Master's thesis

---

ultrasonic sensor based self localization for autonomous vehicles within mapped environments

---

Graz, University of Technology

Institute for Software Technology

Audi Electronics Venture GmbH

Author: **Altinger, Harald, Bsc.**

Accessor: Wotawa, Franz, Univ.-Prof. Dipl.-Ing. Dr.techn.
Graz, University of Technology

Supervisor: Steinbauer, Gerald, Ass.Prof. Dipl.-Ing. Dr.techn.
Graz, University of Technology

Supervisor: Stümper, Stefan, Dipl.-Inf.
AUDI AG

Graz, February 14, 2013

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ……. ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

……. ……..
date (signature)

Contents

H. Altinger NOTE: short overview about work
G. Steinbauer NOTE: give a short intro to solution

Abstract

Localization is one of the key tasks within robotic applications. The ” Monte Carlo Localization (MCL)” approach, as suggested by [1], has been implemented in combination with the ”beam model for range finders”, see [2], and a new resampling approach, ”Slot Resample”.
Modern day vehicles are equipped with a wide range of sensors, e.g. odometry, sonar, radar, etc. Various experiments were conducted to determine the sensor parameters. To estimate the best parameters for this sensors and methods a simulation has been implemented. The identified parameters have been tested under real conditions within a mapped parking garage. A system overview will be given and the resulting localization performance will be presented and discussed.

Zusammenfassung

Ein Lebensgrosses Fahrzeug soll innerhalb einer realen Umgebung lokalisiert werden. Das Fahrzeug verfügt über Seriensensorik, in der Umgebung wurden keine künstlichen Landmarken verbaut. Basierend auf dem ”Monte Carlo Localization (MCL)” Ansatz von [1] wird ein Modul zur Lokalisierung innerhalb eines geschlossenen Gebädes entwickelt. Es werden unterschiedliche Ultraschallsensoren und mathematische Beschreibungsmodelle untersucht. Faktoren die den möglichen Operationsbereiche bestimmt werden ermittelt. Simulierte Daten werden mit realen Ergebnissen eines Standardserienfahrzeuges in zwei unterschiedlichen Umgebungen verglichen. Das Kapitel Ausblick schlägt Parameter und Komponenten vor um den möglichen Operationsbereich zu vergrössern. Ein Geamtüberglick des Systemes wird dargestellt, dieser beinhaltet weitere Module die für das komplexe Gesamtsystem darzustellen.

1 Introduction

1.1 motivation

According to World Health Organization (WHO) Global Health Risks, see [3], lethal crashes can be reduced up to 61% by engineering measurements, e.g. seat belts. Within the EU-27 zone the number of death people (caused by road accidents) per million inhabitants decreased 131 in 1995 to 78 in 2008, see Statistical Office of the European Communities (EUROSTAT) care database [4]. Statistik Austria lists a total number of 45.858 accidents with injured people on Austrian roads, see [5]. According to various police reports, e.g. [6], the number of accidents only causing car body damage is much higher and caused by carelessly drivers. All those statistical data share a common factor, a human as the driver! Since the early days of automotive, when Nicolaus August Otto invented the petrol engine in 1876 and Henry Ford released his T-Model in 1908 there has always been the need for a driver.

"Stanley", Stanford University's robotic car, showed an autonomous cars ability to make its way by winning Defense Advanced Research Projects Agency (DARPA) Grand Challenge in 2005. A good collection of publication can be seen within [7] and [8]. "Boss", the Tartan Racings robotics car, overall winner of the 2007 DARPA urban challenge, demonstrated the ability to get rid of a human driver, even within urban areas. See [9], [10] and [11] for a medley on this topic.

Various surveys showed that first of all the market requests safety assistance systems, e.g. emergency braking. Drivers are willing to accept fully automated systems to take over in emergency situations, see [12], or if the car is driving on its own when there is no passenger on board the car.

Considering these aspects it is reasonable to develop advanced driver assistance systems that can prevent accidents and assist the driver or to take over. The overall aim will be to develop cars that still look and behave like normal street cars (they do not carry 10 laser scanners and 4 computers like Stanley). Those cars should reduce the number of accidents and release the human from the duties as the driver.

The "Wiener Weltabkommen" from 1968 states: "Every driver shall at all

times be able to control his vehicle or to guide his animals", see [13]. This convention limits the ability for autonomous functions on public roads. Therefor every driver assistance system has to be initialized and supervised by the driver or has to act on private ground.

1.2 project objectives

G. Steinbauer NOTE: motivation for project, tasks to solve

Looking back to previous autonomous systems ([14], [9]) this vehicles look like robots and carry sensors worth more than the vehicle itself. Taking in account market rules, advanced driver assistance systems need to be reliable and cheap. So the first step will be to determine which "in production" sensors, a modern day car is equipped with, can be used.

When looking forward to fully autonomous systems a second step will be to find use cases on private ground, to reduce conflicts with law. Demonstrating such systems that can handle simple tasks, a human driver likes to avoid, e.g. park and pull out a car, maybe increases the acceptance rate.
Never the less it is necessary to develop common modules e.g. a localization system or map representing the actual surrounding. Such a module will be usable for a wide band of driver assistance and autonomous systems.

The overall aim is to use as much as possible already existing infrastructure (blueprints, ground markings, etc.) and components (ultrasonic sensors, Inertial Measurement Unit (IMU), etc.), but to avoid external dependencies (building mounted LIght Detection And Ranging (LIDAR), Radio Frequency IDentification (RFID), etc.).

Figure 1.1 gives a short overview to the target test area. The vehicle should be able to fully autonomous travel along a path with a maximum speed of $4\frac{km}{h}$. The scene will be static, but not all obstacles need to be mapped.

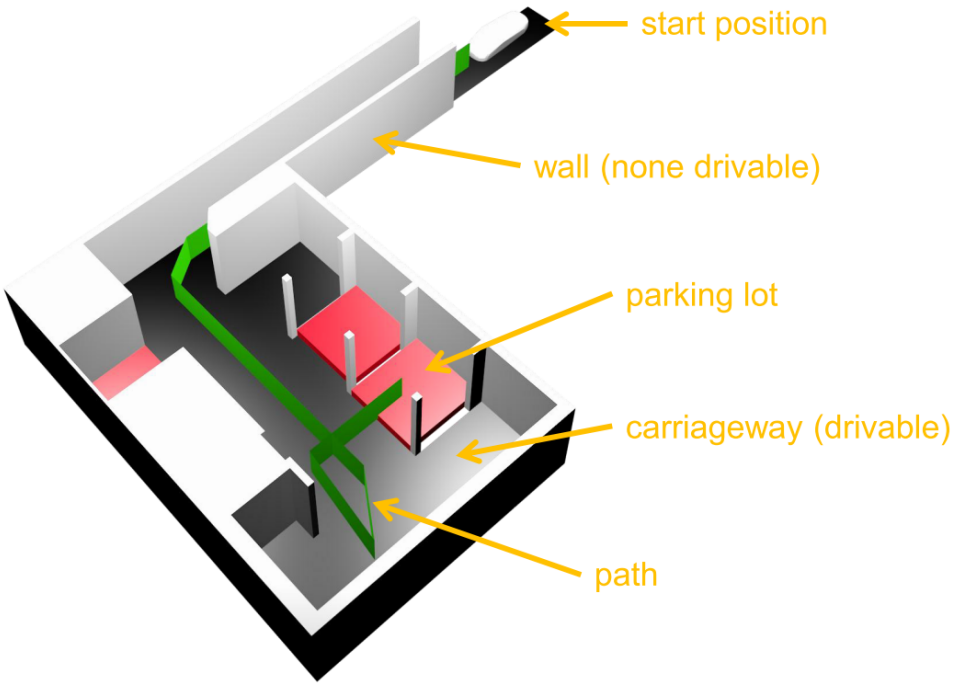


Figure 1.1: The used garage at the first underground level. The car starts at the start position and knows the parking lot to target.

1.3 thesis objectives

G. Steinbauer NOTE: part of project to be solved within this thesis

One core requirement of an autonomous system is to determine its position. Determining a vehicles position with Global Positioning System (GPS) (mostly Differential Global Positioning System (DGPS)) and odometry is a rather good solved task, see [15],[16] and [17], but only applicable when there is a clear line of sight to the sky.

Within this project the application will be indoor. So the receivables of this thesis is to develop a localization module that can be used without GPS support and deals with already standard production "in vehicle" sensors. To enhance the localization already existing maps, e.g. blue prints from the architect, needed to be integrated. The system needs to runable within a fully sized real environment and a state of the art standard production vehicle.

Dealing with sensor means dealing with uncertainty and noise. The implemented system has to cope with this problem. Not every sensor work all conditions, therefor the system has to fusion position hypothesis from different sources to enhance the quality of the resulting position hypothesis.

2 prerequisite

H. Altinger NOTE: using chapter to give literature overview

G. Steinbauer NOTE: give short summary to available literature

2.1 Mathematical overview

This section will give a short overview about the math required for the topic of localization. The following subsections will give a short introduction and explain the required "working tools". References to further literature will be given fore those who are interested in mathematical derivations and proves.

H. Altinger NOTE: using section to give overview about coordinate systems used, transformations, ...

2.1.1 coordinate systems

To make sure everyone is talking about the same point a common description is required, a coordinate system. Within a mathematical definition every vector basis is a coordinate system. It consists of a start point, the origin, and a definition how to reach every point (a number and unity). This can be done by vectors e.g. $\begin{pmatrix} x & y & z \end{pmatrix}$, grid elements, e.g. A10, distance and bearing, etc. Figure 2.1 shows the coordinate system according to Deutsche Industrie Norm, engl. german industrial norm (DIN) 70.000, see [18], which is generally used within automotive engineering.

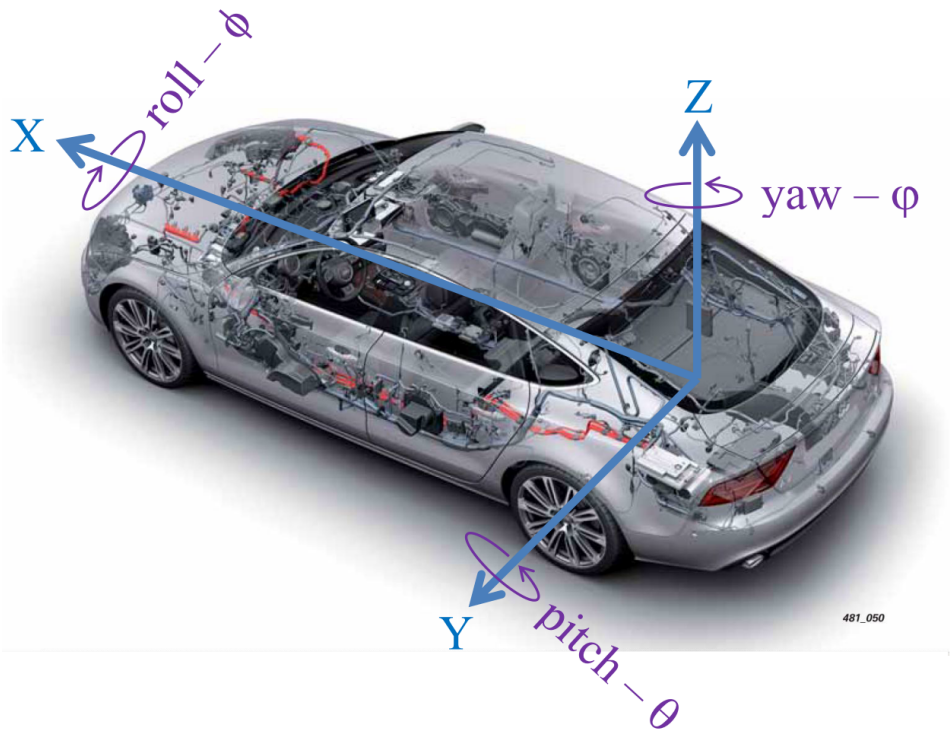


Figure 2.1: the coordinate system according to DIN 70.000

2.1.2 homogeneous coordinate transformation

H. Altinger NOTE: is a short introduction (e.g. this style represents the "working tools") enough or do I need a more detailed description of the theory behind?

In general there is more than one coordinate system, e.g. a sensor measures a distance, the sensor is mounted within a mobile platform which travels through a room. Within this simple case one can see 3 Coordinate systems (sensor, platform, world) with one fixed transformation (the sensor is mounted at a fixed position at the platform) and a dynamic transformation (the platform moves through the world). See Figure 2.4 for visualization.

As known from literature[1] this thesis requires 2 types of coordinate transformations:

- translation
- rotation

1 for more deep mathematical derivations and proves, see [19]

A general object within the plane-world (2D) can be represented by its position (x,y) and its orientation (φ), see equation 2.1. Any kind of transformation can be expressed with a matrix multiplication. The Cartesian coordinates extend by a 4^{th} dimension, the homogeneous 1. In comparison to none homogeneous coordinates every translation and rotation can be described with a matrix multiplication in a similar way. As dealing with complete matrices, the result can be countermand by multiplication with the inverse transformation. If there are multiple constant coordinate transformations, they can be combined within one matrix due to their associative attribute, which enhances performance. The 4^{th} dimension can be used to normalize the coordinates, which enhances homogeneous coordinate transformations to be independent of the used value unit.

$$\vec{x} = \begin{pmatrix} x & y & \varphi & 1 \end{pmatrix}^T \tag{2.1}$$

A plane translation is shown in Figure 2.2 and can be expressed with a translation matrix, see equation 2.2. In general, those transformation matrices sourced by the unit matrix, extended by their transformation values.

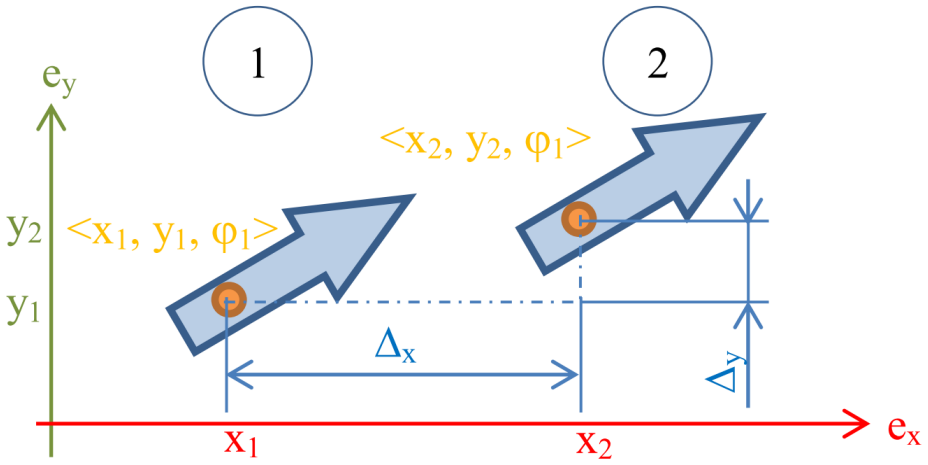


Figure 2.2: The blue arrow moves from position 1 to 2. The x, y and z coordinates changes, but the orientation stays the same. If there is only 2D movement, δ z can be 1.

$$T_{trans} = \begin{pmatrix} 1 & 0 & 0 & \Delta x \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & \Delta z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

A plane rotation is shown in Figure 2.3 and can be expressed with a rotation

matrix, see equation 2.3. Knowing from literature there exists 3 kindes of rotation matrices, T^x_{rot} to describe a rotation around the x-axis, T^y_{rot} to describe a rotation around the y-axis, T^z_{rot} to describe a rotation around the z-axis. As this thesis deals with plane transformations, T_{trans} will refer to T^z_{rot}.

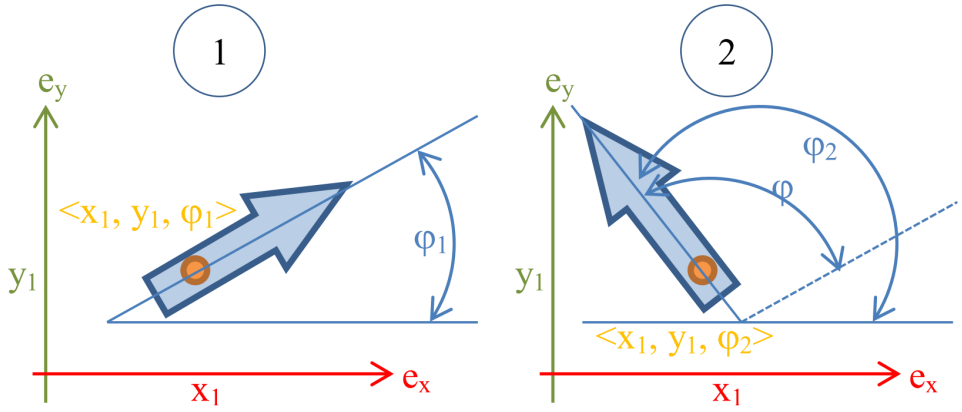


Figure 2.3: The blue arrow rotates around the z-axis from 1 to 2. The φ angle changes, the position stays steady.

$$T_{rot} = \begin{pmatrix} cos(\varphi) & -sin(\varphi) & 0 & 0 \\ sin(\varphi) & cos(\varphi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

In reality there are combinations of translation and rotation. As equation 2.2 and 2.3 describe regular matrices (associative and commutative law apply), they can be combined in any way.
A typical scenario has been described at the beginning of the section. Figure 2.4 explains derivation 2.5 - 2.8

Within Figure 2.4 the blue arrow represents a mobile platform traveling through the world coordinate system. At time T its position is known as x^w_p, y^w_p and φ^w_p. The transformation from world to platform can be expressed by T1 and will change at every T. The sensors coordinate system will be fixed within the platforms system. Its position is known as x^p_s, y^p_s, φ^p_s. T2 transforms sensor coordinates into platform coordinates. As the sensor is assumed to be mounted at a fixed position, T2 will stay constant over time. T3 describes the transformation between a measured point within the sensors coordinate system into the world coordinate system. This transformation consists of T1 and T2 and will change over time.

