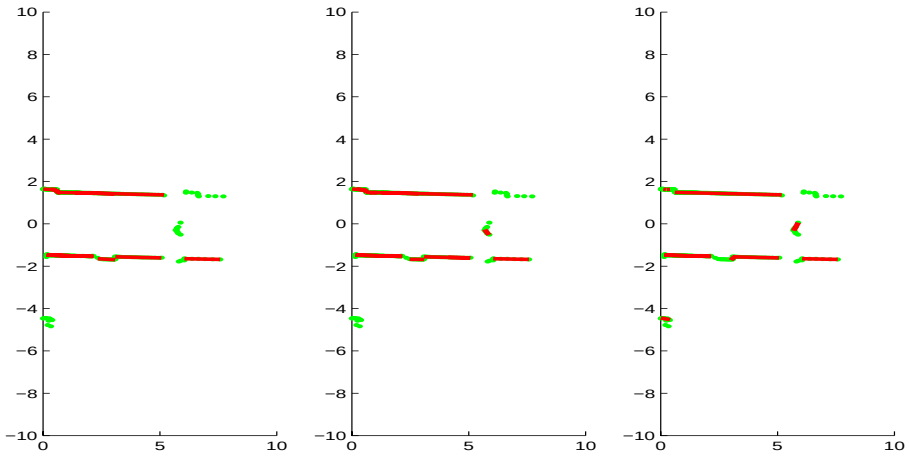

Figure 62: Final results of different EM versions after 20 Iterations [2]

8 Algorithm Comparison

In this chapter we want to use the results of the previous chapter and apply a direct comparison of the algorithms.

8.1 Direct Comparison

For each algorithm the following parameters were used:

- **Common Parameters**
  - Minimum Number of Points: 5
  - Inter Segment Distance: 0.05 $\frac{m}{m}$
  - Max. Number of "Invalids": 2
  - Minimum Segment Length: 0.1 m
- **Split & Merge**
  - Max. Point-Line Distance: 0.08 m
  - Merging by MNE
- **RANSAC (with Split)**
  - Max. Point Line Distance: 0.06 m
  - ω: 0.8
  - z: 0.9
  - $\rightarrow$ k = 3
  - Max. Split Point Line Distance: 2 m
- **Hough (by Revoting)**
  - θ-step: $6^\circ \quad (\equiv 0.1571 rad)$
  - ρ-step: 0.1 m
  - Max. Point Line Distance: 0.05 m
- **EM**
  - Initialization with medium sized systematical Model
  - Max. Iterations: 20

* Convergence Threshold: 0.01 m
* Number of Phantom Expectation Maxima: 3
* Maximum Phantom Expectation Mean: 0.04

Result Comparison
In the following Figures the results for each algorithm with the above given parameters will be presented.

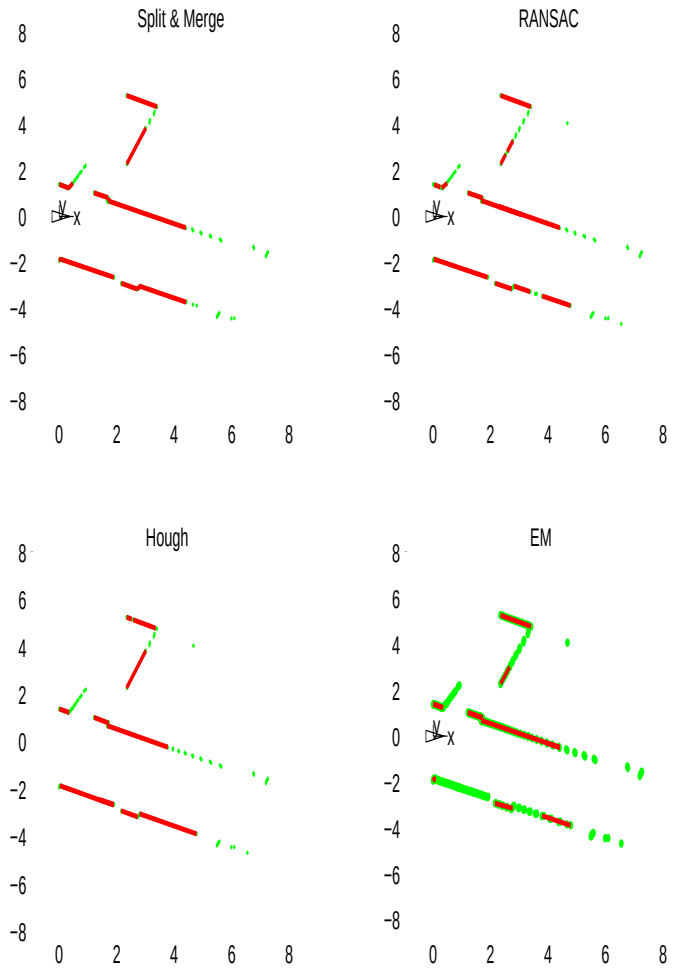

Figure 63: Results of all algorithms on equal scan data [1]

The first series (Fig. 63) shows the results on the example which was used previously.

This typical scan was captured in a university corridor. The Laser was situated angularly to the corridor direction and had insight into a laboratory (on top) through a double door where only the left door part is opened.

The results of Split & Merge (hereinafter also denoted as SM) and Hough were rather good since all major walls and segments, which comply the segment requirements, are detected as complete segments. The closed door on the bottom and the second part of the double door, both with a depth around 0.1 meter, also are detected separately.

RANSAC shows some separations e.g. of the wall segment on the bottom with a gap of $\approx$ 30 cm or a segment in the laboratory with a gap of $\approx$ 10 cm. But in general all the major segments are detected as well.

The results of EM are not very satisfactory. The important segment of the bottom wall are only partially or not even at all detected. Nevertheless the two closed doors are identified as such.

The next series (Fig. 64) shows a case which was merely untypical. This scan was captured whilst the laser was situated in a laboratory door and had direct sight into the corridor with an opposite wall with several closed doors.

In this case the results are more or less the same for all algorithms though regarding some characteristics RANSAC and EM showed even better results than Split & Merge and Hough. The door on the most top was only detected by RANSAC and EM. In reality the segment on the bottom is a window so the laser beam reflection characteristics weren't optimal. Due to this Hough doesn't detect it at all equally to EM. RANSAC separates it whereas SM generates it as a whole. One large wall segment directly opposite to the laser was separated only by Hough.