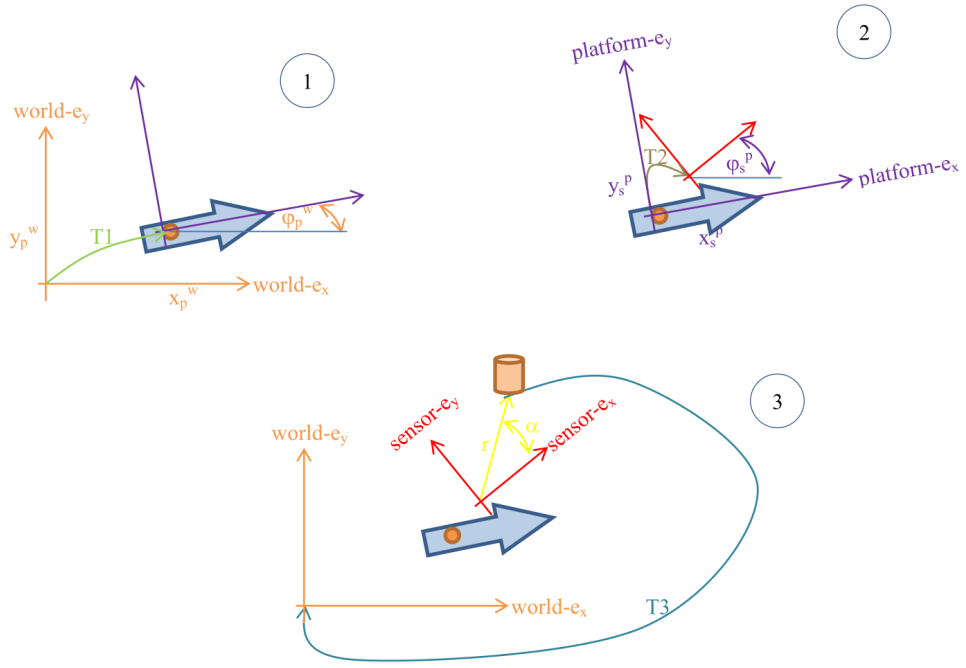


Figure 2.4: coordinate transformation, combining multiple coordinate systems to transform coordinates to base.

$$T1 = T1_{trans} \cdot T1_{rot} \tag{2.4}$$

$$T2 = T2_{trans} \cdot T2_{rot} \tag{2.5}$$

$$T3 = T1 \cdot T2 \tag{2.6}$$

$$\begin{pmatrix} cos(\varphi_p^w) & -sin(\varphi_p^w) & 0 & x_p^w \\ sin(\varphi_p^w) & cos(\varphi_p^w) & 0 & y_p^w \\ 0 & 0 & \varphi_p^w & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} cos(\varphi_s^p) & -sin(\varphi_s^p) & 0 & x_s^p \\ sin(\varphi_s^p) & cos(\varphi_s^p) & 0 & y_s^p \\ 0 & 0 & \varphi_s^p & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.7}$$

$$= \begin{pmatrix} cos(\varphi_p^w)cos(\varphi_s^p) - sin(\varphi_p^w)sin(\varphi_s^p) & -cos(\varphi_p^w)sin(\varphi_s^p) - sin(\varphi_p^w)cos(\varphi_s^p) & 0 & cos(\varphi_p^w)x_s^p - sin(\varphi_p^w)y_s^p + x_p^w \\ sin(\varphi_p^w)cos(\varphi_s^p) + cos(\varphi_p^w)sin(\varphi_s^p) & -sin(\varphi_p^w)sin(\varphi_s^p) + cos(\varphi_p^w)cos(\varphi_s^p) & 0 & sin(\varphi_p^w)x_s^p + cos(\varphi_p^w)y_s^p + y_p^w \\ 0 & 0 & 1 & \varphi_s^p + \varphi_p^w \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

2.1.3 basics in probability

The following section gives a short introduction to basics in probability which is used to explain later on sections. The content is based on literature, see [2] chapter 2. Random variables can assign any value within a defined range, according to specific probabilistic laws. Within robotics random variables can be measurement data (e.g. sensor data) and control update (e.g. desired distance to travel).

When solving localization problems normal distributed noise is assumed. To model this random variables posses Probability Density Function (PDF), an example is given in equation 2.9 the commonly known one dimensional Gauss function. Similar to other Gaussian functions, the parameter σ^2 represents the variance and μ the mean. This is used to model a belief, e.g. a position.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu^2)}{\sigma^2}} \tag{2.9}$$

In comparison to equation 2.9 equation 2.10 is called multivariate, due to the fact that the modeled variable $\vec{x}$ consist of more than one dimension. Σ is called the "covariance matrix". In general terms equation 2.9 is the generalized of equation 2.10 which describes a density function.

$$p(\vec{x}) = det(\frac{1}{\sqrt{2\pi\Sigma}}) e^{-\frac{1}{2}(\vec{x}-\mu)^T \Sigma^{-1} (\vec{x}-\mu)} \tag{2.10}$$

Equation 2.11 states that a PDF always integrates to 1, but the value of a PDF is not upper limited to 1.

$$\int p(x)\, dx = 1 \tag{2.11}$$

The notation $p(x,y)$ denotes a variable X has got the value x **AND** at the same time variable Y 's value is y. In contrast the notation $p(x|y)$ represents the probability of X 's value is x **IF** Y 's value is y. This is the case if X depends on Y. If two variables are independent, their joint probability can be expressed using equation 2.12. If X and Y are independent[1] equation 2.14 can be used.

$$p(x,y) = p(x) \cdot p(y) \tag{2.12}$$

1 e.g. throwing a dice does not have an influence on the value if throwing it a second time, this is called independence

$$p(x|y) = p(X = x|Y = y) \tag{2.13}$$

$$p(x|y) = \frac{p(x,y)}{p(y)} = p(x) \tag{2.14}$$

The Bayes rule can be used to relate $p(x|y)$ to its inverse $p(y|x)$, see equation 2.15 and 2.16 for a third variable Z, representing X 's value is x IF Y 's value is y and Z 's value is z (conditional probability [1]).

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{2.15}$$

$$p(x|y,z) = \frac{p(y|x,z)p(x|z)}{p(y|z)} \tag{2.16}$$

The Markov Assumption postulates that posteriori and a priori sensor data are independent of each other or the current state. Literature, see [2], this can be achieved in reality if the state variable has been chosen appropriate.

2.1.4 Markov chains

A Markov chain describes the transition from one (finite) state to another state, out of a finite number of possible states. The basic assumption states that one state change only depends on the actual state, not on the history before. This is known as memoryless or Markov property. Figure 2.5 shows a typical Markov chain. A simple example of such a Markov chain is rolling a dice (assuming the dice is not cheated).

1 e.g. extracting a ball out of a hut filled with two different colored balls has got an influence on the next extraction, this is called conditional probability

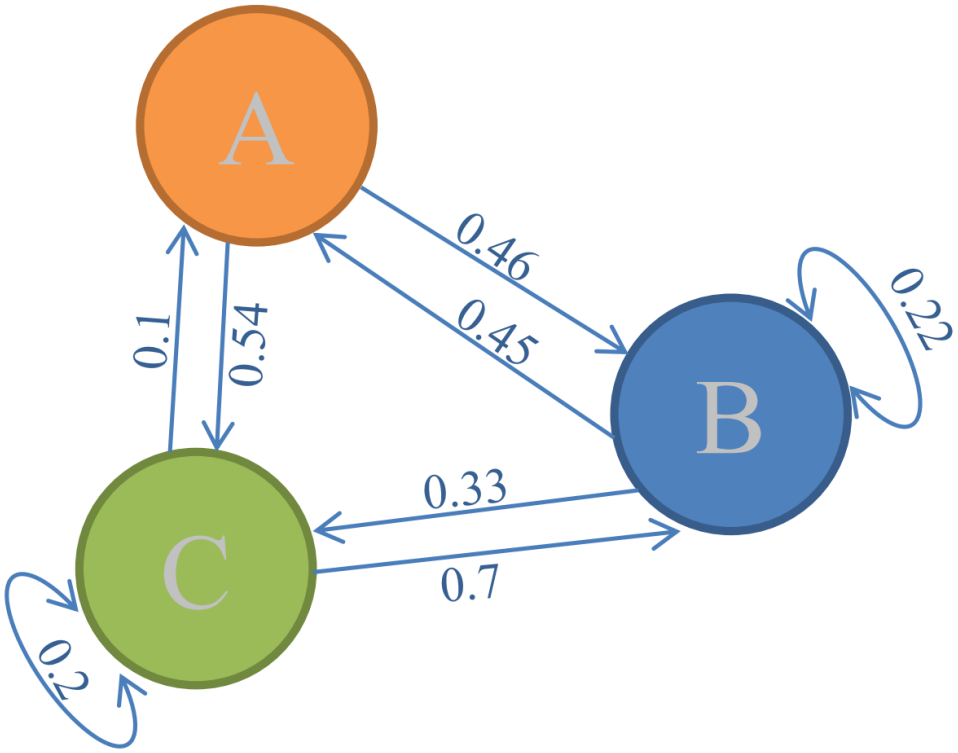


Figure 2.5: Every state A,B,C has got probabilities for a state change. All leaving probability per state sum up to 1. The probability does not depend on the previous state. The number of states and possible state changes is countable.

2.2 maps and grid based world representation

A Map can be seen as an a priori information about the world. This assumption can be valid, because e.g. every building has got a construction permit which requires a permission drawing. But even if such a "map" exists, they do not need to comply with reality, so there is always the need to update the world information. This leads to a "Hen and Egg" Problem: need a map to localize, need to localize to draw a map. Within literature, see [2], this is known as Simultaneous Localization And Mapping (SLAM) or "concurent mapping and localization". They define 4 core problems when building maps:

- **size:** the larger the world, the bigger the map, the hard to maintain, match features and to store it
- **noise:** no sensor is perfect, so no information is 100 % accurate, otherwise map-building would be too easy
- **perceptual ambiguity:** some places look similar to others, some change over time, but they are always hard to differ or recognize

- **cycles:** when moving the same path away and back, noise can be corrected, but if a different way is used a cycle will be created and it is hard to match the start- to the endpoint.

Figure 2.6 addresses these problems. Clearly one can differ the bad influences and results within the above maps.

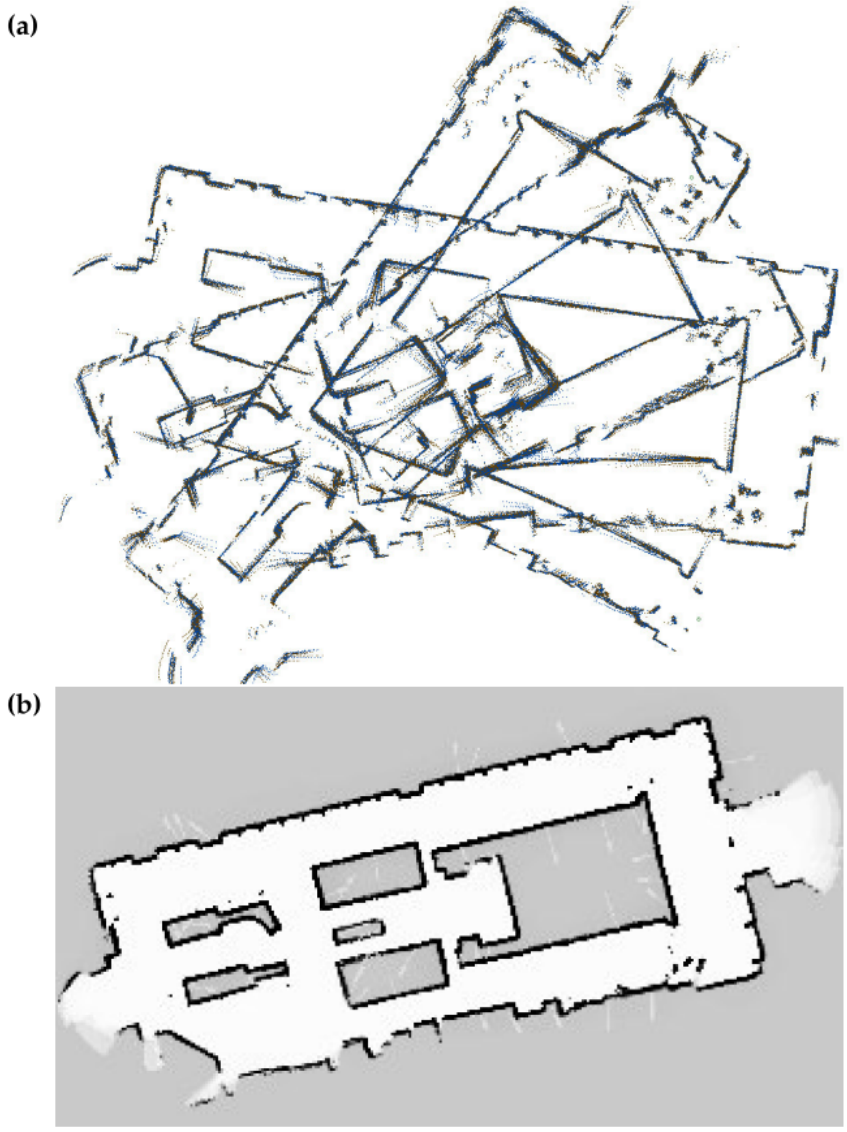


Figure 2.6: (a)Shows the result of raw sensor and odometry data. Clearly visible orientation and matching errors, because measurement data is not matched. (b)shows a post processed occupancy grid map where odometry data got corrected. Black dark symbols occupied cells, white free cells and gray marks cells which have never been updated, therefor the state is unknown. The figure has been taken from [2].

2.2.1 Occupancy Grid Maps

Originally introduced by [20], occupancy grid maps are a discrete representation of the world. They represent the posterior $p(m|z_{1:t},x_{1:t}) = \prod_i^n p(m_i|z_{1:t},x_{1:t})$ of the map m (consisting of small, discrete cells m_i) based on measurements $z_{1:t}$ and pose (aka. a followed path) $x_{1:t}$ ($1:t$ denotes the time from origin to now).

Essentially an "Occupancy Grid Map" is a binary Bayes filter. It distinguishes between 2 (3) states:

- **occupied**: the cell represents e.g. a solid wall
- **free**: the cell represents e.g. a drivable surface
- (**unknown**): the cell has never been updated

These 2 (3) states can be maintained using "Log Odd Ratios" as shown in equation 2.17. Commonly this is used to find associations between two variables, within this case occupied/free and actual measurement value. Every time a sensor senses over a cell (logarithmic occupancy l_t), it gets updates with the sensors belief if the cell is free or occupied ($p(x|z_t)$). If there is an a priori information about the cell, e.g. from a blue print, $p(x)$ can be used. l_t ranges from negative to positive values. Depending on the reliability of the used sensor one has tho define thresholds to distinguish between occupied and free cells.

$$l_t = l_{t-1} + \frac{log(p(x|z_t))}{(1-p(x|z_t))} - \frac{log(p(x))}{(1-p(x))} \tag{2.17}$$

Based on the fact that every cell has got a finite size, an occupancy grid map becomes a discrete assumption of the world. Depending on the required accuracy of the map (indirect proportional to the cells size) and the area to cover, those maps can lead to a very high memory consumption and a high computation when updating with sensor data.

2.2.2 inverse sensor model

The inverse sensor model approach, see [2], uses the information given by the sensor and its model. The obtained measurement value is used to determine which cell within the sensor range will be free or occupied. This description will be added to the map where every cell gets updated. Figure 2.7 shows cells occupied or free per one measurement update (based on the sensor model). Figure 2.8 shows how maps are build from multiple inverse measurements from sensors.

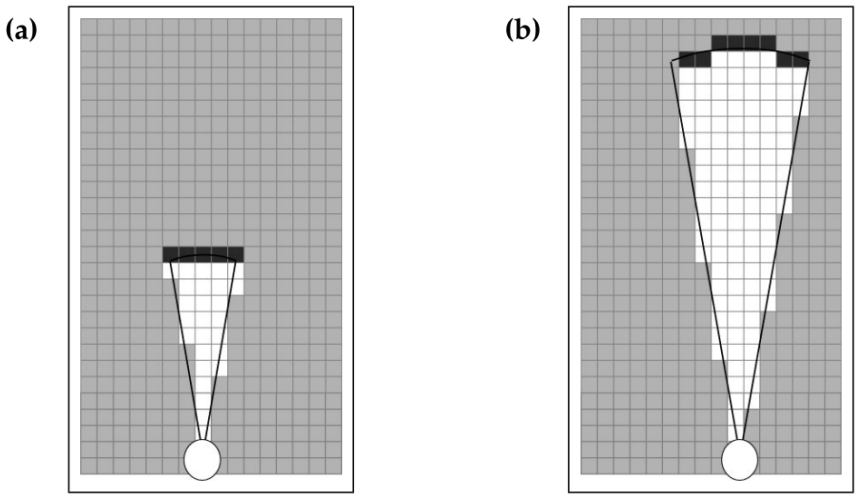


Figure 2.7: Two examples of sensor updates. The number of cells to update differs between (a) and (b) due to the different sensed distance. White cells symbol free, dark black occupied and gray is unknown. The figure has been taken from [2].

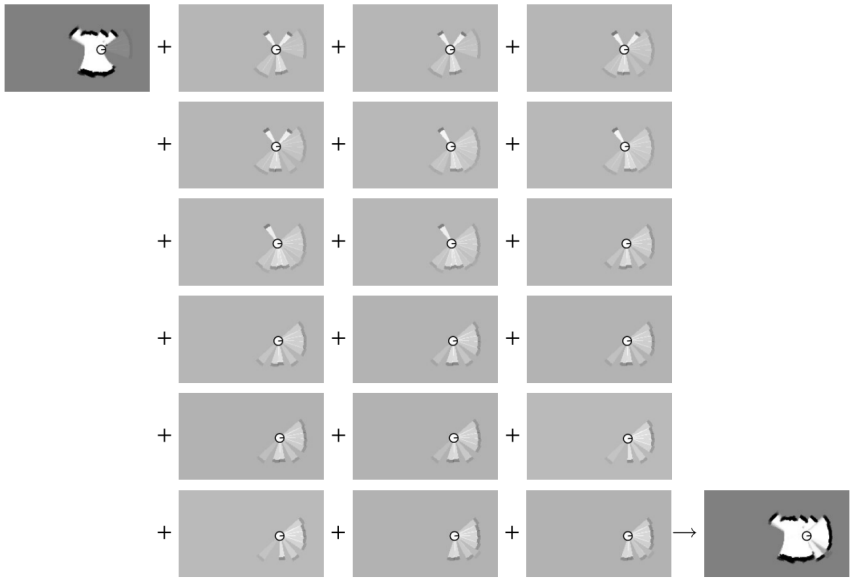


Figure 2.8: The left upper shows the initial map, the right lower the resulting. In between single measurement cones have been used to show the process of inverse sensor models. A single cone like in Figure 2.7 is used to map the whole measurement area for the used ultrasonic sensor to the map. The figure has been taken from [2].