The opened door on the bottom leads to a complete different segmentation. SM and RANSAC can't detect the part of the opened door neither the corner between the door and the window. Hough detects the partially visible door and EM only the small wall segment between the door and the window.

The small part of the inner laboratory door was specified by $\approx$ 50 captured points with a distance to the laser of approx 45 cm and was extracted exactly equal by all 4 algorithms.

Figure 65 was captured in a laboratory with artificially placed walls. In this

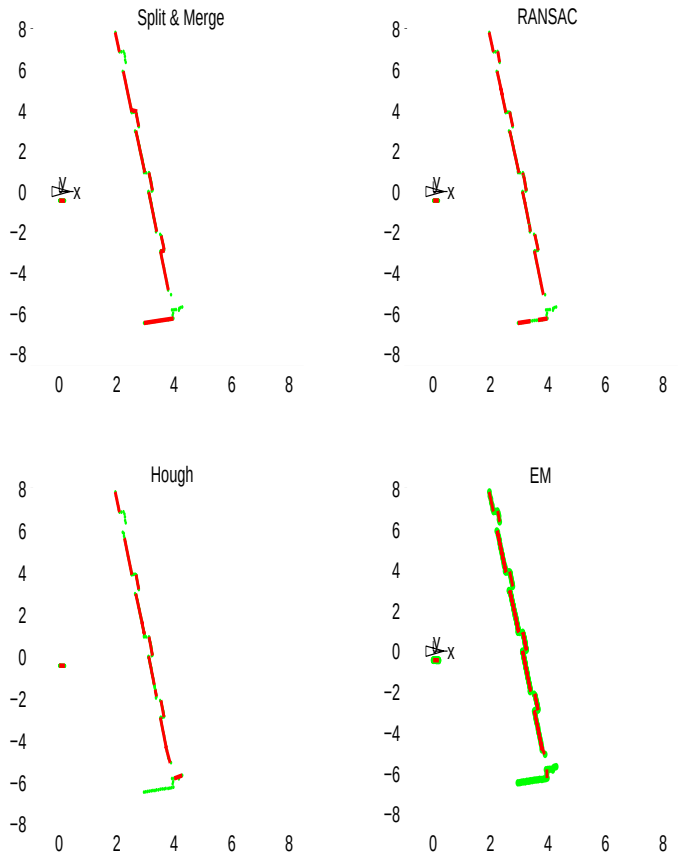

Figure 64: Results of all algorithms on equal scan data [2]

way corners and gaps with behind situated walls were created. In this scan almost all points should be assigned to 5 major segments. The results are rather good in all cases but I want to use this series to show the particular characteristics of each of the algorithm.

Split & Merge is able to use corner points for two different segments. In this case it's very good to recognize how the generated vertices are used to build a connection between two segments thus between two segments are no gaps. In case of the large segment on the bottom the segment was separated due to only 1 scan point which had a larger distance than the specified threshold but wasn't declared as outlier or as inter segment gap. So this point lead to a separation of a major segment with a gap with ≈ 20 centimeters.

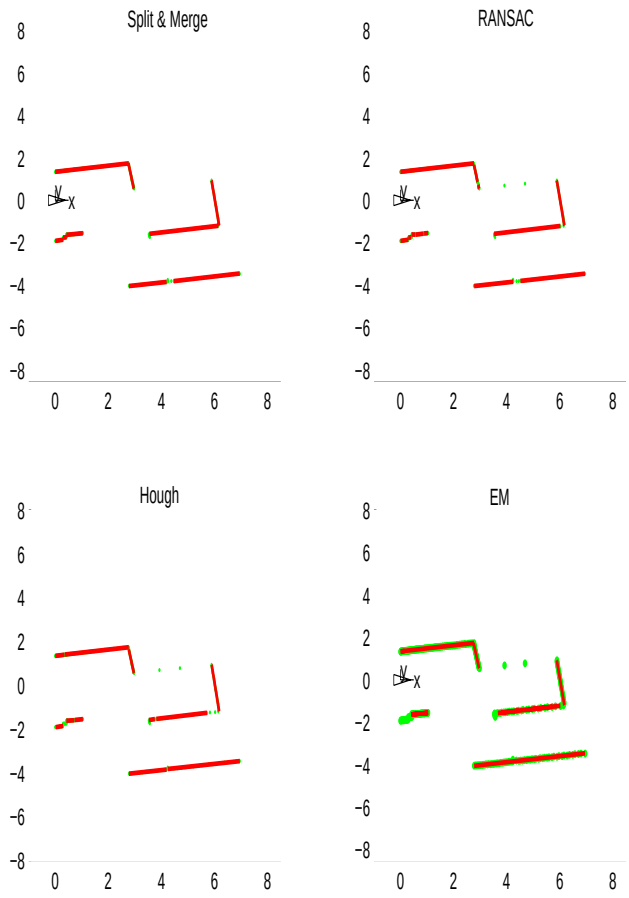

Figure 65: Results of all algorithms on equal scan data [3]

RANSAC shows on the bottom segment and on the right segment on top its typical fragmentation characteristic.

In this case the problem on reduced data with Hough can be seen very well. In the corner on the right one segment considers some points of the orthogonal segments as compatibles (respectively the points within the specified line error tolerance). So these points aren't at the other segments disposal. The segment on the bottom isn't completely situated in a angular corridor of a particular θ-step so the segment gets separated with a gap around 10 centimeters.

EM shows for his means very good results. All major segments are de-

tected without any separation. Only two small segments on the left side aren't detected though they are specified by 10 and 20 points.

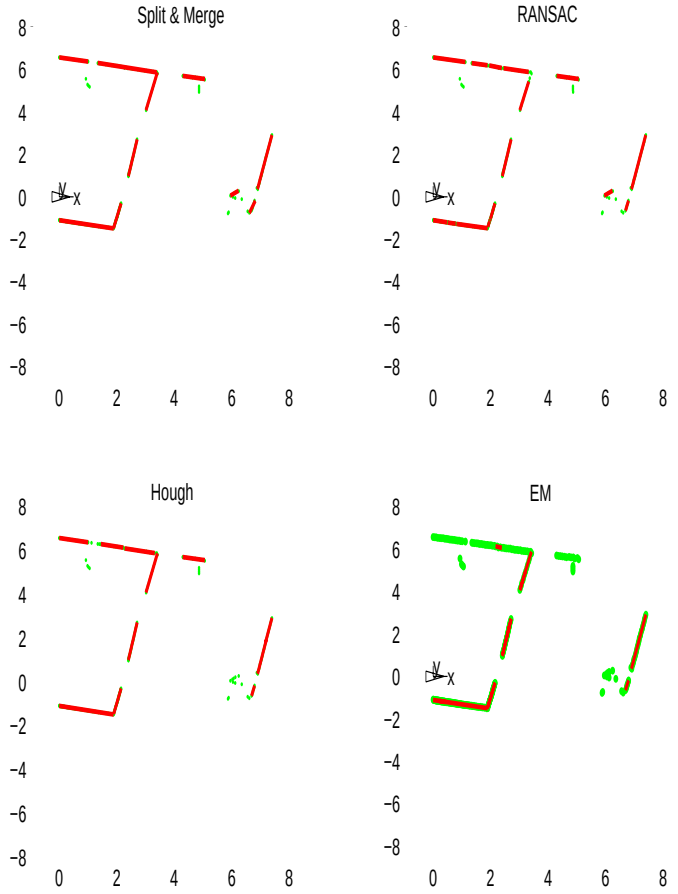

Figure 66: Results of all algorithms on equal scan data [4]

The last figure I want to present illustrates a very typical result for all algorithms. The used scan in Figure 66 was captured inside a laboratory with separated wall segments and behind situated walls inside the laser range. Some obstacles produce point clouds which are vaguely to identify as segments.

Split & Merge produces very good results. It detects all major segments as well as the minor ones. From vague point clouds it extracts the most

probable segments which comply the segment requirements. Corners are closed.

RANSAC detects all major segments though in many cases these segments are separated and contain gaps.

Hough also detects all segments though in some cases segments are separated due to the inconvenient angle. Small point clouds without major orientation are ignored.

EM ignores some major and obviously existing segments. But neither it is affected by small vague point clouds.

Computational Time Comparison

By the means of the parameters which were used to obtain the above given results the following average times were measured (Fig. 67):

Figure 68 shows a chart of the measured average times without the time captured from EM. EM needs more than 2700% longer than Hough so the subtle differences of the other algorithms wouldn't be detectable anymore.

	time [sec]
Split & Merge	0.01864
RANSAC	0.00784
Hough	0.06613
EM	1.8695

Figure 67: Measured average Times of all Algorithms

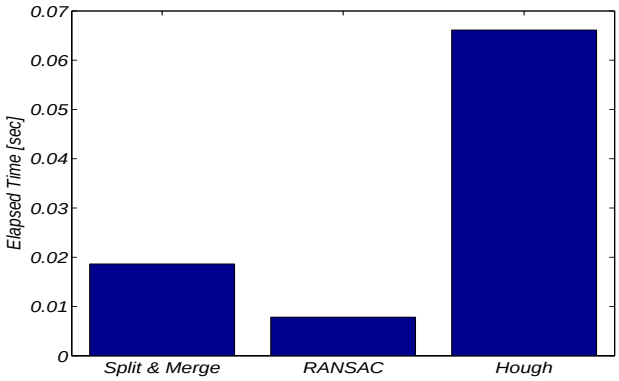

Figure 68: Chart of the measured average times

Used Points Comparison

Table 7 shows the measured values of used Points. Evidently the average number of valid measurements is larger in case of RANSAC and Hough since they don't apply an Outlier filtering. With Outlier filtering the average value accounts to 348.29 which represents 96.48% of the whole captured scan data.

In case without Outlier filtering the value accounts to 351.68 representing 97.41%. Thus in average 3.39 Points are considered to be Outliers.

	Valid Points	Used Points	%-value of valids
Split & Merge	348.29	317.52	91.17%
RANSAC	351.68	256.52	72.94%
Hough	351.68	287.25	81.68%
EM	348.29	251.20	72.12%

Table 7: Average Number of used Points from Measurement Set

A weighting of theses results only is relevant if we can be sure that the major part of the scan data can be considered to be correct. With the definition of segment requirements (criteria) we tried to evaluate an estimation of the quality of the scan data though this still doesn't give an estimation about the rate of good scan data.

Though in general can be said that the more measurements are used the better it can be considered to be, since due to the small laser error tolerance the major part of the data can be considered to be correct and that the major part of the environment consists of segments which should be relevant for a later mapping.

Robustness Test
A typical test on *Robustness* of a segmentation algorithm is the test of finding a line in an unordered point cloud. Therefore a defined area is filled with a specified number of randomly distributed points. The same number of points specifying a segment are set into this area thus we know that 50% of the data specify relevant segment data. A robust algorithm should find the line exactly without being influenced by the random points. This test only can be applied for algorithms that don't need an ordered group hence Split & Merge was excluded.

Figure 69 shows such a point cloud on which RANSAC, Hough and EM were applied. On the top left the pure point data is given. The further sequence consists of the results given by RANSAC (top, right), Hough (bottom, left) and EM (bottom, right).

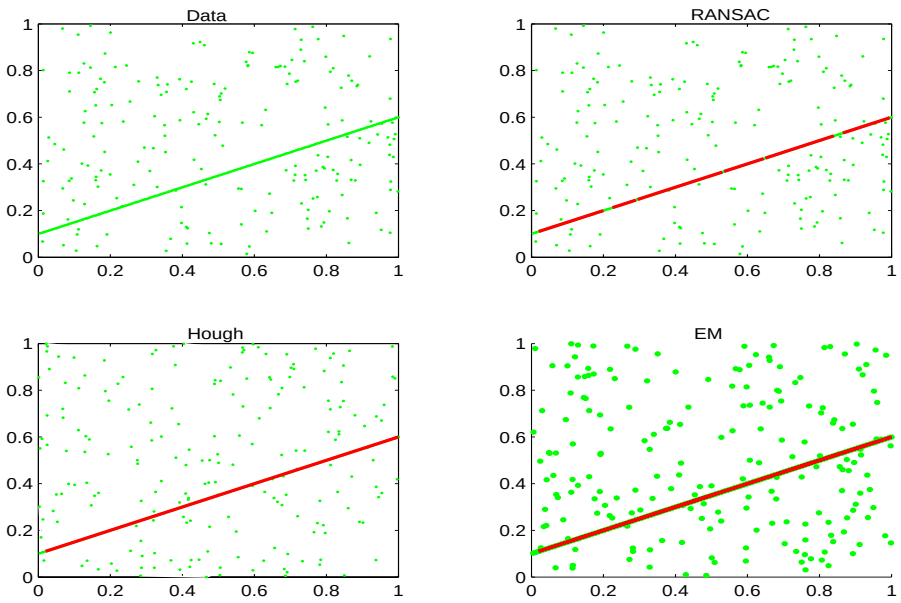

Figure 69: Test on Robustness of RANSAC, Hough and EM

In general all the algorithms produced the correct results.