2.2.3 forward sensor model

[21] discusses a sensor model which describes the measured value based on obstacles. Ray casting is used to determine which cell inside the sensors coverage

is likely to cause this measurement result, compare Figure 2.9. The cells log odd ratio will be calculated, as given within equation 2.18.

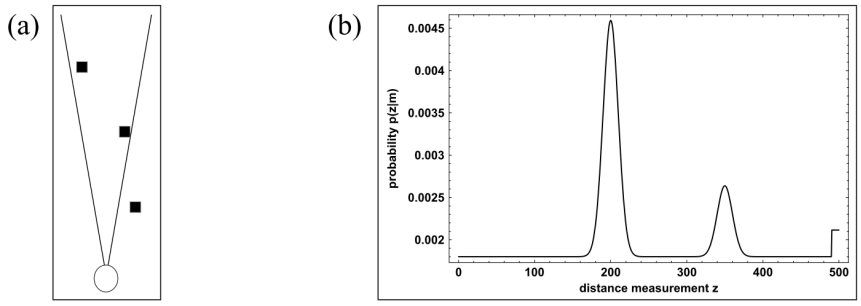


Figure 2.9: (a) shows a scene where the black squares mark obstacles. The outer left and the center are likely to cause a measurement, the outer right not. (b) shows the probability to obtain a measurement. The first peak is caused by the center square (which is highly likely to cause the reflection of e.g. a sound wave from ultrasonic sensors). The second peak corresponds to the left outer square. Both obstacles will be denoted with the p_{hit}. The right outer may causes a random measurement which gets modeled with p_{rand} Source: [21]

$$p_{c_t} = \begin{cases} p_{rand} & if c_{t,*} = 1 \\ (1 - p_{rand})(1 - p_{hit})^{K_t} & if c_{t,0} = 1 \\ (1 - p_{rand})(1 - p_{hit})^{k-1} p_{hit} & if c_{t,k} = 1 for k \geq 1 \end{cases} \tag{2.18}$$

p_{rand} and p_{hit} are obtained in the same way as for the beam model within Section 3.2.1.

2.3 Bayesian filters

H. Altinger NOTE: using section to give mathematical backgrounds

H. Altinger NOTE: maybe enhance by G:\ds-9_save\harald\Tu_Graz\DA\Literatur\ungelesen\various_PF_filters_DA.pdf

2.3.1 general introduction to Bayes filters

The following section will give a short introduction to Bayes filter in general. Unless otherwise this section is based on literature, see [2] chapter 2.4.

A very good summary of the filters task has been given in [22]: "Bayes filters probabilistically estimate a dynamic system's state from noisy observations". Bayes filter can give a probability about ones pose according to the history of

sensor measurements; if you tell them, what you have seen so fare (through sensor measurements), they can tell you how likely you are at a specific position (valid for any position within your world).
A very popular graphical visualization demonstrating this fact can be seen within Figure 2.10. The robot can measure its traveled distance and sens a Door, but can not difference between those 3 doors. (a) shows the initial belief, where every position is equally possible. (b)Senses one door, and increases $bel(x)$ at all 3 doors positions. (c)The robot has continued its path (it predicts the position by its odometry updates, moves and decreases the $bel(x)$ to model uncertainties.) and senses another door. (d)The situation only fits on location, therefor the $bel(x)$ increases at the second door (which is the most likely position) Denote, $p(z|x)$ it the same for all 3 doors, but the $bel(x)$ represents the fact that two doors are closer than a third and the predicted odometry only fits to one position observations. (e)Predicting its position based on odometry slightly decreases its position belief (due to required modeled uncertainties), but there is still a high likely position. The next measurement update will occur when the robots senses the 3^{rd} door, which will be similar to (d). The figure has been taken from [2].

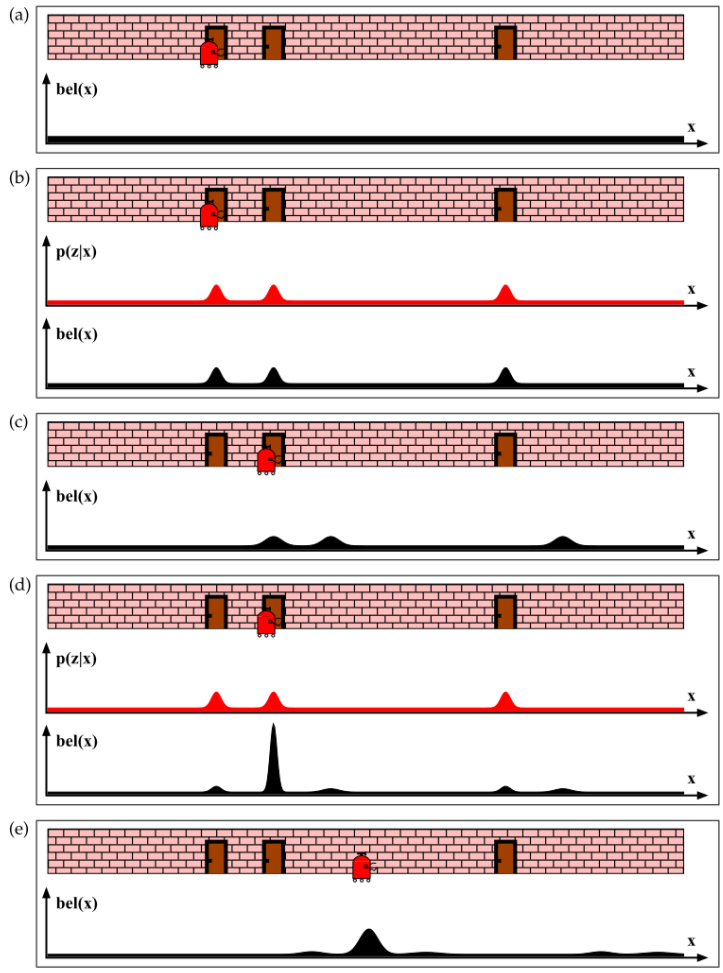


Figure 2.10: A simple illustration of Bayes process. A robot is traveling along a path an senses a door. This event is used to update the robots position belief.

The Bayesian filter is the most general approach to calculate a belief ($bel_t = x_t$) based on measurement z_t and control data[1] u_t. A Bayesian filter processes the following stages:

- prediction or control update[2], by processing u_t
- measurement update[3], by processing z_t
- normalization of the belief to 1 using η

1 control data can be anything that changes (or represents a change in) the environment, e.g. an odometer measures the robots change of position when moving through the world
2 control update can be a drive command, e.g. 1 m straight line
3 measurement update can be a sensor reading

Bayes filter relies on the Markov assumption, that the current measurement only depends on the actual state (e.g. a position) and the past event (x_{t-1}). Older events do not add any kind of information.

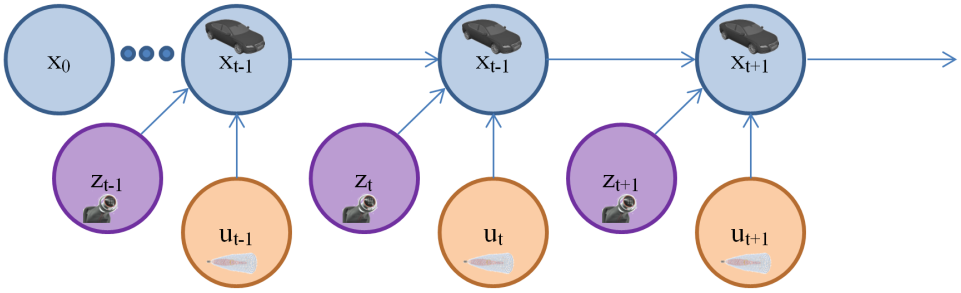


Figure 2.11: A general Bayes network: the actual belief x_t consists of the previous belief x_{t-1}, the actual control input u_t and the latest measurement data z_t. Every belief calculation is recursive. Therefor x_t will be an input for x_{t+1}. To start the network an initial belief x_0 is required (due to the recursive update rule).

The belief for state x_t at time t defines the probability that all past measurements $z_{1:t}$ and controls $u_{1:t}$ are likely at state x_t, refer to equation 2.19. This belief can only be calculated after handling the data (z_t,u_t).

$$bel(x_t) = p(x_t|z_{1:t},u_{1:t}) \tag{2.19}$$

It is more practicaly to calculate the belief befor the last data (z_t,u_t) is available (or can be processed). Equation 2.20 expresses $\overline{bel}$ this.

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1},u_{1:t}) \tag{2.20}$$

Literature knows equation 2.20 as "prediction", due to the fact that it predicts the belief by processing the past measurement data. Calculating $bel(x_t)$ from $\overline{bel}(x_t)$ is known a "correction" or "measurement update". Knowing probabilities for every random variable, conditional probability can be calculated using Bayes theorem. When considering [2] this calculation leads to equation 2.22.

$$\overline{bel}(x_t) = \int p(x_t|u_t,x_{t-1},)bel(x_{t-1})dx_{t-1} \tag{2.21}$$

$$bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t) \tag{2.22}$$

Commonly known representatives of Bayes filters are Kalman or particle filters, see the following sections.

2.3.2 Kalman filter

This section gives a short introduction to Kalman filter in general. It is based on literature, see [2] chapter 3.2 for a more detailed breakdown.

The Kalman filter, as introduced by [23], is an optimal filter. It is designed to work within environments with normal distributed noise, so Gaussian functions can be used to model. It assumes a continues state with linear system dynamics. The Kalman filter is often refers as a linear optimal filter. Its belief describes the probability for a state change. As being linearized and optimized, the filter is computational efficient ($O(k^{2.4})$) k is z_t dimension.
In many applications the Kalman filter is used to merge different measurement data. The belief consists of a state variable ($\vec{x}$) and a covariance matrix (Σ) to represent the probabilistic dependencies between the data. Equation 2.23 shows the linear system equation the Kalman filter is based on. A_t is the state change transition matrix, which describes the state change based on the state variables. B_t is used to handle the influence of the control data u_t, e.g. odometry represents controlled movement through the world. ε_t is used to model the error within state change. Equation 2.24 is the multivariate[1] form of Kalman filters state transition probability. Literature refers to R_t as the covariance in case of $p(x_t|u_t,x_{t-1})$ and Q_t in case of $p(z_t|x_t)$.

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \tag{2.23}$$

$$p(\vec{x}_t|\vec{u}_t,\vec{x}_{t-1}) = det(\frac{1}{\sqrt{2\pi R_t}})e^{-\frac{1}{2}(\vec{x}_t - A_t\vec{x}_{t-1} - B_t\vec{u}_t)^T R_t^{-1}(\vec{x}_t - A_t\vec{x}_{t-1} - B_t\vec{u}_t)} \tag{2.24}$$

Like other Bayes filters the Kalman filter uses a prediction and a correction of measurement data. Equation 2.25 shows the linear system equation of the update step. C_t describes the transition between state and measurement, δ_t the measurement errors. Equation 2.26 describes the multivariate measurement probability for the update step.

$$z_t = C_t x_t + \delta_t \tag{2.25}$$

1 the state variable $\vec{x}$ consists of more than on dimension

$$p(\vec{z}_t|\vec{x}_t) = \vec{z}_t = det(\frac{1}{\sqrt{(2\pi Q_t)}})e^{-\frac{1}{2}(\vec{z}_t - C_t\vec{x}_t)^T Q_t^{-1}(\vec{z}_t - C_t\vec{x}_t)} \tag{2.26}$$

The Kalman filter requires an initial state, which can be expressed by equation 2.27, where μ_t is the mean, Σ_t is the covariance.

$$bel(x_0) = p(x_0) \qquad = det(\frac{1}{\sqrt{2\pi\Sigma_0}}e^{-\frac{1}{2}(x_0-\mu_0)^T\Sigma_0^-1(x_0-\mu_0)}) \tag{2.27}$$

The whole Kalman filter process, for a simple 1D scenario, is visualized within Figure 2.12. Moving along a straight line the initial position belief is shown in (a). Due to a real system, which is amused to be influenced by normal distributed noise (system and measurement), the curve has got the shape bell, in contrast to the plane line of an ideal system. After the first measurement update (bold line in (b); μ and σ cause the width and height of the update, the peaks position differs slightly from the initial position at (a)). The Kalman filter combines those two distributions at (c) which reduces the uncertainty. Moving to the right (d) is modeled by the prediction and increases the uncertainty. A measurement update (bold (e)) can correct the position belief to (f). Comparing (a) and (f) the uncertainty is significant smaller.

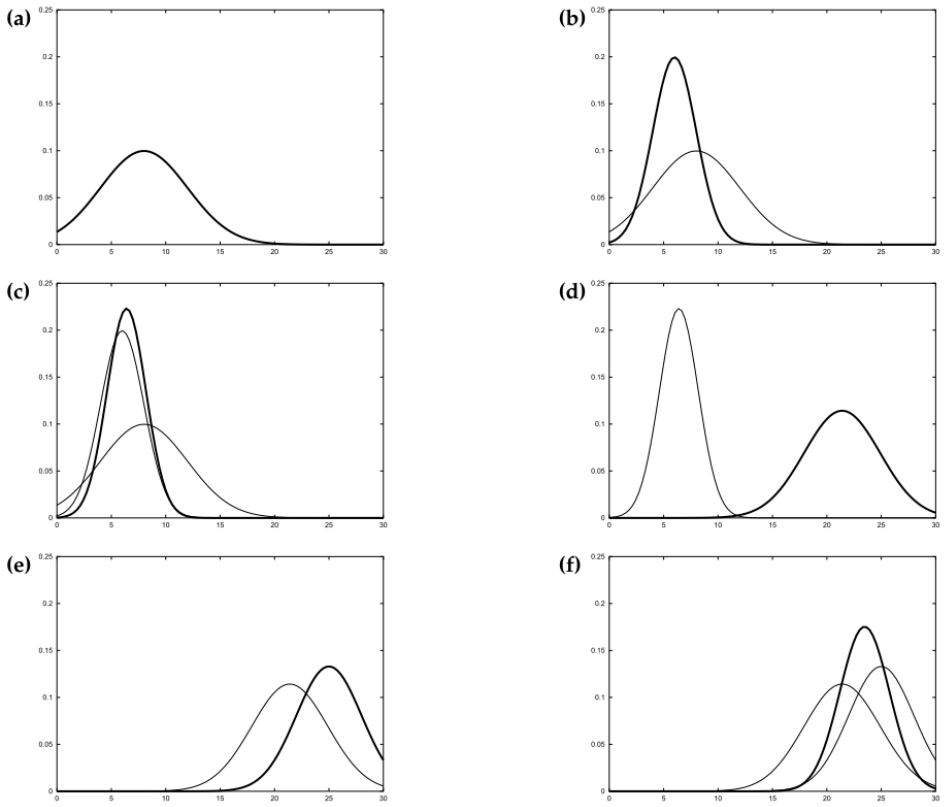


Figure 2.12: "Illustration of Kalman filters: (a) initial belief, (b) a measurement (in bold) with the associated uncertainty, (c) belief after integrating the measurement into the belief using the Kalman filter algorithm, (d) belief after motion to the right (which introduces uncertainty), (e) a new measurement with associated uncertainty, and (f) the resulting belief." The captions text and figure has been taken from [2] as granted on the authors website

2.3.3 particle filter

The following section gives a short introduction to the topic of particle filter in general. It is based on [2] and explicit referenced literature.

"Any realistic model of a real-world phenomena must take into account the possibility of randomness.", quoted from Sheldon M. Ross.

In some cases the assumed linearity, when using a Kalman filter, is to far away from reality and can cause divergence. As shown by [24] even an Exended Kalman Filter (EKF) fails if the world is highly nonlinear (or the modeled linearization contains too much assumptions) and contains normal (Gaussian) distributed noise, which is the case for a more complex systems.

A particle filter is a nonlinear and nonparametric Bayes filter and can address

Markov chains. The posterior $bel(x_t)$ is represented by a random number of state samples (drawn from this posterior) and a normalized weight w_i ($\sum^i w_i = 1$), representing the importance of the linked sample. The (exact) exponential function would describe such a distribution the best, but would be parametric. Therefor an approximation via drawn samples is used. This represents much more types of distributions (compared to exponential closed form), due to its nonparametric nature.
Particle filters are limited by the available computation power. Within a higher state space (aka. a higher dimension), an exponential higher number of particles is required, see [25]. Thus successful applications are limited to nonlinear, nonparametric low dimensional problems, like the localization problem. (e.g. 4 dimension state: Cartesian coordinates + heading: $< x,y,z,\varphi >$)

A very common illustration of this process is shown within Figure 2.14, which is the same problem domain as described within 2.10.

The particle filter use M numbers of state hypothesis, see equation 2.28. But this is also the cause of its biggest limitation. A sufficient number of samples per dimension is required, which can cause high computation $O(N^2)$ (N beeing the number of particles), see [26]. Therefor a particle filter is practical not applicable for high dimensional spaces.

$$X_t = x_t^1, x_t^2, \cdots, x_t^M \tag{2.28}$$

In general, a particle filter follows a procedure very similar to the general Bayes ansatz.

1. sample all hypothesis, see equation 2.29
2. predict the new state based on control input u_t
3. weight all samples (see equation 2.30), based on measurement update z_t
4. add samples to state space
5. draw a number of samples (probability is proportional to their weight)
6. resample (if required) and add good hypothesis to the state space, replace all others with new ones

$$x_t^m \sim p(x_t|u_t, x_{t-1}^m) \tag{2.29}$$

$$w_t^m = p(z_t|x_t^m) \tag{2.30}$$

As any other Bayes filter, the particle filter calculates its $bel(x_t)$ based on its $bel(x_{t-1})$. As shown in [2] and [27] the approximation through the samples convergences against the real state for an infinite number of samples. The most important step is the resampling. It changes the distribution from proportional to $bel(x_t)$ to $\eta p(z_t|x_t^m)\overline{bel}(x_t)$. Any good hypothesis will be copied and pursued, others will be replaced by new ones. This step is most important to prevent degeneration of the sample set.

Particle filter can solve global localization problems[1]. [28] introduced this as the "kidnapped robot problem"[2].

As stated within [22] the particle filter is a very accurate and robust approach to localization problems. In addition it has got other benefits (simple implementation, Sensor variety, etc.) too. Therefor a particle filter is a good choice to estimate a robots position. Two types of particle filters will be explained within the next sections.

representation of localization

The position $bel(x)$ of a robot can be expressed in different ways. This section briefly introduces 3 common representation based on literature, see [2].

The **closed form representation** is used within the Markov localization algorithm. For every position the initial belief $bel(x)$ is uniform and represented using $p(x_t|z_t,u_t)$. Figure 2.10 shows the propagation of the belief when the robot moves. The result is a closed form containing an explicit value for the belief, e.g. the Kalman filter uses such a representation. This can only be done for specialized cases with some assumptions.

The **grid form representation** divides the possible positions into discrete divisions, called grids. As seen in Figure 2.13 the robots position belief can be analyzed using histograms above all subdivisions. This method can be used in more general causes than the closed form representation and generates less computational costs. But the achievable accuracy strongly depends on the granularity of the grids which is related to the amount of memory required.

The **sample form representation** uses nonparametric samples. Those are sampled uniform at possible positions. If the robot moves the samples will be assigned importance factors if the sensed data fits the samples. Based on this good samples will stay, low weights samples will be redrawn. The height of the

1 The robot is placed elsewhere in the world. There is no information about its a priori position or any external reference. The robot needs to localize its self.

2 The robots gets kidnapped and placed somewere else.

importance weight and the density of samples represents the position belief $bel(x)$. Figure 2.14 shows such a process. This representation is nonparametric, but its computation requirements strongly depend on the number of samples.

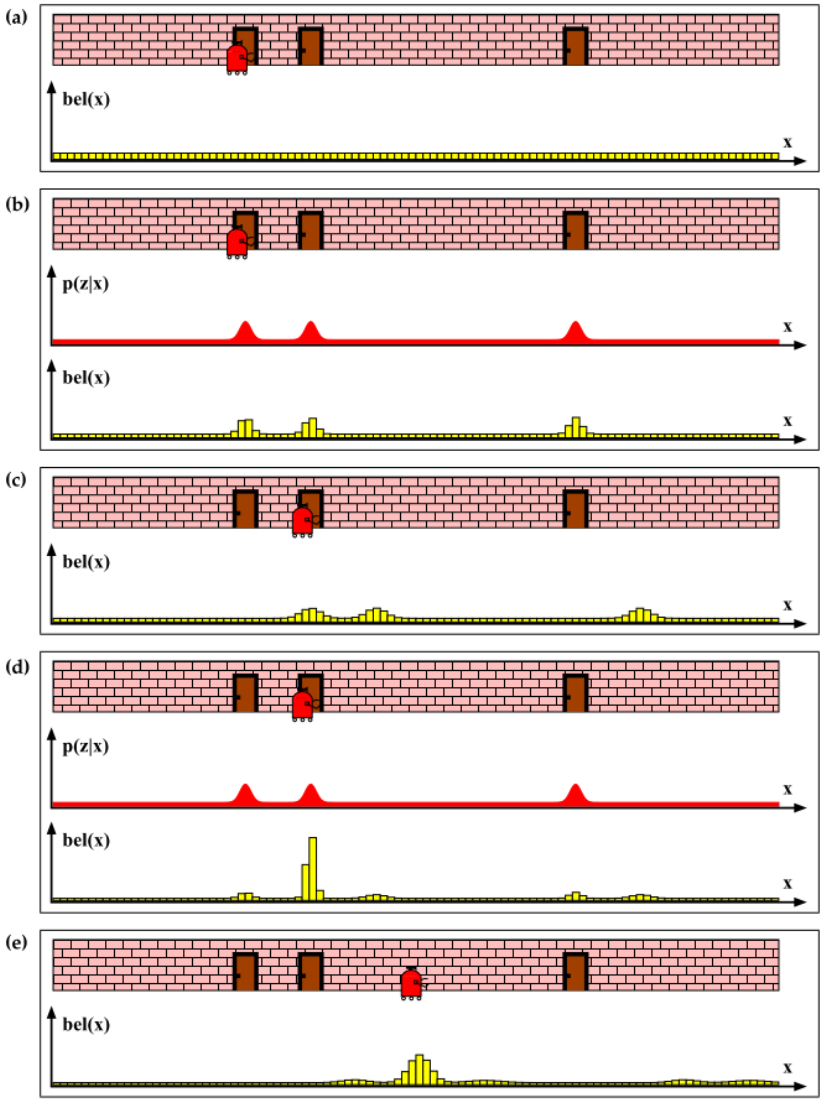


Figure 2.13: Basically the process is the same as described within Figure 2.10. The figure has been taken from [2].

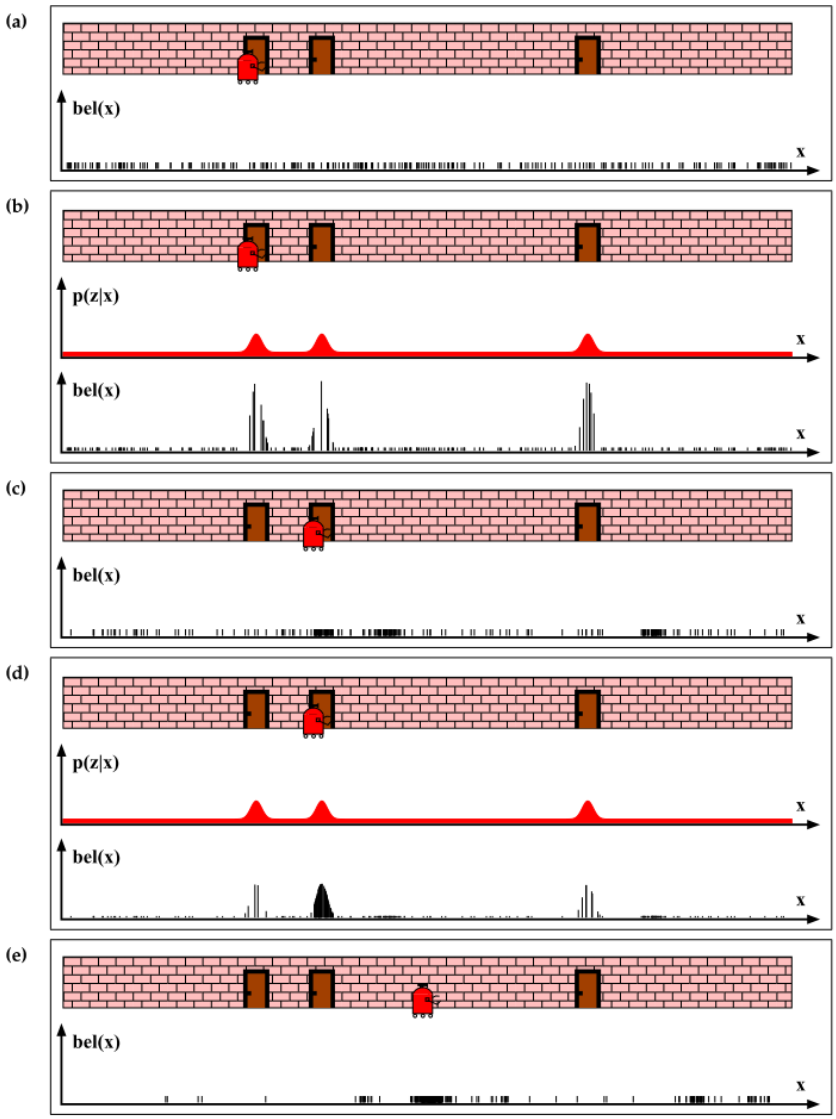


Figure 2.14: Basically the process is the same as described within Figure 2.10. The figure has been taken from [2].

Monte Carlo Particle Filter

Monte Carlo Localization (MCL) has been introduced by [1] and according to [28] it is a simple, but in practice, a very useful filter. In comparison to grid based filters it requires by far less memory, it is more accurate than Markov based (which the MCL is based on) localization approaches and it is able to represent multi-modal distributions (in contrast to Kalman filters).

As beeing a sampling / importance resampling methode, the key idea is to represent the posterior $bel(l)$, which is the robots belief to be at position l, by N weighted samples. If the state can be represented with $< x,y,\varphi >$ the MCL

extend this with a weight w to $<< x,y,\varphi >, w >$ ($\sum_{i=1}^{N} w_i = 1$). At every processing step a (new) set of samples is distributed according to motion models (compare Chapter 3.2.2) to predict the possible robots location. If nothing gets corrected (e.g. using measurement data to rank and correct those samples), Figure 2.14 shows the propagation of the sample distribution. The area this distribution covers increases because of uncertainties the motion model has to map. After every prediction step a correction step (called measurement update) follows.

Equation 2.31 describes the likelihood of sample n at time t, which relates to the $bel(l)$ of the sample. $p(x_t|z_{1:t},u_{1:t})$ is the probability function, α is the normalization factor, to ensure $\sum_{i=1}^{N} p_i = 1$. $z_{1:t}$ represents the measurements from 1 to actual time t, and $u_{1:t}$ the control update.

$$x_t^n = \alpha \cdot p(x_t|z_{1:t},u_{1:t}) \tag{2.31}$$

Equation 2.32 is used during the measurement update step. w_t^n represents the weight a particle is assigned. p is again the probability function taking in account the latest (at time t) measured data z_t and the particle n's actual position x_t^n (at time t).

$$w_t^n = p(z_t|x_t^n) \tag{2.32}$$

Altogether these basics are very similar to particle filters in general, see Section 2.3.3. Figure 2.14 shows a general procedure of a particle filter. The MCL difference is to implement equations 2.31 (aka. 2.29) and 2.32 (aka. 2.30) with models approximating the reality. The chosen motion model (refer Section 3.2.2, equation 3.27) will be extended by a random distribution according to equation 2.33, see [2], which results in equation 2.34.

Equation 2.33 represents a normal distribution with μ as the mean and σ^2 as the variance.

$$sample(\mu,\sigma) = \frac{1}{\sqrt{(2\pi\sigma^2)}} e^{-\frac{1}{2}\frac{\mu^2}{\sigma^2}} \tag{2.33}$$

Equation 2.34 extends the Constant Turn Rate and Velocity (CTRV) model with a normal distribution. As the velocity v and the rotation rate ω are

measurement the sample distribution uses them.

$$\vec{x}_{t+T} = \begin{pmatrix} x_{(t)} + \frac{sample(v,\sigma_v)}{sample(\omega,\sigma_\omega)} sin(\varphi_t + \omega T) - sin(\varphi) \\ y_{(t)} - \frac{sample(v,\sigma_v)}{sample(\omega,\sigma_\omega)} cos(\varphi_t + \omega T) - cos(\varphi_t) \\ \varphi_t + sample(\omega,\sigma_\omega)T \\ v \\ \omega \end{pmatrix}^T \tag{2.34}$$

The "beam rangefinder model" has been used as an measurement update model, refer to Section 3.2.1 and [2]. The parameters have been obtained with experiments described within Section 4.3.

As being a nonparametric filter the MCL can approximate any distribution and is not limited to linear systems like, e.g. the Kalman filter. The accuracy strongly depends on the number of samples. In general the higher the number the more a particle filter becomes an "optimal Bayes Filter". Therefor a working solution is to choose a very high (e.g. 200.000, see [2]) number of particles (see [29]) at the cost of long computation time. On the other side of computation, [30] demonstrated a successful localization with 50 particles. [31] derives equation 2.35 to determine the minimum number N of samples for a given border of accuracy (ϵ and δ) and number of bins b. This whole particle filter with adaptive number of particles is known as "Kullback-Leibler distance (KLD)". [2] demonstrates KLD particle filters number of samples varies for different kind of used sensors and changes over time.

$$N = \frac{b-1}{2\varepsilon}(1 - \frac{2}{9-(b-1)} + \sqrt{\frac{2}{9(b-1)}} z_{1-\delta})^3 \tag{2.35}$$

If the number of particles is to low, or the weight distribution decreases, degeneracy can occur. [32] demonstrates the usage of equation 2.38 to determine if the weight of all particles is degenerated. A low N_{eff} is an indicator for degeneracy.

$$N_{eff} = \frac{N_s}{1 + Var(\omega_k^{*i})} \tag{2.36}$$

$$\omega_k^{*i} = \frac{p(x_k^i | z_{1:k}}{q(x_k^i | x_{k-1}^i), z_k)} \tag{2.37}$$

Although equation 2.37 can not be calculated, equation 2.38 is used to approximate. N_s is the total number of particles, ω_k^i the normalized weight of particle i at time k.

$$N_{eff} = \frac{1}{\sum_{i=1}^{N_s} (\omega_k^i)^2} \tag{2.38}$$

If N_{eff} falls below a certain level countermeasurements are recommended, e.g. resampling, change number of particles, reinitialize the filter, etc.

MCL may also known as Sample Importance Resampling (SIR)- or bootstrapfilter or condensation algorithm or survival of the fittest.

An open source MCL implementation is the "Adaptive Monte Carlo Localization (AMCL)" package for Robot Operating System [89] (ROS). It implements the KLD algorithm and a map to determine the robots position. Another free implementation is known as "CMonteCarloLocalization2D" which is part of the SLAM package of Mobile Robot Programming Toolkit, see [88] (MRPT).

Some publications, e.g. [33], uses MCL with a very high number of particles ($> 3.5 \cdot 10^6$) as reference implementation to measure the performance of their approaches.

Rao Blackwellized Particle Filter

The Rao-Blackwell Theorem, as introduced by [34], defines $\delta_{rbf}(\theta)$ as an observable estimator of an unobservable variable θ. $\delta(\theta)$ will be the true estimator, and E the function of expectation, compare equation 2.39. A more detailed mathematical derivations can be found at [35].

$$E(\delta_{rbf}(\theta) - \theta)^2 \leq E(\delta(\theta) - \theta)^2 \tag{2.39}$$

As state in [36] and [2] the number of Markov states grow exponential with the dimension of the state space. Analog the computational complexity grows with the number of states for classical particle filters, e.g. MCL, because they are based on these Markov Models. In many cases MCL (or similar particle filters) require much more computation time than e.g. Kalman filters, see [37]. [38] uses Rao-Blackwellization and Kalman filters to increase the efficiency and accuracy achieved by a particle filter.

Within literature Rao-Blackwellized particle filters are used to solve the SLAM problem. [2] describes the "FastSLAM" algorithm. Every particle consists of its own movement (state and history) and map. [39] defines this combination between particle filter for simple (3 dimensions) state and a Kalman filter for higher state (n dimensions) with equation 2.40. $\vec{x}_{t+1}$ is the state-vector (index pf refers to particle filter, kf refers to Kalman filter). A is the state transition matrix, B the control matrix, u_t the control update.

$$\begin{pmatrix} \vec{x}^{pf}_{t+1} \\ \vec{x}^{kf}_{t+1} \end{pmatrix} = \begin{pmatrix} I & A_{pf} \\ 0 & A_{kf} \end{pmatrix} \begin{pmatrix} x_{pf} \\ x_{kf} \end{pmatrix} + \begin{pmatrix} B^u_{pf} \\ B^u_{kf} \end{pmatrix} u_t + \begin{pmatrix} B^f_{pf} \\ B^f_{kf} \end{pmatrix} f_t \tag{2.40}$$

SLAM posterior $p(y_{1:t}|z_{1:t},u_{1:t})$ can be calculated within factorized form, see equation 2.41. x_t is the location belief, z_t the measurement update, u_t the control update, y_t the particles, m_n the map of a single particle and c_t the identity, see [2]. The subindex $_t$ denotes the time stemp.

$$p(y_{1:t}|z_{1:t},u_{1:t},c_{1:t}) = p(x_{1:t}|z_{1:t},u_{1:t},c_{1:t}) \prod_{n=1}^{N} p(m_n|x_{1:t},z_{1:t},c_{1:t}) \tag{2.41}$$

[38] gives a more detailed mathematical derivation and a generic algorithm to Rao Blackwellized Particle Filter (RBPF). The procedure itself is rather similar to other particle filters. First the particles are sampled according to the chosen distribution, the particles prediction step is performed, followed by a weight calculation and normalization. To prepare for the next round low weighted samples will be suppressed (high weights will be multiply) and a Markov transition will be performed.
In addition they can ensure for bounded weights w_t and their selection schemes (resample step) the RBPF convergences independent the total number of particles.

Compared to MCL the RBPF is much more complex to implement. Its main target is to address the SLAM problem and building maps.

An open source MCL implementation is the "gmapping" package for ROS. It implements the RBPF algorithm and a map to determine the robots position. A detailed description is given at [40] and [41].

2.3.4 Comparison and decision advices

Due to their various qualities there is no "Golden Bayes Filter" to solve a localization problem. Depending on the available sensors, sourroundings, updaterates, computational power, etc. one kind of filter has to be chosen. [22] compares the most common filters and can give hints which one to choose. Table 2.1 gives a short overview.

In general terms, the Kalman filter is the most efficient in terms of computational power, but it requires accurate sensors with a rather high update rate. The nonparametric particle filter uses a discrete belief, its accuracy strongly depends on available computation power (aca. the maximum number of particles), but it is a rather robust approach that can deal with uncertainties and

less accurate sensors.
Both filter techniques can be used to merge sensor information and to enhance a position calculation compared to single sensor systems.

	Kalman	*Particle*
Belief	Unimodal	Discrete
Accuracy	+	+
Robustness	0	+
Sensor variety	-	+
Efficiency	+	0
Implementation	0	+

Table 2.1: Comparison between Kalman and particle filter. 0 represents neutral, + good and - weak. Source: [22] which includes the additional methods Grid, Topology and Multi-hypothesis tracking

[37] compares Kalman filters against particle filters and demonstrates a better accuracy for particle filters in case of nonlinear and non-gaussian environments, but at the cost of higher computation time for particle filters.
Within [42] and [43] various localization approaches are compared. They can show that simple Kalman filters are outperformed by MCL in terms of accuracy and robustness. Combining a particle filter with adaptive particle set sizes and resetting approaches, the high computation cumbersome disadvantage can be reduced.

[44] demonstrates a new approach to address the particle filters performance issue with higher dimensional state spaces. Rather similar to RBPF they combine multiple filters. A set of connected particle filters which acts on subsets of the computation space is used, which avoids the usual exponential grow with the dimension of the state space.
[45] compares different resampling methods for particle filters. Multinomial is compared against residual and stratifed. The authors give a good overview to the methods and mathematical boundaries. They also prove the variance of the particle estimator almost can be independent of the previous step.

2.4 furthers localization approaches from literature

The following section will give a brief overview about further localization approaches not considered within this thesis.

Some of these methods are improvements to well known techniques, e.g. the EKF and Unscented Kalman Filter (UKF) are based on the Kalman Filter. Other methods are already commercial applicable like the GPS system or a

trivial geometric ansatz like the triangulation.