In this case RANSAC produced a separated line though in other passes this was not the case. Obviously this depends on the randomly generated lines.

The quality of the resulting segment by Hough obviously depends on the angle the line has got. If it lies in a specified Hough-θ-corridor it will be found and not separated. Is this not the case in general the line will be found but the segment will be partitioned.

EM found the correct and complete line in every pass the test was applied.

This test was executed further times with less relevant data compared to the number of points of the complete set.

EM produced the desired results until a rate of valid segment data of 30%. Below this value it converged towards two undescript points which weren't part of the sought segment.

The results of Hough depend on the angle and the number of points which are used. So the rate itself is not essential for Hough but the absolute number of relevant data and the specified θ-step. Thus an essential characteristic

of the given point set is the *Density* of the data. With a number of 200 given Points in an area of $1\,m^2$ Hough provided the correct results until a rate of around 10%. But as explained above the results of Hough only are conditionally relevant for a comparison.

The results of RANSAC showed its typical characteristic of fragmentation. Though the probabilistic parameters were changed accordingly to the rate of relevant data it produced below 60% more and more segment particles with increasing size of the gaps. Nevertheless the segment only was fragmented but the general direction was found.

8.2 Characteristics of Split & Merge

The results showed that Split & Merge produces very good results in short time.

In only few cases it didn't produce an optimal result regarding the required criteria. The procedure of iterative splitting and merging to optimize the results doesn't provide weightings on particular segments which could be important to find small segments in the same way as large ones. On the other side it provides no information about the relevance of the different segments since the segments are extracted simultaneously containing no information of segment relevance.

It is able to produce closed segment connections by using one measurement (a vertex) for two segments so Split & Merge thus can provide closed segment series.

One problem of Split & Merge is its sensitivity to outliers so they have to be filtered out. Another big disadvantage is the requirement on a ordered list of measurements. Split & Merge only works if the data is available in an ordered list to specify measurement adjacency.

There are only few parameters which specify the mode of operation of Split & Merge. The only relevant one is the maximum distance that is used as threshold for a splitting. This arbitrary value can be chosen accordingly to the characteristics of the environment or the required task. If an arbitrary choice should be avoided a further possibility is to use the error tolerances, respectively the covariances, of the measurements and to decide against a split only in case if the uncertainty of the line and the points give the probability of a distance =0.

Nevertheless Split & Merge produces in general the best results of the pre-

sented algorithms.

8.3 Characteristics of RANSAC

The big advantage of RANSAC is its velocity and its independence of the size of the measurement set. Its complexity is only dependent on the specified probabilistic parameters. If the probabilistic distributions of the scan are known or well estimated it produces very fast good results. Furthermore RANSAC can be considered to be unsensible against outliers.

A disadvantage of RANSAC certainly consists in the iterative proceeding on reduced data sets. This characteristics avoids to obtain any good results as desired. The choice of randomly generated lines leads to increasing separation of large segments thus RANSAC can be considered to be sensible to a large sensor error tolerance.

The parameters of RANSAC are easy to estimate accordingly to the task. z only depends on the self chosen, desired probability of result-quality. ω specifies the probability of finding a correct line. In general the rate of relevant data of the entire measurement set can be considered to be much higher than the irrelevant data[23] so ω depends on the number of segments and their respective number of points. To ensure the finding of correct data we should tend to chose ω smaller than necessary since an exact a priori knowledge of the scan in most cases isn't available. The value of maximum distance for points to comply compatibility to the line should be chosen reasonable taking into account the desired "resolution" of the resulting segments as well as the error tolerance of the sensor.

Nevertheless RANSAC by far is the fastest of the presented algorithms and provides in most cases good results.

8.4 Characteristics of Hough

Through its "Voting Characteristic" Hough is able to produce a weighted set of the potential segments. The revoting version of Hough always finds the currently most relevant segment first though this has to be payed with repeatedly transformation and a proceeding on a reduced data set. The "rougher" the segments are situated the faster Hough by revoting can be

[23] See the rate of number of estimated outliers compared to the entire set.

working thus with a priori knowledge of the environment the optimization possibilities are very good.

The two different versions are showing how flexible the Hough space can be used to acquire likely results hence the possibilities to expand Hough are immense e.g. by filtering the Hough space or varying clustering algorithms etc. .

By varying the Hough parameters the results can be obtained in almost any desired quality though this obviously would increase the computational costs significantly.

The choice of parameters depends on the desired compromise between velocity and quality of the results. By decreasing the size of θ- and ρ-step any desired resolution of the scan can be obtained though obviously the limits are specified by the sensor error tolerance. The maximum distance parameter for the finding of line-compatibles points depends directly on the chosen resolution of ρ. The breaking condition for the algorithm depends on the given criteria of minimum segment point number which can be estimated by regarding scan resolution, i.e. potential distances between measurements, as well as the desired point density of a segment.

Hough can be considered to be a very flexible algorithm with plenty of possibilities of algorithm tuning and optimization thus the previously described modes of operation still don't tap the full potential of the Hough Transformation.

The obtained results showed that Hough is able to produce very good results in an acceptable necessary time.

8.5 Characteristics of EM

Evidentally the big disadvantage of the Expectation Maximization algorithm is the necessary computational cost. The in this project presented algorithm has no possibility to reach the velocity of the other presented algorithms.

An advantage of EM could be considered with the initialization with a certain model. In case of a priori knowledge of the environment a high quality of the results can be acquired. It's convergence characteristic and the iterative adding and terminating of models makes it possible for EM to provide any desired result-quality if the computational time can be considered to be less relevant. Furthermore EM provides already after few iterations a

relevant model thus it gives the possibility to proceed as long as you want but always holds a current solution model.

In addition EM showed the best characteristics on an unordered point cloud with partly relevant data.

The parameters of EM are not easy to evaluate analytically. σ influences the velocity of convergence due to it's influence on the weightings for the particular single models. A small value leads to fast and significant optimization but this also leads to a fast elimination of potentially good models, which in this case are considered to be irrelevant for the result. The number of maximum iterations has to be found experimentally but obviously leads to better results the higher it is. The breaking condition parameters which should define the quality of results are varying significantly and so should be chosen sufficiently low to ensure to proceed a procedure break only in case of absolute certainty of good results. The specification of the initial model affect the quality of results essentially. Hence the possibilities of algorithm optimization can be considered best in case of prior measurement knowledge.

Nevertheless EM currently can't be considered to compete with the other presented algorithms though in special cases, e.g. unordered point clouds, its results are very good and there could be a lot of possibilities of optimization.

8.6 Conclusion

The in this project presented algorithms all are able to produce good segmentation results depending on parameters or time. The very different algorithm paradigms showed different strengths and weaknesses hence each algorithm could be recommended for particular tasks.

- If computational time is the most relevant characteristic of the task RANSAC would be the best solution.
- In case of required high quality results Split & Merge should be the choice. Further it depends only on few parameters which have to be specified.
- The Hough transformation produces in most cases very good results. As well it gives the possibility of weighted segments and flexible optimization possibilities.

- EM could be produce very fast a good model state in case of prior knowledge and respective initialization though in our case this isn't the case. As well it shows good characeristics in case of highly randomly distributed data.

So each of the algorithms can be considered to be relevant in a certain particular way for segmentation embedded in the operation cycle of a mobile robot.

9 Summary and Perspective

9.1 Summary

In this thesis we presented a project of analysis, implementation and comparison of different segmentation algorithms treating on 2 dimensional laser captured sensor data. Segmentation of sensor data constitutes an embedded part of the complete data proceeding process in the field of mobile robotic.

The objective of segmentation is to extract geometrical segments from an ordered list of measurement data which can be used in the following process step to build a virtual map. This map in turn is used to allow an error free navigation in the mobile robot environment.

After an introduction to the available Robot Hardware and the used Development Software in chapter 2 we introduced the geometrical and stochastic fundamentals which were used through this project in chapter 3. Beside the different trivial geometrical objects we presented different possibilities of geometrical representations with the *Cartesian Space* and the *Polar Space*. One of the most relevant relations of geometrical objects consists in the distance whereby we presented the *Euclidean Distance* and the *Mahalanobis Distance*. This statistical estimation takes into account the statistical distribution of error affected sensor data, which was presented furthermore in this chapter.

To begin with the algorithms we had to introduce to the requirements, we are asking for, to define a segment which is specified primarily by mere measurement points. These requirements consist of a set of parameter thresholds which have to be complied by the later segments.

The first part of the complete algorithms consists in the *Preprocessing*. This is necessary to prepare the raw scan data for further data proceeding. Thus in chapter 4 we presented the necessary procedure of *Range Filtering* and the further *Outlier Filtering* which was applied in some cases.

In chapter 5 we started to describe the mode of operation of the different algorithms. The chosen algorithms were *The combined Split & Merge*-Algorithm [JaScKa95], the *Random Sample Consensus(RANSAC)*-Algorithm [FiBo81], the *Hough Transformation* [JaScKa95] and the *Expectation-Maximization-*

Algorithm [LCBT01] & [DeLaRu70].

In the beginning we presented the Split & Merge algorithm and the particular parts it consists of. The iterative application of polyline splitting and bottom-up merging was shown as well as the mode of mutual complementing.

The next algorithm was specified with the RANSAC algorithm. After introducing to the parameters of probabilistic description of scan measurements the way of randomly generated lines and extracting the respective points was presented.

Further we introduced the Hough-Transformation. We presented the procedure of transforming the available 2-dimensional measurement data into the discrete 2 dimensional line parameter space. Each measurement proceeds a voting on particular coordinates, which are defining with its own dimensional parameters a particular line in the 2 dimensional cartesian space. The characteristics of a *Voting Algorithm* were shown by introducing the *Accumulator* which is used to hold the cumulated votes and finally the weighted discrete line parameters. Different ways of extracting the local maxima were demonstrated by the implementation of the *Hough by Revoting* and the *Hough by Neighbourship* Algorithm.

The last algorithm we presented was the so called *Expectation Maximization* Algorithm. Its fundamentals lies in a specified line model which in an iterative procedure of calculation of probabilistic expectations and the optimization of the current model parameters gets improved.

To complete the cycle of segmentation we introduced the *Postprocessing* procedure in chapter 6. Each of the algorithms produces equally formatted point groups of *potential Segments*. To generate the resulting segments several steps have to be carried out. The different implemented steps consisted in *Segmentation* of the point groups, *Line Generation*, where we presented the algorithms of *Total Regression* and parameter extraction using the *Discrete Information Filter*, the elimination of *Segment Overlappings*, *Endpoint Acquirement* and the terminating *Length Check*. The final results are uniformly formatted and build the output of the Implementations.

In chapter 7 we described the algorithms and their relevant parameters more detailed.

Firstly we analysed the usage of uncertainty handling and came to the conclusion that it could be applied but in these special cases the differences between the results were very small. So we concluded the usage of uncertainty

characteristics as useful but the improvement are standing in no relations to the additional used computational times. Finally the uncertainty handling was implemented as off turnable feature in the prototypes.

In the following chapters we presented first results of the algorithms and the variations in quality and used time depending on the variation of the algorithm parameters. Regarding the Split & Merge algorithm we modified basically the used threshold for the split-decision and illustrated the variation of the results and measured times. Further we introduced different modes of operation for the Merging-part which consisted in using the *Maximum Normalized Error* and/or using angular deviations.

For RANSAC we presented views of specifying the parameters and pointed out the importance of the segment criteria for the proceeding of RANSAC. We demonstrated the results and came to the conclusion that an improvement of the results could be achieved by the extension of the RANSAC algorithm by adding a previous application of the Split algorithm. By comparing the varying results we demonstrated the advantages of the extended RANSAC algorithm. After examination of results and characteristics we concluded the extended Ransac version as the superior solution.

Furthermore we introduced the differences of distinguished Hough - Transformations which are consisting in the maxima extraction of the accumulator. We described the differences of parameters and illustrated the compared results. Especially we introduced the clustering algorithm applied by the Hough by Neighbourship algorithm and pointed out the advantages and disadvantages of both versions of Hough. In effect the biggest difference constitutes the repeatedly calculation of the accumulator in the Revoting version and the iterative search of the current global maximum whereas the Neighbourship version only calculates the accumulator once and tries to extract all local maxima by clustering the contents. In various illustrations we compared the results and pointed out the advantages and disadvantages of each of the versions. Finally we concluded an optimized version of Hough by Revoting with a decreased number of discrete θ-values as the best solution regarding a compromise between quality of results and computational costs.