H. Altinger NOTE: using section to give an overview about other state of the art localization approaches and links to implementation

2.4.1 Extended Kalman filter localization

The EKF is an extended version of the well known Kalman filter to address nonlinear problems. In contrast to the Kalman filter (as described within Section 2.3.2) the EKF uses linearization with results in the following differing equation:

$$A_t\mu_{t-1} + B_tU_t \longrightarrow g(u_t, \mu_{t-1}) \tag{2.42}$$

$$C_t\overline{\mu}_t \longrightarrow h(\overline{\mu}_t) \tag{2.43}$$

The two functions $g(u_t, \mu_{t-1})$ and $h(\overline{\mu}_t)$ are linearized functions to approximate the nonlinear world. On the left hand side of equations 2.42 (state prediction) and 2.43 (measurement update prediction) the classical Kalman filter is shown. On the right hand side the EKFs equivalent is presented. Equation 2.44 shows the whole EKF calculation which is rather similar (expect for the noted differences within equations 2.42 and 2.43; the Matrices A_t and C_t are replaced) to the classical Kalman filter as listed within equation 3.40.

$$\begin{aligned}
\overline{\mu}_t &= g(u_t,\mu_{t-1}) \\
\overline{\Sigma}_t &= G_t\Sigma_{t-1}G_t^T + R_t \\
K_t &= \overline{\Sigma}_t H_t^T (H_t\overline{\Sigma}_t H_t^T + Q_t)^{-1} \\
\mu_t &= \overline{\mu_t} + K_t(z_t - h(\overline{\mu_t})) \\
\Sigma_t &= (I - K_tH_t)\overline{\Sigma}_t
\end{aligned} \tag{2.44}$$

Due to the fact that the EKF uses linearization through Taylor approximation, it can not be used with any kind of nonlinear problem. Depending on the degree of uncertainty and the degree of local nonlinearity the quality of the approximation is better or worse. It is rather hard to find good suiting functions for g and h, but the computational complexity stays rather low ([2] gives them with $O(k^{2.4} + n^2)$, k as the dimension of the measurement vector.). Compared to the linear Kalman filter, the EKF is not an optimal estimator. If its process is modeled in an inaccurate way or the initial estimated state is wrong, the filter can diverge rather fast. [46] gives a detailed description of this divergence problem due to too optimistic estimations about the real uncertainty

in position heading.

A detailed mathematical derivation can be found within [2] and [47]. For performance analysis between the Kalman filter and EKF refer to [48].

2.4.2 Unscented Kalman filter localization

The UKF has been introduced as an improvement to the EKF by [33].

It is aimed to be more robust than EKF (because it is not using linearization), but less computational complex than a Particle Filter (PF).

The basic idea is rather similar the EKFs approach. Find an approximation to nonlinear functions h and f but still satisfy Bayesian transitions. A group of sample points with known mean and covariance will be transformed by a nonlinear transformation. The resulting distribution needs to be approximated. Monte Carlo methods, e.g. MCL, will draw samples randomly. The unscented transformation will use a deterministic algorithm. The source of noise is not only limited to Gaussian. Figure 2.15 shows the basic architecture (and process) of the UKF.

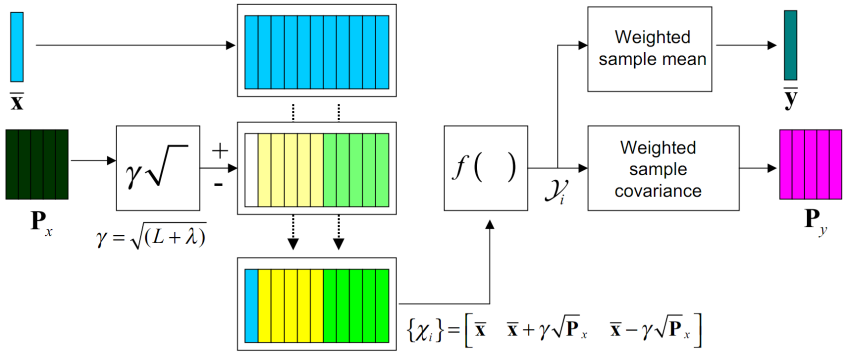


Figure 2.15: Known distribution mean $\overline{x}$ and covariance P_x will be scaled $y\sqrt{(}L+\lambda)$ and merge to sigma vectors X_i. The time and measurement update will be used calculate the weight posterior distribution $\overline{y}$ and P_y ; Source [49]-Chapter 7

A detailed mathematical derivation can be found within [33] and [49]. For performance analysis between the Kalman filter and UKF refer to [48].

2.4.3 Triangulation

A rather simple approach is triangulation. With a known pose and distance measurements one can determine the position. Figure 2.16 visualizes these geometric considerations. Depending on the accuracy of the measurement system the position can be determined with a very high precision.

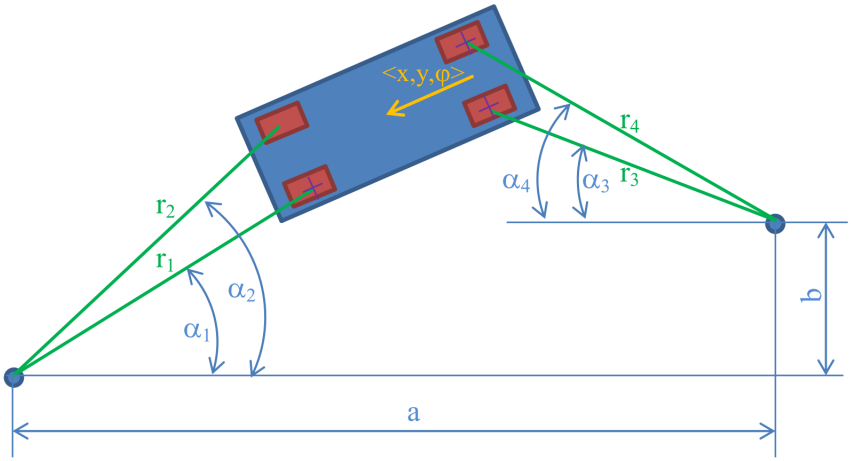


Figure 2.16: With two known measurement bases (a and b), measured distances (r_i) and a bearings (α_i) the position. With at least 2 measurements a Cartesian 2D position (x,y) can be determined, with a third measurement a Position $< x,y,\varphi >$ with orientation can be obtained.

2.4.4 Satellite based Navigation

The most popular representative, the Navigational Satellite Timing and Ranging - Global Positioning System (NAVSTAR GPS)[1] has been developed by the United States Department of Defense and became fully operational around 1990. It uses 24 satellites with known orbits and positions to determine the position of a receiver back on earth. Every satellite transmits a time signal and its position. When receiving at least 4 satellites the receiver is able to determine its position through triangulation. Figure 2.17 demonstrates this basic principle. For a general triangulation at least 3 distances are required. GPS determines distances based on the time difference when the beacon signal arrives at the receiver. Therefor 3 distances can be calculated out from 4 received timestamps, see [50].
Other satellite based global navigations systems are GALILEO[2], GLONASS[3] and COMPASS[4]

1 commonly known as GPS
2 developed by the European Union, operational around 2014
3 engl. global navigation system, developed and operated by the Russian military, became fully operational in 1996
4 developed and operated by the Peoples Republic of China, became fully operational in 2020

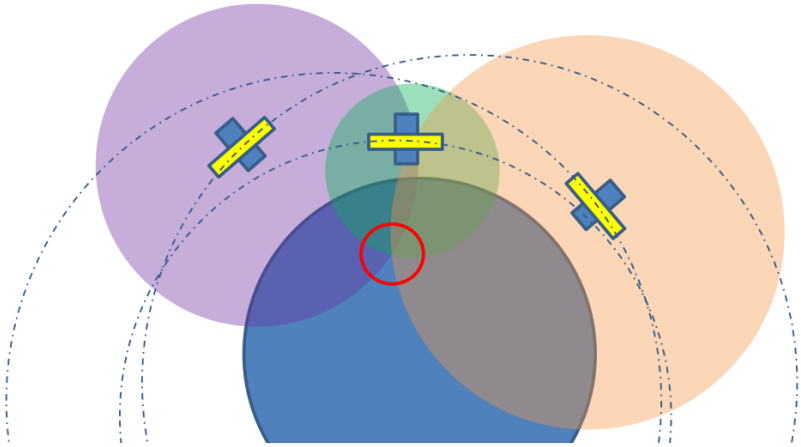


Figure 2.17: Every satellite transmits a time signal which the receiver can use to determine the distance to the satellite. Every satellite can be seen as a sphere (purple,orange,green) with the measured distance as the radius. The receivers position on earth is at the intersection (red circle).

According to [51] the original GPS system could achieve an average accuracy around 100m. To enhance this, DGPS could be used to obtain a more precise position between 3 and 5m. Figure 2.18 explains the principle idea. A fixed base station with known coordinates monitors the received GPS signals. The difference between the known position and the actual measured can be used to derive correction parameters. Assuming the same error[1] sources at the mobile units site, the raw data can be corrected and the positions quality can be enhanced. [51] mentions the commercial Wide Area Augmentation System (WAAS) to reach an accuracy below 3m.

1 possible error sources: ionospheric disturbances, timing, the errors in the satellites orbit and the local weather conditions

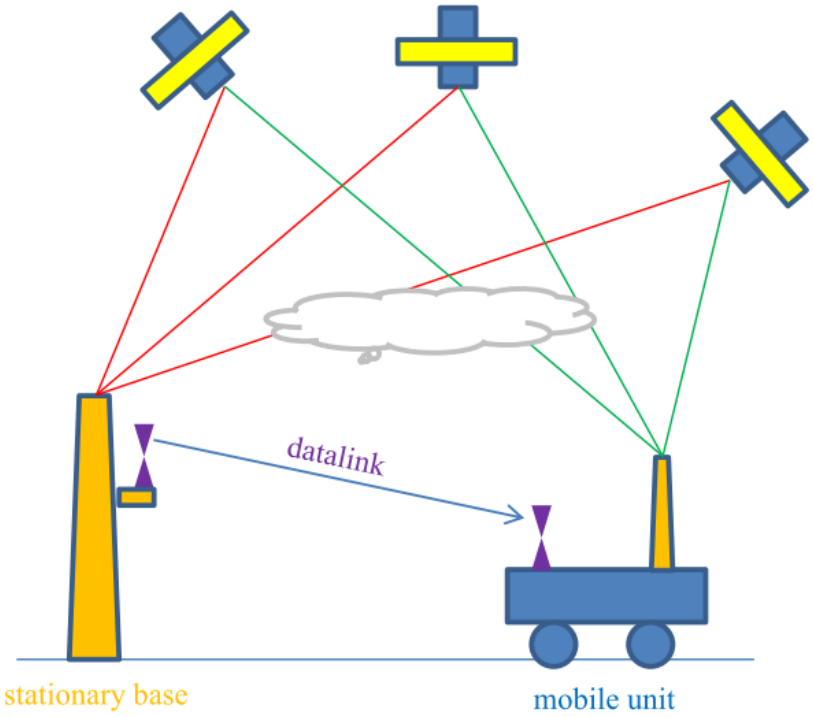


Figure 2.18: The fixed base (orange) and the mobile unit (blue) will receive the satellites signal (red and green) with the same interferences (e.g. caused by the gray cloud). With the known position of the base station a correction data can be calculated which will be transmitted to the mobile unit via some sort of data link (purple), e.g. cell phone network. The correction data can be used to enhance the mobile units position up to 3m accuracy, according to [50].

Further mathematical derivations and explanations can be found within [50] and [52].

2.5 literature discussion

H. Altinger NOTE: using section to introduce the general domain of autonomous cars

G. Steinbauer NOTE: comparison with others; what are the main differences, reason for differences

This section will start with a short overview to autonomous driving vehicles. There will be a short summary which sensors and methods were used to fulfill the various tasks. Afterwards seven papers will be discussed in details to give an overview to topics covered by other authors.

- [53]: **"Autonomous Ground Vehicles – Concepts and a Path to the Future"**

- [54]: **"Semi-autonomous virtual valet parking"**
- [55]: **"Stadtpilot: Driving Autonomously on Braunschweig's Inner Ring Road"**
- [56]: **"Digital Time of Flight Measurement for Ultrasonic Sensor"**
- [57]: **"Robuste Navigation autonomer mobiler Systeme"**
- [58]: **"New Resampling Algorithm for Particle Filter Localization for Mobile Robot with 3 Ultrasonic Sonar Sensor"**
- [30]: **"Sensor Resetting Localization for Poorly Modelled Mobile Robots"**

2.5.1 related work

B. Müller-Bessler NOTE: short intro to other autonomous cars

Within the last decades numerous autonomous cars where developed to solve various tasks. In cooperation Figure 2.19 and Table 2.2 give a short time-lined overview enriched with literature references.

Multiple of these cars uses high resolution, e.g. LIDAR, and absolute position sensors, e.g GPS to define their actual location. Some (compare [54],[59]) of them uses SLAM to build up a map to localize and one (compare [60]) uses a generic approach to determine its relative position within an expected environment.

In contrast to LIDAR and video ultrasonic sensors do not require high computational power and are known to be very robust (in terms of mechanical stress). Those sensors are used within automotive environments since the last two decades. Standard robotics literature did use ultrasonic sensors until late 1980 for in-building applications. In comparison to the listed cars this thesis focuses on ultrasonic sensors and relies on a mapped environment. Using enhanced sensor models and enriched maps a refined localization approach with ultrasonic sensors may lead to reduced computational power.

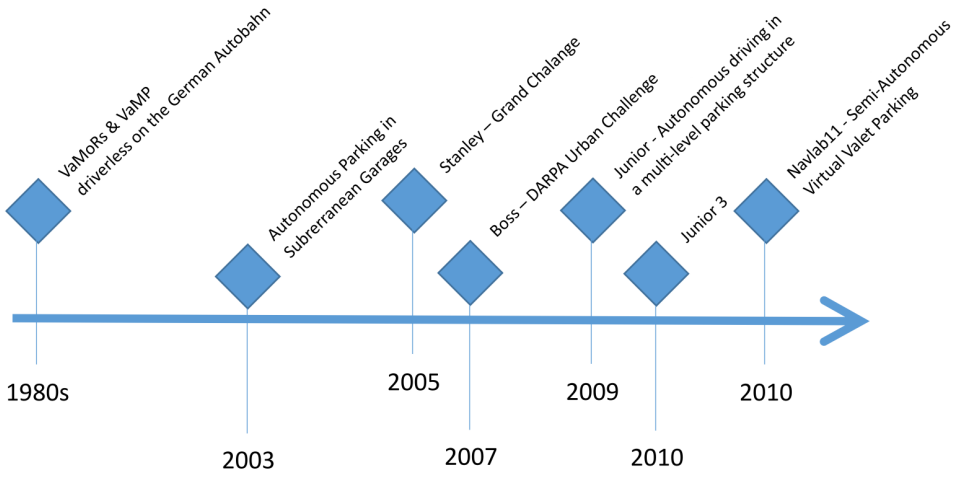


Figure 2.19: Short history of selected autonomous cars. For sensor configurations and literature references see Table 2.2

Robot	odometry	GPS	vision	LIDAR	sonar	terrein	literatur
VaMP	✓	✗	stereo	✗	✗	Autobahn	[60]
APiSG	IMU	✗	mono	2D & 3D	✗	parking garage	[61]
Stanley	IMU	✓	mono	2D & 3D	✗	offroad	[62]
Boss	IMU	✓	mono	2D & 3D	✗	urban	[14]
Navlab11	IMU	✗	mono	2D	✗	parking garage	[54]
Junior	IMU	✓	mono	2D & 3D	✗	parking garage multilevel	[59]
Junior3	IMU	✓	mono	2D	✗	parking spot	[63]

Table 2.2: sensor configuration of selected autonomous vehicles including literature references

Comparing Table 2.2 with 2.3 one can see a much simpler sensor set on the car used within this thesis ([54],[59],[63]). In contrast to other sonar based localization approach, this system needs to work in a full sized outdoor environment instead of an office environment ([20]). The car will operate on private ground and the scene will be static, so no fast object detection and tracing is required ([55]).

Robot	odometry	GPS	vision	LIDAR	sonar	terrein	literatur
Audi A7	✓	✗	✗	✗	✗	parking garage	this thesis

Table 2.3: sensor configuration of this thesis

2.5.2 general outlook on autonomous driving

"Autonomous Ground Vehicles – Concepts and a Path to the Future", see [53], gives an overview about autonomous vehicles. The authors give a short historical introduction, continue with actual and common sensors and conclude with an outlook to future development.

The authors identified the following common sensors for autonomous vehicles: Velodyne 3D Laser scanner, 2D LIDAR, mono and stereo cameras and maybe radar sensors to relative determine velocity of obstacles. Wheel encoders and Inertial Measurement System (INS) for ego motion, mostly (if operated in coverage areas) supported by GPS. In general sensordata need to be merged, therefor reference frames are required and calibration is mandatory. Sensorfusion extracts more information than from a single sensors raw data. Redundancy in sensors and actuators is required when performing the step from research to production. The main task to be solved with this sensors is environmental modeling, the (robotic) car needs to sense its world.

Obstacle (detection and) recognition is mostly done with a grid map. When updating the map with measurement data, a good ego-position is required. The authors identified that grid maps often not larger than 200 x 200 m and each cells side length from 0.1 to 0.5 m. Participating in road traffic requires obstacle detection, segmentation, e.g. cars (Ackerman steered cars move on clothoids paths) vs. pedestrians (can move in every direction) and tracking[1] to avoid collisions. Road lane recognition and tracking is required to work within a human suited environment. Lanes can be detected using vision or LIDAR sensors. Road network maps can be used to enhance localization.

Behavior control is commonly solved with hierarchical state machines. The task of navigation is more diverse. For parking related domain global navigation uses metric maps and pre calculates path to be followed. The planing algorithm operates on the map, independent from the sensors constructing the map. Main task is to maintain the map. Problems occur if map changes over time, maybe re planing is required.

Further development will be miniaturized sensors, e.g. LIDAR sensors. Position sensors will enhance their accuracy, e.g. Galileo should perform 10 times better than GPS or improve cost to performance ratio, e.g. mono vs. stereo cameras becoming cheaper.

Technologies from autonomous vehicles became available for production cars, e.g. lane keeping assists tracking the road lanes and keep the car within the boarders. Due to legal issues the driver still needs to be in the control loop.