The last algorithm was presented with the Expectation-Maximization-algorithm. Primarily we introduced the parameter σ and its altering effect to variations. Further we demonstrated the importance of the initialization of the starting model and showed different possibilities to achieve this with the respective results. In effect the possibilities of model initialization resulted

in choosing randomly generated lines or an initialization with a systematic model. The different options of breaking conditions were presented as well as an estimation of the respective parameter specification. After illustrating several series of results we assembled a set of parameters which were found to be considered as a reasonable compromise between quality of the results and computational costs.

In the final chapter (chap. 8) we applied a direct comparison between the algorithms. Therefore we used with each algorithm a set of parameters which previously were considered to be optimal. By means of several sequences of finally obtained segmentation results on equal scan data we demonstrated standard cases as well as untypical examples of obtained results. Further comparisons were presented regarding necessary computational time and number of used measurements. A final test consisted of a comparison on algorithm robustness which only was applied on RANSAC, Hough and EM.

The final paragraphs treated on pointing out the the particular characteristics, advantages and disadvantages of each of the four implemented algorithms. In a final consideration we concluded particular tasks for which each algorithm could be considered to be optimal or a reasonable choice of application.

9.2 Perspective

The implemented Software is intended to be embedded into the complete procedure of data proceeding given by a mobile robotic task. The Software module receives as input the raw laser scan data and produces as output a set of estimated environment segments. So far the software only was tested and simulated *offline* i.e. only on the given test data. A further step should be the integration of the segmentation algorithms into a real time robot application.

As well there exist some starting points to advance the algorithm development.

The Hough Transformation could be extended by further accumulator proceedings like filtering (e.g. smoothing) or a different clustering algorithm (slope clustering) could be applied.

In case of scan series which are captured with small location differences EM could be modified by taking into account the previous obtained results

to initialize the model.

The current implementation of a graphical output under C/C++, using the free library *Allegro*, was developed task-specific. For further implementations of distinguished procedures a general usable graphic module could be useful for further C-simulations.

A Developped Software

A.1 Prototypes developed by using MATLAB

The Software Prototypes were implemented under the development environment MATLAB.

MATLAB is a scripting language with the possibility of defining function-scripts and source them out into files (*.m). Thus the Software consists of a collection of function files which will be listed in the following. Only the most important files will be listed and explained.

The given directories are named as follows:

```
proceed_splitandmerge
proceed_RANSAC
proceed_hough
proceed_em
```

A.1.1 MATLAB Software - Split & Merge

Mainfile: This is the main script of the Programm.

```
proceed.m
```

Parameter Initialization and Preprocessing:

```
initCriteria.m
initMahalanobis.m
validCoords.m
checkPointPointDistance.m
```

Split & Merge Algorithm:
Recursive-iterative Split & Merge algorithm with subfunctions.

```
splitAndMerge.m
split.m
splitRec.m
checkPointLineDistance.m
perpDist.m
getNormMaxErr.m
```

Postprocessing with Segmentation, regression line, endpoint acquirement.

```
createSegments.m
clearGapPoints.m
createRegLines.m
computeTotReg.m
ReglineByFilter.m
calcRegEndPoint.m
```

The preprocessing and postprocessing step use more or less the same function for each algorithm hence in the following I only will list the most important files for each algorithm. In some cases each the implemented functions have the name of the algorithm as prefix.

A.1.2 MATLAB Software - RANSAC

RANSAC procedure with estimating the maximum number of tries and random line generation with subfunctions.

```
ransacEstim.m
getNumOfTrials.m
singleRandPair.m
oneLineRansac.m
```

A.1.3 MATLAB Software - Hough

Hough by Revoting: Transformation, proceeding accumulator, maximum extraction and compatible point extraction.

```
hough_getRhos.m
hough_getVoteValues.m
hough_getMaxParam.m
hough_getClosePoints.m
```

Hough by Neighbourship: Transfomation, proceeding accumulator, clustering with subfunctions, maximum parameter extraction and subfunctions.

```
hough_getRhos.m
hough_getVoteValues.m
hough_getMaxAndClusters.m
hough_testOnNeighbourship.m
hough_markNeighbours.m
hough_extractLocMax.m
```

A.1.4 MATLAB Software - EM

Model initialization, adding randomly generated lines, Correspondences and Expectation Calculation, Model optimization, Convergence Check and Final compatible line extraction.

```
clearModel.m
addRandomLines.m
calcCorresp.m
calcExpect.m
calcMaxEp.m
optimizeModel.m
checkConvergence.m
createLineParam.m
```

A.2 Algorithm Implementation under C++

The algorithms were developed under the development environment Microsoft Visual C++©. The complete software is organized as a MS-VC++ - workspace in which each algorithm is specified as an embedded Project.

The modules were implemented as C++-Classes and except the algorithm classes all classes are used by all algorithms.

In the following I will give the class-names and the functionality of each class. The most important class members will be presented for each class.

A.2.1 Common Classes

```
=========================================================================
class c_scandata

Members:
    int num_scans
    char *data_file_name
```

This class was used to read from the given data files the raw scan data. It opened the specified data file and hold the number of entire scans and the respective scan data.

```
=========================================================================
class c_scan

Members:
```

```
    long double pol_elements[361]
    int *valids_indices
    int num_valids
    long double *points_x
    long double *points_y
```

This class contains one entire scan as polar elements. The entire functionality regarding a scan were implemented here. Conversion to cartesian coordinates as well as the range filtering. After the range filtering the indices of the valid coordinates are hold as well as the respective number.

This class contains the functionality of extracting from a given point set compatible points to a line specified by input parameters as well as deleting them from a point set. All the methods were implemented for polar and cartesian coordinates as well as for different line representations.

```
=============================================================================
class c_parameters
```

This class reads all specified parameters from the general parameter file `parameter.dat`. In this file all relevant parameters are defined and can be read on runtime.

```
=============================================================================
class c_lines

Members:
    int num_lines;
    int num_segments;
    s_cartarray *lines[MAXIMUM_LINES];
    s_segment *segments[MAXIMUM_LINES];
```

This class holds the complete functionality regarding point groups, lines and segments. The lines are hold as cartesian point groups in a defined structure called `s_cartarray` in an array of structure pointer `*lines` with a fixed maximum number. Besides trivial basic functions like `eraseLine` or `addLine` the entire postprocessing is implemented in this class. Therefore the following major methods are implemented:

```
Methods:
    void clearInvalidSegments(void);
    void clearOverlappings(void);
    void createAllSegments(void);
```

```
    s_pol_line calcTotalRegression( s_cartarray* );
    s_pol_line calcRegByInfFilter ( s_cartarray*);
    float segmentLength(s_segment*);
```

The point groups which are hold as members are checked, split etc. due to the given segment criteria. An overlapping check is realized as well as the generating of the segments by the point groups using Total Regression or the Discrete Information Filter. Additional Endpoint Acquirement and Length Check.