1 a popular system is EKF

”Precise Farming” capable to perform autonomous mowing or lane keeping. Nowadays tractors only need a driver for obstacle avoidance or to guide a second vehicle.

Driver less cars are still legally vague, because who (supervising driver, manufacture, etc.) has to pay in case of an accidents. The state of Nevada[1] released a bill to allow autonomous cars for research on public roads, after Google lobby within their ”driverless car” project.

Rich enhanced maps will be available in near future, e.g. Geographic Information System (GIS). This can be used to guide vehicles and support them when detecting static obstacles. To conclude, the authors state Autonomous vehicles maybe on the road in around 10 - 15 years. The big challenge will be the interaction between computer and manual driven cars.

2.5.3 similar scenarios

The following section briefly reviews 4 publications from different authors dealing with indoor navigation where there is no GPS coverage. They are all related to the problem domain of indoor navigation and parking vehicles.

Within **”Semi-autonomous virtual valet parking”**, see [54], the authors present an use case scenario for an automatic parking system. They argue that disabled people, e.g. wheelchair users, can not park in every parking lot. A robotic car[2] equipped with Sick LMS200[3] sensors, odometry, INS and GPS is used.
The system is using a SLAM approach with an a priori teach in path and marked drop off points. The authors define obstacle detection and collision avoidance as a key feature to ensure a secure and reliable system.
Additionally they present a mobile interface to operator. If an obstacle is detected, the driver needs to be notified. Their aim is not force the driver to enable remote control, but just to ask if it is save to continue.

”Autonomous driving in a multi-level parking structure”, see [59], demonstrates a complete solution to autonomous driving within a multi level building. The authors mention that GPS can be used to localize oneself rather good, as long as there is clear sight to the sky. The described scenario is a large scaled, multi level, garage park which is indoor with no ability to use GPS. Even INS do have problems due to the large scales of the building. Figure 2.20 shows the difference between INS and the authors localization approach.

1 part of United states of America
2 Navlab 11
3 manual see [64]

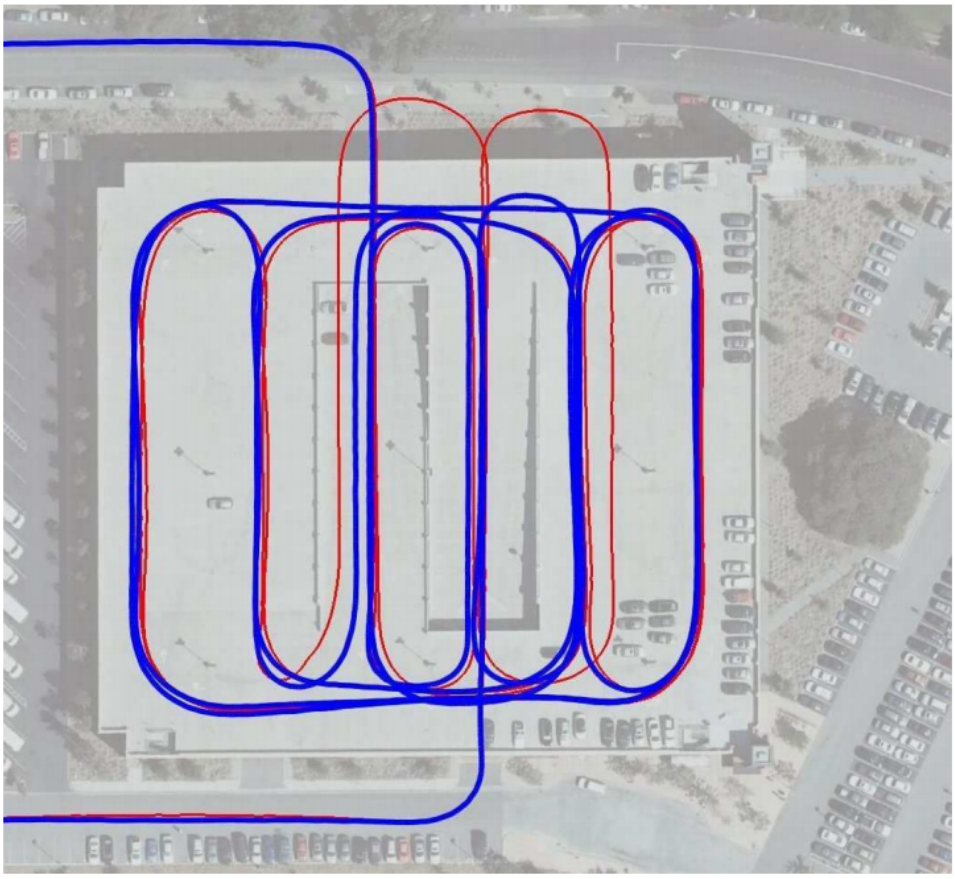


Figure 2.20: Top view of a large scaled garage parking house. The red trajectory is pure INS, the blue uses the authors approach. Clearly one can see the INS even beliefs to "leave" the building. The plot has been taken from [59].

The used robotics car is "Junior", which already participated at the 2007 DARPA urban-challenge. The authors use the top mounted 3D [1] and two side mounted laser scanner[2]. As the spatial resolution is very high, the 6-DQF ($< x,y,z,\phi,\theta,\psi >$) localization can be reduced to 3-Degree Of Freedom (DOF)$< x,y,z >$. The position is determined with using a 1.000 particles MCL and odometry at 200Hz. The prediction is based on INS and odometry, the correction on laser and (if it is possible) GPS, e.g. on the top level of the garage park. The initial start position is given by GPS.

The authors use the "GraphSLAM", see [66], algorithm to localize. To represent the world, the authors use a multi-level surface map. It consits of a 2D grid which contains height of surface patch and its variance. The implementation is very efficient with 128MB of size for a 20x20cm grid dimension. The map is

1 Velodyne HDL-64E, see [65]
2 Sick LMS200, see [64]

updated by adding new nodes containing the actual pose.
The problem of "closing loops"[1] is solved using 3D scan matching, which can identify previously visited positions. The same cell can be visited at different heights. To determine if the car has moved to another floor the number of updated cells is counted. If exceeding a threshold, the grid map will be cleared and prepared for the next level.

$A*$ with a cost function is used to plan the path. The overall system is able to drive through the multi level garage house with 6.6 to 9.5 km/h.

"Range sensor based outdoor vehicle Navigation, collision avoidance and parallel parking", see [67], examines the usage of sonar sensors for fast obstacle detection.

The used robotics car is equipped with a color camera and a 3D ERIM laser[2]. Booth sensors deliver reliable distance information but lack a long data processing time. Back in 1995 this meant too slow for fast obstacle detection. Based on this, the authors suggest to use sonar sensors.

Due to wavelength limitations only rough surfaces or edges reflect sonar echo, but outdoor environment often consists of them. To be used in outdoor environment sonar sensors need to be robust against moisture and dust, and specially robust against other vehicle noise sources, e.g. vibrations from the engines.

A 9 Hz trigger rate is used, which results in 50 ms traveling (again 50 ms echo) time which enables the sensor to measure distances up to 8 m. To reduce the probability of mis-measurement only the first echo is used. Compared to indoor usage the sonar sensor is more reliable because generally objects are bigger. Therefor reflections are more likely to occur due to objects than to false positives. When mounted on cars a main error source are reflections from the ground. To avoid this, sonar sensors should be mounted high enough and point upwards, as suggested by the authors. During their experiments they could achieve an average accuracy up to $\pm 1cm$. The drawback of sonar sensors is their low update rate (due to their measurement principle and the low traveling speed of sound). This can be bypassed when traveling slow, as the authors suggest when trying to detect parking lots.

To find gaps when passing by parallel parked vehicles the authors uses a local grid map which contains an occupied or free flag, the position and the cells history. If an obstacle has been seen before, a value will be added. The obstacle disappear from the map if it has not been seen for a while.

1 reentering an already visited area
2 a late 1980 3D laser scanner

"Autonomous parking in subterranean garages-a look at the position estimation", see [61], introduces a production near car capable to find a parking lot on its own.

The authors state that modern x-by-wire driver assistance system already enable autonomous driving. For their experiments they require sensors with high spatial resolution. Based on this their are using a production like Mercedes Benz S500 series equipped with side mounted Sick LMS200[1]. A blueprint of the garage house will be used as priori information represented within a 2D gridmap. To determine the initial start position a gateway at the entry is used.

The measurement data from the laser scanners is evaluated using least square error scan matching against the map. This procedure is presented as very efficient and capable of real time procession (around 40 ms duration time). To reduce computation the scan is matched only to potential visible areas based on the last known good position. For every scan the variance and curvature is calculated. Based on this a weight for every scan is calculated. An EKF fusions the scan data with the odometry.

The authors determined for short parking manuevers (<20 m) the odometry error can be ignored. Therefor the system can relay only on odometry data for the actual parking manuever. The experiments showed no position losing up to a speed of 8 m/s. Those were done for a parking garage up to 90% taken parking lots.

2.5.4 autonomous cars tested on public roads

The following section gives a brief overview to actual existing robotic cars out on public roads.

Google street car project uses 7 cars (6 Toyota Prius and 1 Audi TT). Until 2010 1.609 km were driven without human interaction and 225.308 km with occasional intervention on public roads. A Velodyne HDL-64E, see [65] has been used as the main sensor. GPS and odometry to predict the movement. Cameras are used for obstacle detection. **H. Altinger NOTE: there are no papers available, just online sources like http://www.techradar.com/news/world-of-tech/car-tech/google-granted-driverless-car-patent-1048313**

"Stadtpilot: Driving Autonomously on Braunschweig's Inner Ring Road", see [55], the authors present a successor of a DARPA urban challenge participant. In contrast to simulated traffic in a rather controlled environment,

1 manual see [64]

Leoni, a robotics car of TU Braunschweig, drove through city of Braunschweig during normal working hours. The car has to move up to 60 km/h, merge into moving traffic, behave correctly at traffic signs and master intersections.
The car is build rather straight forward, Velodyne on the top, LIDAR on the front and side, grid map to represent the world and global path planing to navigate. The difference is a real surrounding and a different approach to decision making. The teams regulary operating on the inner road ring of the City of Braunschweig.
Within this publication the authors strident the topic of law related to autonomous driving. They state that a robotics car can drive on its own, but there must be a drained driver able to take over at any time.

2.5.5 ultrasonic sensors

G. Steinbauer NOTE: are there any papers dealing with ultrasonic sensors and low update rates; are there any tricks that can be used

The following section reviews 3 papers examining processes with ultrasonic sensors. The principle of data acquisition is discussed as the application of using ultrasonic sensors with Markov Localization and a new approach to resampling.

"Digital Time of Flight Measurement for Ultrasonic Sensor", see [56], describes a digital pulse echo measurement system with the aid of cross-correlation functions to determine the Time of Flight (which is directional proportional to the distance) of an ultrasonic signal.
If measuring distances beyond 1m a long signal sequence needs to be recorded, which increases processing time. The authors state a finite duration signal, e.g. a sin-function, a shorter period needs to be recorded. First the received signal needs to be demodulated and low-pass filtered. This benefits not to require a high accurate carrier frequency sensing, therefor the system can be much simpler. Low-pass filtering reduces spectral disturbances and noise. The reduced frequency can be oversampled, which reduces conversation errors.
The authors present experimental results with an achievable accuracy up to 0.7 mm with a linear relationship between measured data and set up distances. They used the same module as transmitter and receive, as it is common to automotive systems. The presented algorithm can be implemented on Digital Signal Processor (DSP) with rather low performance due to only 256 point data sequences to be computed.

Within the PhD Thesis **"Robuste Navigation autonomer mobiler Systeme"**, see [57], the author compare different localization methods. In terms of absolute position error Markov Localization with sonar sensors performs worse compared to Markov Localization with LIDAR or laser scan matching approaches. It is states that Markov Localization is very good in terms of

global position and recovery after a position lose occur. Additional Markov Localization is demonstrated as a very robust localization system against high noise if the probability values are tunes correct. But in all compared cases methods based on LIDAR perform better than the same method based on sonar, due to the higher spatial resolution of the laser and the multi reflection problem of the sonar.

"New Resampling Algorithm for Particle Filter Localization for Mobile Robot with 3 Ultrasonic Sonar Sensor", see [58]. A LEGO NXT mindstorm robotic set is used to build the real robot. It only carries 3 ultrasonic sensors. With this stint the authors state traditional MCL divergence very fast and need long time until convergence. Initial pose of robot needs to be known and solving the kidnapped robot problem is not always possible. Due to this limitations a new resampling algorithm is proposed. Instead of copy good particles, new particles are drawn from the initial distribution. 90 % will be placed near good particles, 10 % will be sampled within the whole world. The particles weight will be normalized after estimation and afterwards adding the new particles. The authors expect to solve the kidnapped robot problem based on this.
During their experiments three different resampling algorithms were benchmarked, Linear Time Resampling (LTR), Select With Replacement (SWR) and the new proposed. The results show a significant faster convergence (60 time stamps compared to 150 steps) and a lower position error (5 cm instead of 20 cm) for the new algorithm. The source of ground truth is not given.

2.5.6 sensor problems

"Sensor Resetting Localization for Poorly Modelled Mobile Robots", see [30], presents an extension of MCL which performs better when the robot is delocalized. The used robot is a Sony 4 Leg AIBO within the RoboCup[1] standard platform league 1998. Previously they used particles with prediction (robot locomotion) and update (landmark based vision sensor). Even with 400 samples they got problems within computation and missed sensor update cycles. MCL can not recover from systematic errors in movement.

The authors present Sensor Resetting Localization (SRL), which usees smaller sample sets, but more robust to systematic error and unmodeld movements. The key idea is to replace under weighted samples by new samples drawn from the actual sensor distribution. In comparison to MCL the authors use the same prediction steps but when performing the sensor update they calculate a number of new samples to replace old ones.

1 www.robocup.org

Within the experiment section a detailed performance comparison with MCL is given. SRL is able to deliver similar results, but with less samples, which reduces computation requirements. The authors present simulation results with different levels of noise and systematic errors. SRL is able to reset itself before errors accumulate.

3 solution

H. Altinger NOTE: using chapter to give basics in self localization and used methods

G. Steinbauer NOTE: give hints which papers were used

3.1 System overview

G. Steinbauer NOTE: define/introduce every formula symbol here

This section will give a short overview about the implemented system. Figure 3.1 shows a schematic representation of the systems process cycle which is required to calcualte a position hypothesis. As it is a development system, the evaluation path is sketched too. The output of the system need to be compared with a reference to determine the position hypothesis performance.

G. Steinbauer NOTE: explain why one need Figure 3.1 –> to solve localization!

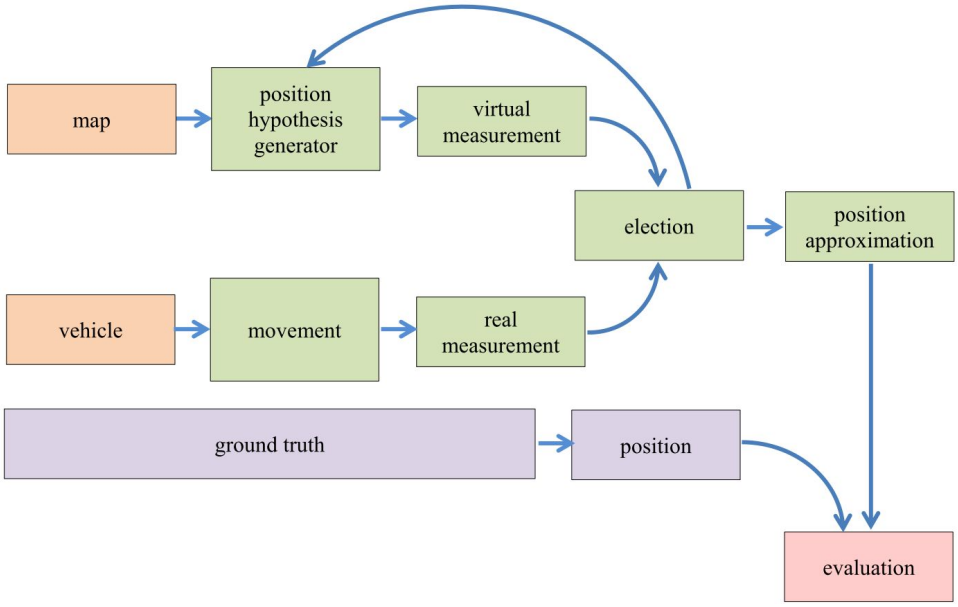


Figure 3.1: Red rectangles symbol inputs and outputs to the system, green rectangles the systems calculation steps. Purple rectangles represent the reference system.

Within the Automotive Data and Time-triggered Framework (ADTF) framework, refer to Section A, every module is realized within so called filters. During this thesis there is a general terminology: software modules ending with ”Server” realize such a filter. Those modules are used to receive and send data, acquire user setting, etc. Software modules ending with ”Lib” hosts the functionality. They are compiled as Windows static libraries and linked against the filter modules. All units follow the ”international system of units”, distances are measured in [m], time in [s], angles in [rad].

Figure 3.2 shows a schematic representation of all involved modules. The system can be the real world, where the car has to move and the walls are made up from concrete. Blueprints will be used as an a priori information for the map. When the world is simulated the care moves along a path, and the walls from the blueprint will feed the range sensors. The Data AcQuisition (DAQ) contains two sensor types, range sensors (ultrasonic) and motion sensors (accelerometer and wheel encoders). Their driver modules can not difference between simulation and real world data. Distance sensors will be used to update the occupancy grid map. Motion sensors will be used to calculate the odometry based on Egomotion Master, see [68] (EgoMaster) approach, refer to [68].

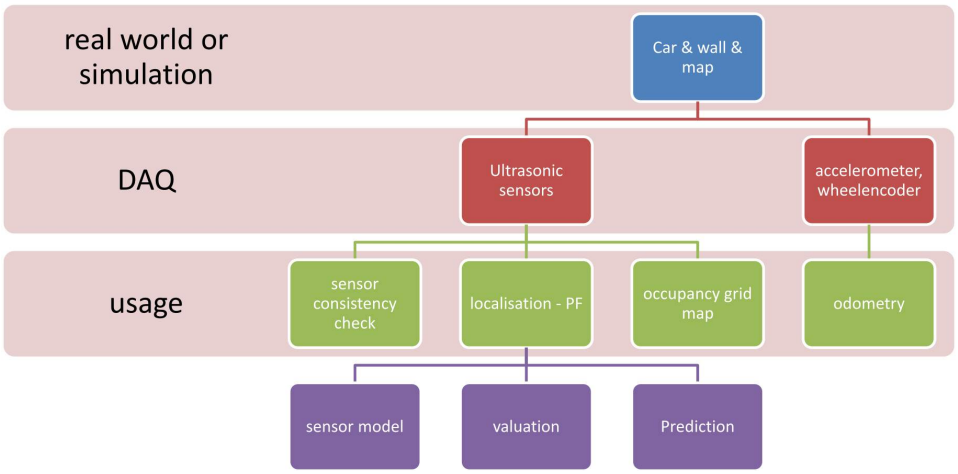


Figure 3.2: schematic module overview

The real filter architecture will be shown within figures 3.3 to 3.7. The following (none self developed) filters were used in addition to the system filters:

- *HPUSData*[1]
- *EgoMaster*[2]
- *CSIOdometryServer*[3]
- *Autopark_filter*[4]

1 Sensor driver for Valeo High Performance Ultrasonic Sensors, which is an external module developed by Manuel Geiger (I/EF-56).

2 Drivers for wheel-encoders and accelerometers and the dead Reckoning for the odometry are called EgoMaster, which is an external module developed by Christopher Demiral (I/EF-56).

3 the CSIOdometryServer is an external module developed by Patrick Gläßer (I/AEV-3).

4 the Autopark_Filter is an external module developed by *fortiss GmbH - An-Institut der Technischen Universit"at M"unchen*. For documentation consolidate the manual [69].

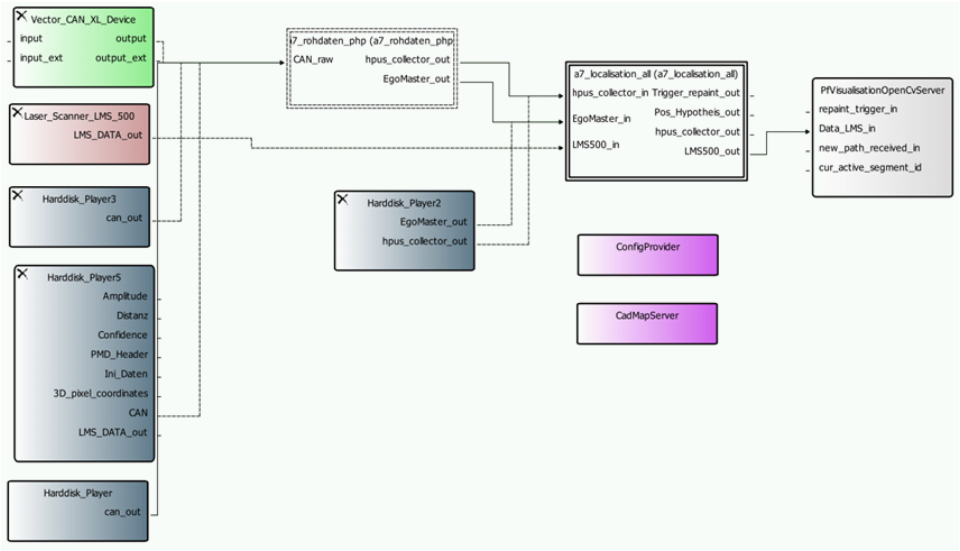


Figure 3.3: The system is realized within ADTF an split into 4 parts. The outer left **VECTOR_CAN_XL_DEVICE** is a system driver to the computers CAN interface. Below the **Harddisk_Player** is another system module, which can be used to replay recorded raw data. **a7_rohdaten_php** is a so called sub configuration. It is responsible to extract the information from the raw CAN messages, see Figure 3.5. **a7_localisation_all** is another sub configuration used to fulfill the localisation task, see Figure 3.7. On the right outer side the **PfvisualizationOpenCvServer** module hosts the and can be used to record video sequences. The representation will be updated if a trigger signal is received via *repaint_trigger_in* input-pin.

3.1.1 architectural overview

As sketched within Section 3.1 Figures 3.4, 3.5, 3.6 and 3.7 will give a more detailed view of the implemented system. Detailed explanation will be given within the figures caption. Nitty-gritty details to the core filter, hosting the particle localization, can be found within Chapter 3.2.

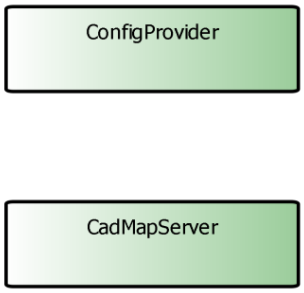


Figure 3.4: The filter **ConfigProvider** can be used to distribute configuration parameters to all modules. Every filter can access an interface to get the struct *tCarconfig*. The beneath filter **CadMapServer** will be used to parse a .obj files. Those should be CAD blueprints of the building. The filter will extract a specific volume and converts it into 2D lines. These lines can be access via an interface.

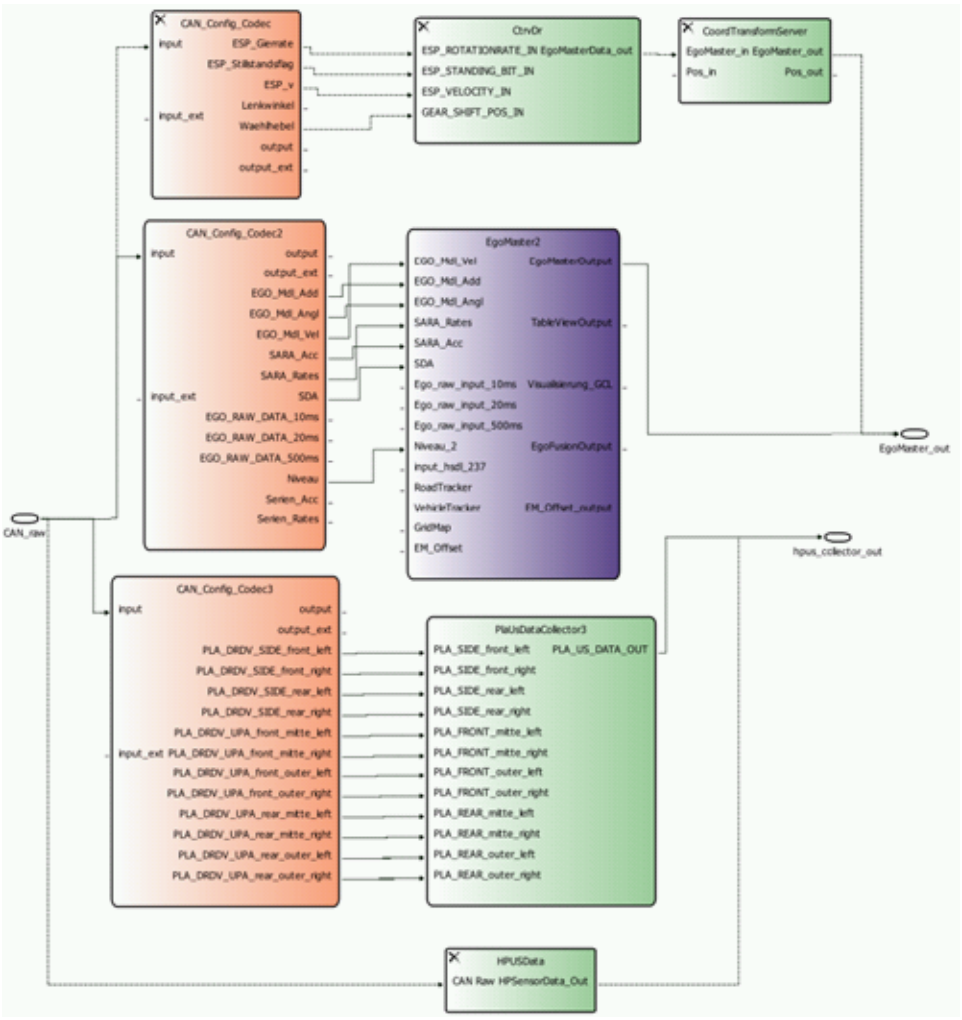


Figure 3.5: ADTF contains a module called **CAN_Config_Codec** which can be used to extract specific CAN messages from the raw data stream (coming from the input-pin **CAN_raw**). The upper path is used to acquire data necessary for the EgoMaster. The module **EgoMaster2** is an odometry implementation following the EgoMaster approach as explained by [68]. The output-pin **EgoMaster_out** will transmit the *tEgoMasterData* struct. The lower path extracts the ultrasonic range measurement data. The filter **PlaUsDataCollector3** collects the raw data, converts the measurement and transmit the data via an output-pin *hpus_collector_out*, according to the *tHPUsData* struct.

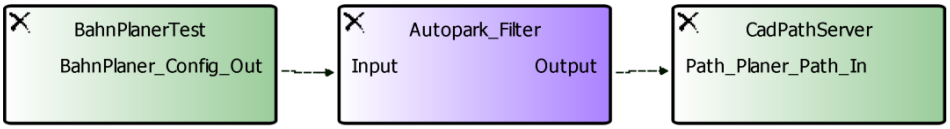


Figure 3.6: The upper path is used to calculate a path. The filter **BahnPlanerTest** can be used to parametrize the **Autopark_Filter**. The **CadPathServer** can be used to store the calculated path.

All modules requried to estimate a position are shown within Figure 3.7. The odometry data via the input-pin *EgoMaster_in* (needs to be connected to *EgoMaster_out*, see Figure 3.5) will be used to trigger the particle filter update (**triggerGenerator**. Every n^{th} odometry sample will initialize a particle filter step). The **particleFilterSimulationModule** can be used to exchange real motion and distance data with simulated values. (If the simulation is disabled, the data via the input-pins will be passed through). The filter **HPUSDataCollector** will assemble single range measurements from ultrasonic senors and will forward them if all sensor have reported, or a timeout has occurred. The **ParticleFilterServer** is a MonteCarlo implementation, see Section 3.2 for a detailed description. The **SensorFusionServer** will collect all sensor data (including the particle filter) and serve them via an interface. At the moment only odometry and particle position will be fused with a Kalman-filter. If this filter detects a jump within position, it will reset the particle filter at the last known good position.) The actual position will be transmited via the output-pin *Pos_Hypothesis_out* according to the *tPose* struct. If a Filter requires historical position data, the **CSIOdometryServer** can be accessed, it will store the position bind with a timestamp. The **OcypancygridMapServer** represents a grid map. When triggered via an input-pin it queries the **SensorFusionServer** and updates the map. This map can be accessed via an interface.

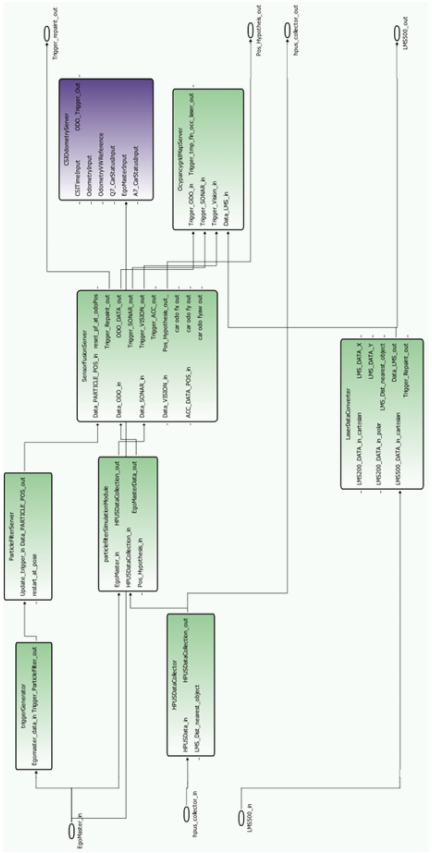


Figure 3.7: ADTF part localization: Modules required to estimate a position.

3.1.2 Synchronisation

G. Steinbauer NOTE: explain sync between data (odo, hpus, ...) –> timestamps, sync via memory (CSIodometry)

The ADTF framework supports the usage of so called timestamps. At acquisition every message is assigned a time based on a realtime hardware clock. When receiving position samples from the EgoMaster they are stored within the *CSIodometryServer*. This module uses a ring buffer to store data. Every module can request a position at a specific time. It there is no position stored for the requested time, an interpolated position between the two nearest stored values will be returned. The base time to request a position are the sonar sensor time, because they trigger the particle filter to update is state. The Kalman filter uses the position from the EgoMaster or the particle filter. Compare Figure 3.8. The ADTF framework uses a hardware clock to generate the timestamps. If using multiple instances a time synchronization protocol like Reference Broadcast Synchronization (RBS), see [70], is used.

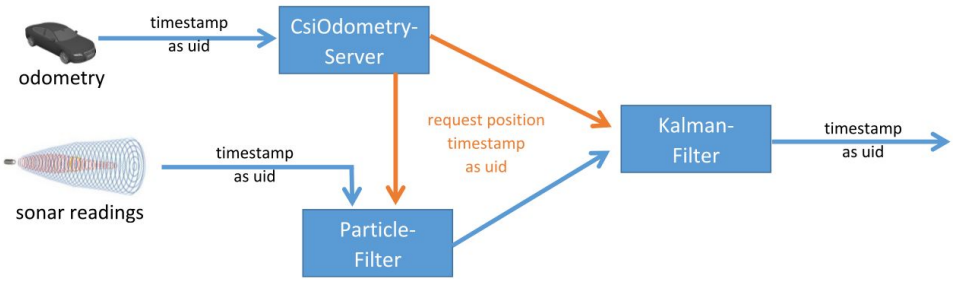


Figure 3.8: Short overview to data synchronization via memory and timestamps.

3.2 Monte Carlo particle filter

H. Altinger NOTE: using section to describe Monte Carlo particle filter; basics; advantages; disadvantages, ...

H. Altinger NOTE: basics of implementation; focus on implemented sensor models
G. Steinbauer NOTE: try to show evolution of solution; give reasons for next step

The used implementation follows straight forward the described method from Chapter 2.3.3. It consists of a sensor model, see Section 3.2.1, a motion model, see Section 3.2.2 and a resampling strategy, see Section 3.2.4. Those implementation can be combined by selection at system startup.

3.2.1 ultrasonic sensor models

Ultrasonic transducers are used to measure distances. The sensors principle is based on traveling ultrasonic[1] sound waves. One transducer transmit a short burst which is reflected by an obstacle. The sensor measures the time difference between transmit and receive. Based on equation 3.2 the distance can be calculated.

$$C = 331.45 + 0.607 \cdot T \tag{3.1}$$

$$x = \frac{t}{C} \cdot \frac{1}{2} \tag{3.2}$$

T represents the actual Temperature in $[°C]$, t the measured time in [s] and x represents the distance in [m]. The factor $\frac{1}{2}$ is required, because the wave has to travels forward and back.

Compared to a LIDAR sensor, which operates with small beams enabling the sensor to report a distance and an angle to the nearest object, an ultrasonic sensor can only report a distance to the next source of reflection.

1 for automotive applications 70 kHz are common

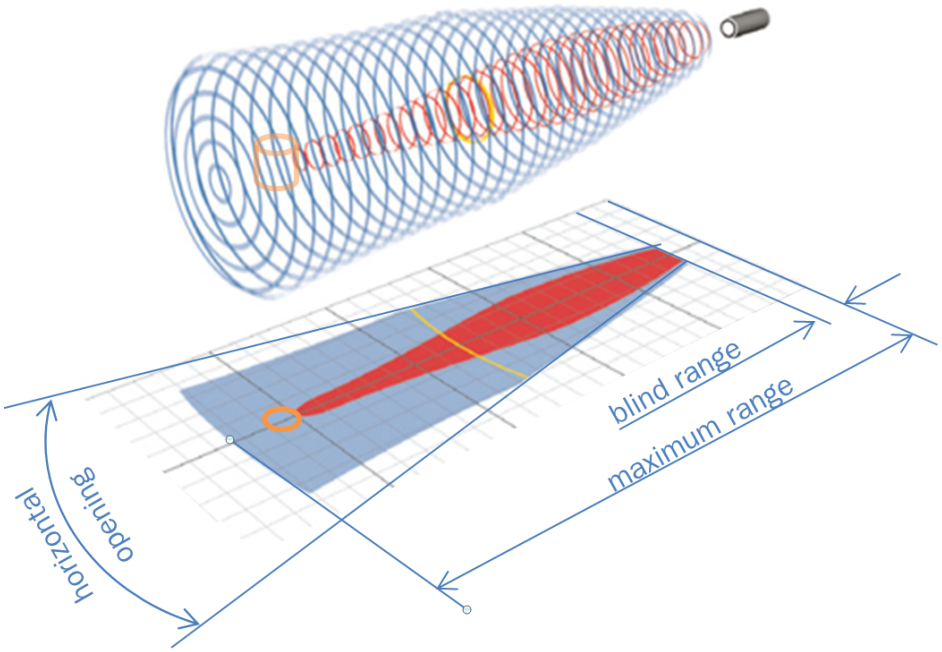


Figure 3.9: Transmitted sound bursts (blue) will be reflected at an obstacle (orange) and travel back (red) to the receiver. Based on the shape and used frequencies there is a characteristic horizontal and vertical opening angle. In automotive applications the transmit and receive uses the same membrane. The necessary time to switching between transmit and receive causes the blind range. The maximum range depends on the sound intensity and the maximum receive time.

For real parameters and setup within the experimental car, see Figure 4.3.

Based on Figure 3.9, where one can see the cone the sensor covers, the implemented models splits the area into n equal beams. Every single beam can be approximated with a probabilistic model. Figure 3.10 visualizes this approach. The whole sensor model is designed to always report the shortest distance for all beams.

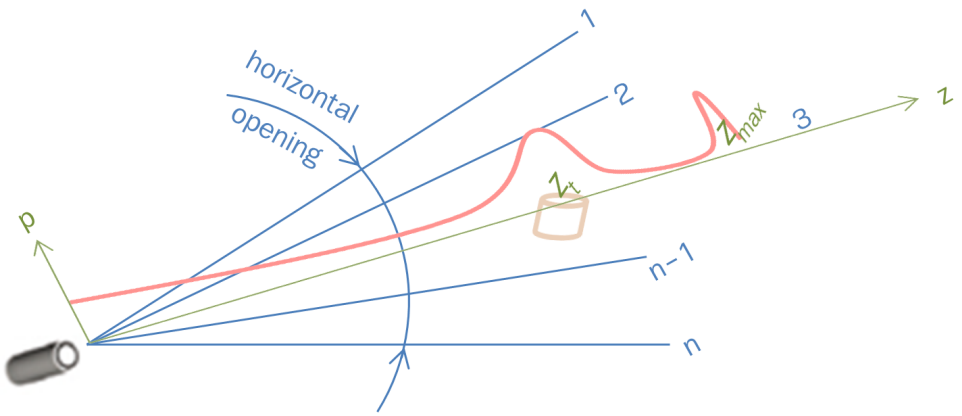


Figure 3.10: The sensors horizontal cone is split up into n equal beams (blue lines). For every single beam (green line) can be approximated with a probability model (red line). The implemented model uses z_{max}, the maximum range of the sensor and z_t, the measured distance. Within [20] a more detailed model is explained.