The generated segments are hold as class members in an array of a defined structure called `s_segment`.

```
=========================================================================
class c_time
```

This class was implemented to provide an easy to use object to proceed time measurements. The time measurements are realized by the ANSI-C function `clock()`.

```
=========================================================================
class c_figure
```

In this class the graphical output was implemented using the graphical library *Allegro*. Due to the introduction and usage of an external library I want to give a more detailed introduction to this library and the class implementation later in this appendix.

A.2.2 Algorithm Classes

Split & Merge

```
=========================================================================
class c_splitandmerge

public:
    s_cartarray clearOutliers(s_cartarray*, float);
    float getPerpSegDistance(/* ... */);
    s_indexarray split(s_cartarray*, float);
    s_indexarray splitandmerge(s_cartarray *, float);
protected:
    float getMaxNormError(s_cartarray*, int, int);
    void getVertice(s_cartarray*, s_indexarray*, float, int, int);
```

```
    int tryOneMerge(s_cartarray*, s_indexarray*);
    int merge(s_cartarray*, s_indexarray*);
    void qsortIndexArray(s_indexarray*, int, int);
    void qswap(s_indexarray* , int , int );
    s_indexarray uniqueIndexArray(s_indexarray*);
```

This class holds all the methods for the Splitting and Merging. It doesn't hold class members thus some methods which had to be used external (e.g. Extended RANSAC) were realized as public.

It holds the methods for the Outlier Filtering as well as the complete splitting and merging. As a subfunction a Quicksort of an index structure `s_index_array` was implemented.

RANSAC

```
=============================================================================
class c_ransac
    int max_tries;
public:
    int getMaxTries(void);
    s_pol_line createRandomLine(s_cartarray*);
```

This class holds the necessary methods for RANSAC. It only contains a generation of `max_tries` by using the parameters read from the parameter file and a generation of a random line (defined structure `s_pol_line`) using an input point group.

Hough

```
=============================================================================
class c_hough
    int *accu;
    int (*indexed_accu) [3];
    s_indexarray maximums;
public:
    void buildAccu(void);
    void voteAccu(s_cartarray);
    s_accu_max getAccuMax();

    int buildIndexedAccu(void);
    void qsortIndexedAccu(int, int);
    void markNeighbours(int, int);
    s_indexarray getMaximumsByNeighbourship(void);

    float getRoundValue(float,float);
```

In this class the functionality of the different Hough versions were implemented. As class member it holds a dynamically allocated accumulator according to the given parameters. Further it holds the used structure for the clustering algorithm and the respective array of extracted maxima.

EM

```
==========================================================================
class c_em {

private:
    int num_P;
    int num_L;
    int num_old_L;

    long double **E;
    long double *Ep;

    s_vec_line *model[MAX_MODEL_LINES];
    s_vec_line *old_model[MAX_MODEL_LINES];
    s_cartarray points;
public:
    int initModelRand(int);
    void initModelSys_big(void);
    void initModelSys_media(void);
    void initModelSys_small(void);
    int initModelVectors(s_vec_line*, int);

    int addVectorModelLine(s_vec_line*);
    int deleteModelLine(int);
    int addRandomModelLines(int);

    void calcE(void);

    s_sol_matrix calcSolutionMatrix(int);
    s_vec_line calcOptVector(int);

    void one_EM(void);

    int checkConvergence(void);
    int clearDoubled(void);
    int clearFewMaximums(void);
    int clearZeroVectors(void);
```

This class hold the functionality of the EM algorithm.

The members are the model as an array of defined (vector)line structures `s_vec_line`, a backup array for the model to proceed a comparison for convergence, the given measurements, a dynamical allocated array for the Expectations and the Phantom Expectations.

The methods are: An initialization for the model with the different, in previous chapter described initial models, a possibility for generating a random single model, adding new model lines and deleting an unsupported model. Calculating the expectation values and optimization of the current model. Further the check on convergence and check on unsupported models. The main function constitutes in `one_EM` which proceeds one expectations and one maximization iteration.

A.3 Graphic Library-Allegro

In this chapter I want to describe the implemented class `c_figure` more detailed due to the introduction to a new graphical library *Allegro*.

The free available graphical library Allegro[24] provides the possibility of accessing a graphical window and applying different graphical functions as well as access to the mouse or keyboard driver. Beside direct access on the video memory in our case the most important thing was the possibility of drawing primitives (Lines, Points, Scaling etc.).

The graphical window can be used inside a console application and draws the graphical entities into the separated window. This was very important for us for debugging purposes. The graphical window has to be initialized in the C-main function:

```
allegro_init();
install_keyboard();
install_mouse();
install_timer();
```

So the graphical methods only can be accessed directly in the main-function thus we implemented the class `c_figure` externally from other classes and only call the graphical functions in the main function.

The general mode to access the graphical window was to draw all geometrical entities to a buffer which has to be *"blitted"* into the video buffer after changing the class context. This is due to the slow direct access to the video

[24] http://www.talula.demon.co.uk/allegro/

buffer which would lead to a screen flickering with each alteration. Therefore we hold all entities (Points, Lines, Segments, axis etc.) in dynamical data structures as class members with the previous described data types.

```
=========================================================================
class c_figure
    BITMAP *buffer;

    s_pol_line *drawn_lines[MAX_LINES];
    int drawn_lines_color[MAX_LINES];
    int num_drawn_lines;

    s_cartarray* drawn_cartarrays[MAX_CARTARRAYS];
    int num_drawn_cartarrays;
    int drawn_cartarrays_color[MAX_CARTARRAYS];

    s_segment* drawn_segments[MAX_SEGMENTS];
    int num_drawn_segments;
    int drawn_segments_color[MAX_SEGMENTS];

    s_point* drawn_point[MAX_POINTS];
    int num_drawn_point;
    int drawn_point_color[MAX_POINTS];
public:
    clearClass();

    void redrawFigure();

    void drawScale(void);
    void drawPoint(float x, float y, int color);
    void drawPoints(s_cartarray input, int color);
    void drawPolyline(s_cartarray*points,s_indexarray*vertices,int color);
    void drawPolarLine(float theta, float rho, int color);
    void drawVectorLine(float x, float y, int color);
    void drawCartLine(float x1, float y1, float x2, float y2, int color);
    void drawSegment(s_segment segment, int color) ;
    void drawSegments(s_segment* [MAX_SEGMENTS], int , int color);
    void drawCartarraySegments(s_cartarray* [MAX_SEGMENTS],int,int color);
```

The used buffer has the library-specific type `BITMAP`. Easy to see from the name that all functions work directly as bitmap primitives. In the above given class definition only the most important members and methods are given due to a large overhead on hereinafter irrelevant data structures.

All the drawing functions add the input entities directly, or after required conversion, into the entity "containers" of the class with the respective used

color. After changing the containers the function `redrawFigure()` copies the entire context of the entity containers into the `buffer` and proceeds a final blit into the video buffer.

Besides the implementation of drawing primitives the functionality of moving and zooming the context of the graphical windows by keyboard keys was realized.

Figure 70 shows a screenshot of the output given by Allegro applied in the Split & Merge algorithm:

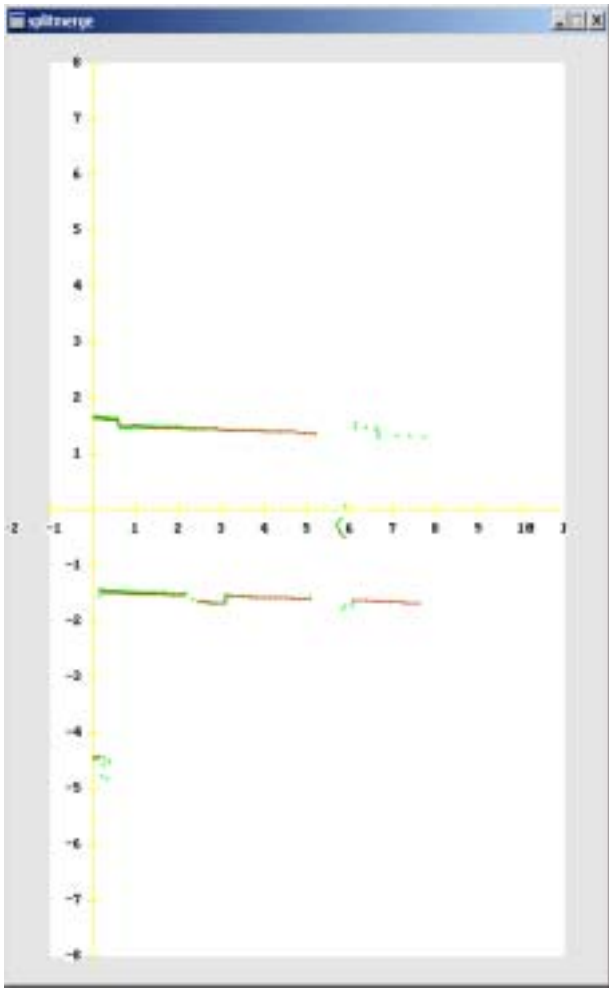

Figure 70: Graphical Screen Output generated by *Allegro*

References

[Cast98] J.A. Castellanos, Doctoral Thesis, *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*, 1998

[DeLaRu70] A.P. Dempster, N.M. Laird, D.B. Rubin *Maximum Likelyhood from Incomplete Data via EM Algorithm*, 1970

[DuHa73] R.O. Duda, P.E. Hart, *Pattern Classification and Scene Analysis*, Wiley-Interscience, 1973

[FiBo81] M. A. Fischler, R. C. Bolles: *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*, 1981

[JaScKa95] R. Jain, B. G. Schunck, R. Kasturi , *Machine Vision*, McGraw-Hill, 1995

[LCBT01] Y. Liu, R. Emery, D. Chakrabarti, W. Burgard, S. Thrun: *Using EM to learn 3D Models of indoor environment with mobile robots*, 2001

[Neira93] J. Neira, Doctoral Thesis, *Geometric Object Recognition in Multisensor Systems*, 1993

[WeBi02] G. Welch, G. Bishop, *An Introduction to the Kalman Filter*, 2002

Declaration

Name:	Gabriel,Josef Nock
Day of birth:	12th of May, 1977
Place of birth:	Furtwangen, Germany
Matriculation Number (Zaragoza)	519 913
Matriculation Number (Konstanz)	268 003

Herewith I declare having my Diploma Thesis with the title:

Segmentation Algorithms for 2D-Laserscans in an indoor environment

realized at the

Universidad de Zaragoza
Dept. de Informática e Ingeniería de Sistemas
Grupo de Robótica y Tiempo Real

under supervision of

Prof. José Neira Parra and
Prof. Oliver Bittel

accomplished independently and without external help as well as having utilized no further help than the mentioned.

Further I declare having indicated the usage of literal cites, tables, graphics, programms from literature or other sources as well as the usage of ideas of other authors on the respective locations inside this thesis.

I'm aware of the potential legal measures on an insincere declaration.

.....

Erklärung

Name:	Gabriel,Josef Nock
Geburtstag:	12.05.1977
Geburtsort:	Furtwangen, Deutschland
Matrikelnummer (Zaragoza)	519 913
Matrikelnummer (Konstanz)	268 003

Hiermit erkläre ich, dass ich meine Diplomarbeit, mit dem Titel:

Segmentation Algorithms for 2D-Laserscans in an indoor environment

durchgeführt an der

Universidad de Zaragoza
Dept. de Informática e Ingeniería de Sistemas
Grupo de Robótica y Tiempo Real

unter Anleitung von

Prof. José Neira Parra und
Prof. Oliver Bittel

selbständig und ohne fremde Hilfe angefertigt habe und keine anderen als die aufgeführten Hilfen benutzt habe.

Desweitern dass ich die Übernahme wörtlicher Zitate, Tabellen, Zeichnungen, Bildern und Programmen aus der Literatur oder anderen Quellen, sowie die Verwendungder Gedanken anderer Autoren an den entsprechenden Stellen innerhal der Diplomarbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

.....