Thrun, Burgard and Fox refer to this as "Beam Models of Range Finders", see [2]. The model itself is split up into 4 parts (see Figure 3.11):

- Gaussian distribution p_{hit} to model the reported distances uncertainty, see equation 3.3
- Exponential distribution p_{short} to model a surface reflection or a random object, causing shorter distance, see equation 3.6
- Uniform distribution p_{max} to model a sensor reading a maximum distance, see equation 3.8
- Uniform distribution p_{rand} to model a random measurement error, see equation 3.9

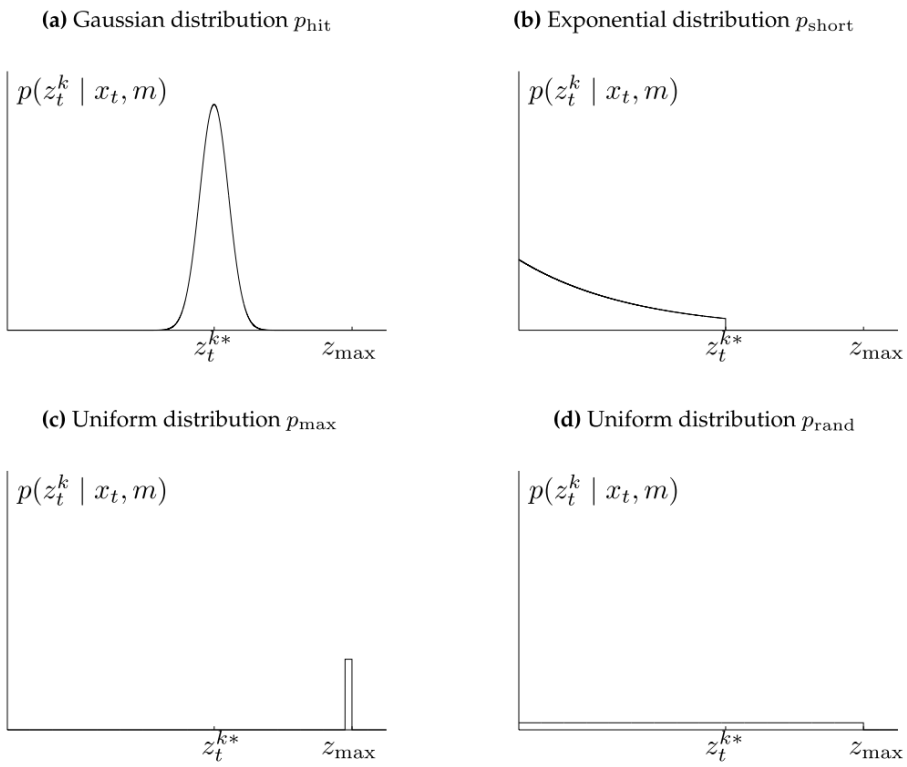


Figure 3.11: 4 basic components the "beam model for range finders" consists of. The figure has been taken from [2] as granted on the authors website

The source of the following parameters are described within Section 4.3. Values can be taken from Table 4.2.

$$p_{hit}(z_t^k|x_t,m) = \begin{cases} \eta\aleph(z_t^k; z_t^k *,\sigma_{hit}^2) & \text{if } 0 \leq z_t^k \leq z_{max} \\ 0 & otherwise \end{cases} \tag{3.3}$$

$$\aleph(z_t^k; z_t^k *,\sigma_{hit}^2) = \frac{1}{\sqrt{2\pi\sigma_{hit}^2}} \cdot e^{-\frac{1}{2}\frac{(z_t^k - z_t^k*)^2}{\sigma_{hit}^2}} \tag{3.4}$$

$$\eta = \int_0^{z_{max}} \aleph(z_t^k; z_t^k *,\sigma_{hit}^2)\, \mathrm{d}k_t^z \tag{3.5}$$

$$p_{short}(z_t^k|x_t,m) = \begin{cases} \eta\lambda_{short}e^{-\lambda_{short}z_t^k} & \text{if } 0 \leq z_t^k \leq z_t^k* \\ 0 & otherwise \end{cases} \tag{3.6}$$

$$\eta = \frac{1}{1 - e^{-\lambda_{short} z_t^k}} \tag{3.7}$$

$$p_{max}(z_t^k | x_t, m) = \begin{cases} 1 & \text{if } z = z_{max} \\ 0 & otherwise \end{cases} \tag{3.8}$$

$$p_{rand}(z_t^k | x_t, m) = \begin{cases} \frac{1}{z_{max}} & \text{if } z \leq z_t^k \leq z_{max} \\ 0 & otherwise \end{cases} \tag{3.9}$$

When using this 4 components, the probability for a measurement to take place at the given position z_t^k will be:

$$p(z_t^k | x_t, m) = \begin{pmatrix} z_{hit} \\ z_{short} \\ z_{max} \\ z_{rand} \end{pmatrix}^T \cdot \begin{pmatrix} p_{hit}(z_t^k | x_t, m) \\ p_{short}(z_t^k | x_t, m) \\ p_{max}(z_t^k | x_t, m) \\ p_{rand}(z_t^k | x_t, m) \end{pmatrix} \tag{3.10}$$

The virtual measured distance will be determined using the occupancy grid map (see Section 2.2.1) and Bresenham's algorithm, see [71]. Figure 3.12 sketches the process. The sensor cone is split into multiple equidistant beams (purple lines). The blue obstacle occupies the red grid cells. Following every purple line with Bresenham's algorithm the green squares are visited. The first visited square marked as occupied will report the virtual measured distance.

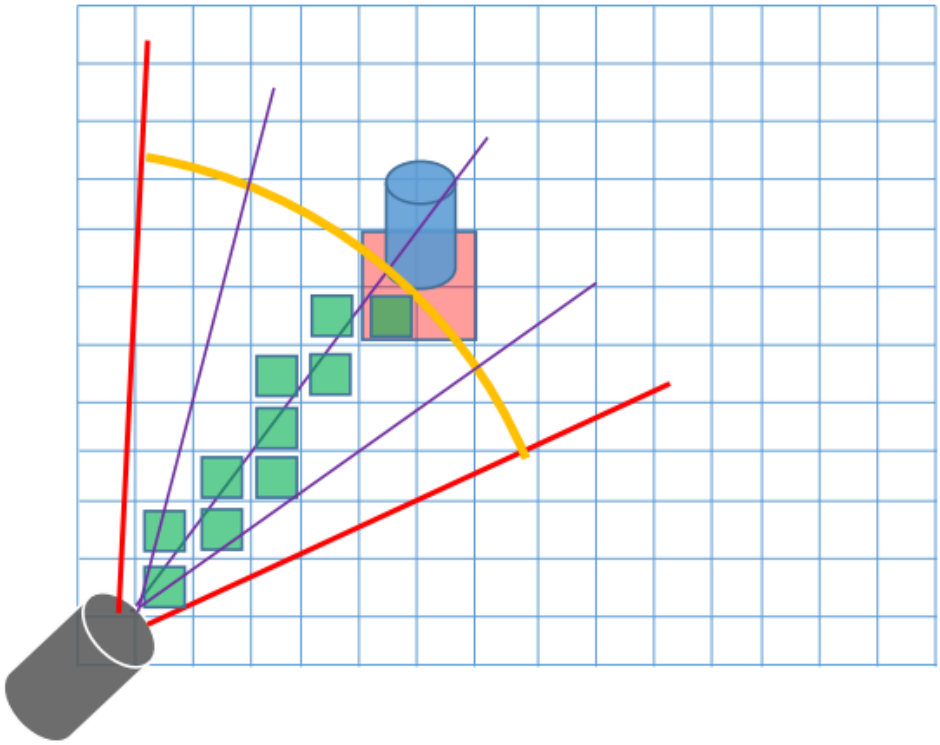


Figure 3.12: determine virtual measurement with occupancy grid map and Bresenham

The sensor model is used to weight the (virtual) particles sensor measurements ($d_{particle}$) against the (cars) real measurements (d_{car}). Every particle is calculated separate. The sum of all sensors per particle represent the particles calculated weight. All following description compare the actual particles sensor to the equivalent sensor at the real car.
There following two methods are implemented:

Simple Model

- If both measure free range, the particle gets award a point
- If one measures a distance, the particle loses a point
- If car measures a shorter distance than the particle, the particle gets award $\frac{1}{|d_{particle}-d_{car}|}$ point
- If car measures a longer distance than the particle, the particle loses $\frac{1}{|d_{particle}-d_{car}|}$ points

This model is based on a rough draft and the most easy to understand. It can be adjusted by adding percentage weights to the awarded points.

Beam Model[1]

If both (car and particle) measure something, the particle gets award points, according to equation 3.12 to 3.15. Otherwise there is no weight gain, but also no weight degeneration.

$$dist = |d_{particle} - d_{car}| \tag{3.11}$$

$$p_{noise} = z_{hit} \cdot *e^{-\frac{1}{2} \cdot \frac{dist^2}{\sigma_{hit}^2}} \tag{3.12}$$

$$p_{short} = z_{short} \cdot \lambda_{short} \cdot e^{\lambda_{short} \cdot z_{realmess}} \vee dist < 0 \tag{3.13}$$

$$p_{random} = z_{rand} \cdot \frac{1}{range_{max}} \vee dist > max(sensor - coverage) \tag{3.14}$$

$$weight_{sensor} = p_{noise} + p_{short} + p_{random} \tag{3.15}$$

This model fits the the model visualized with Figure 3.10 the best. It considers noisy environment, falls readings caused by, e.g. surface reflections, none map obstacles, and random sensor readings.

The beam model parameter estimation is described within [2]. The experiments are exemplified within Section 4.3.

3.2.2 motion model

Derived from Figure 3.13, a new position in general can be obtained by equation 3.16.

1 consolidate [2] for further details

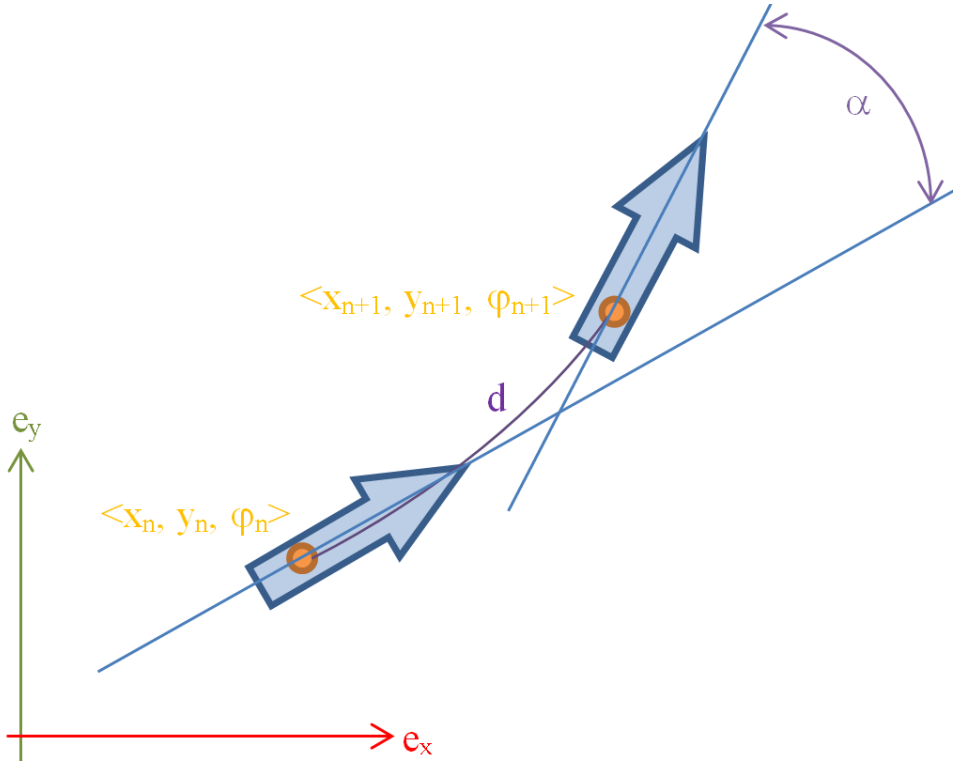


Figure 3.13: The object gets displaced along the path by d(purple line) and rotates by α

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \\ \omega_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \\ \omega_n \end{pmatrix} + \begin{pmatrix} d \cdot cos(\alpha) \\ d \cdot sin(\alpha) \\ \alpha \end{pmatrix} \tag{3.16}$$

Standard production cars can be modeled by "Ackerman-steering" if speed is low and steering angle is constant (as assumed during this thesis). As shown in [72] equation 3.17 can be used to derive the turn angle from the steering angle. Compare with Figure 3.14 for dimensions.

$$cot(\beta) - cot(\alpha) = \frac{d}{l} = const. \tag{3.17}$$

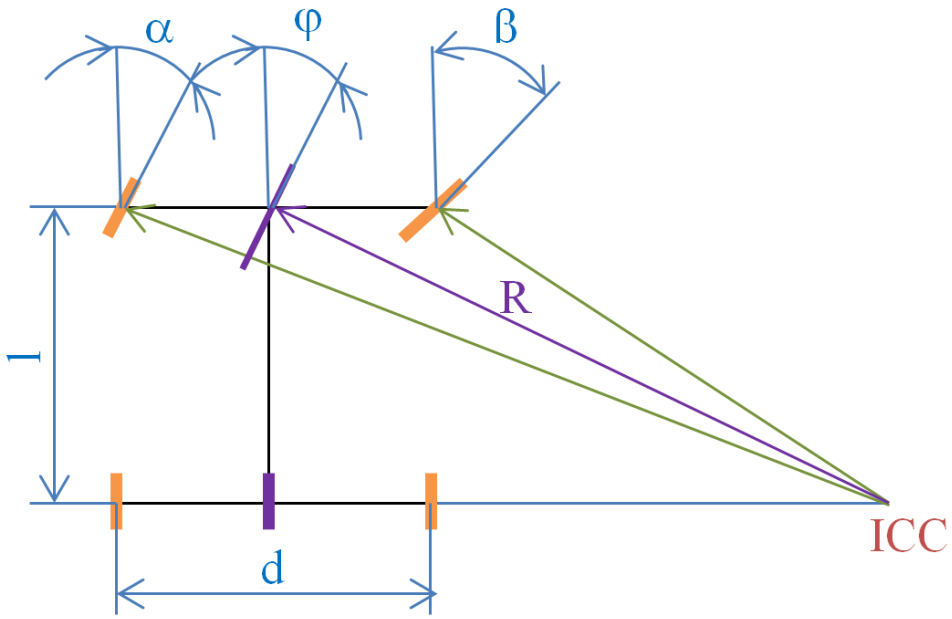


Figure 3.14: If a car turns, there is always the ICC which stays steady. For a classic Ackerman-steering 4 wheels (orange lines) are used (Note: $\beta > \alpha$), for single track model only one wheel (purple line) is approximated (Note: $\beta > \omega > \alpha$).

The single-track model simplifies the Ackerman-steering by reducing two wheels to one at the center. The new position according to equation 3.16 notation can be obtained by Figure 3.15, see equation 3.24 and 3.25.

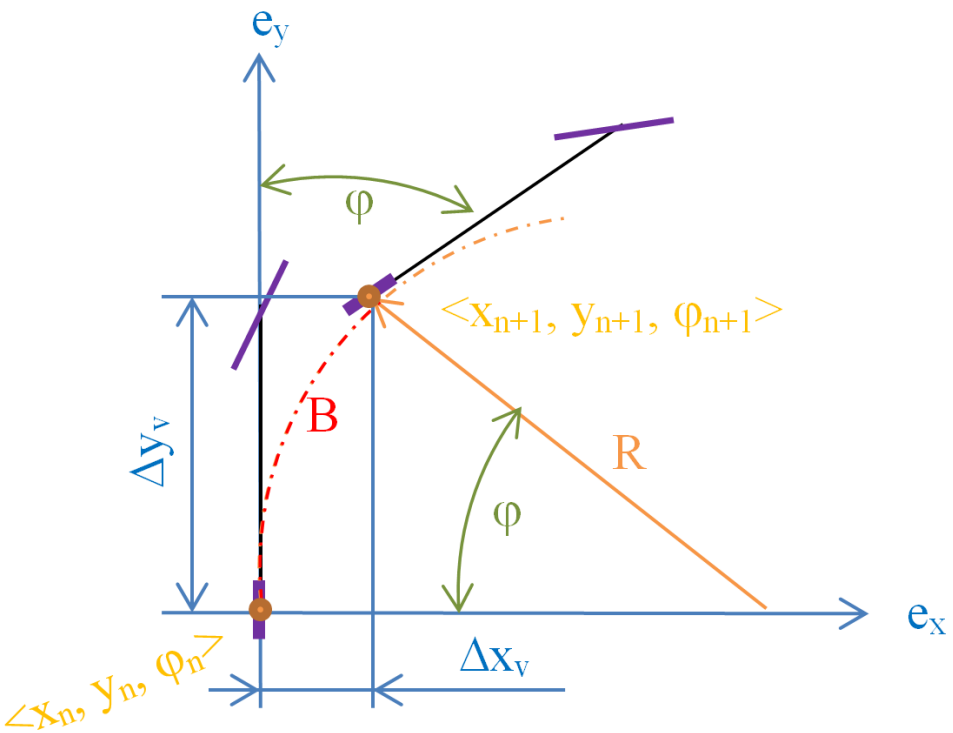


Figure 3.15: The single track model moves from step n to n+1 (during time T). It travels along the a path (orange doted line) constitute by the steering angles radius R and covers the distance B (red doted line) and turns by the angle ω (green).

$$B = v \cdot T \tag{3.18}$$
$$\omega = B \cdot R \tag{3.19}$$
$$\omega v \cdot T \Rightarrow R = \frac{v}{\omega} \tag{3.20}$$
$$\omega = \omega T \tag{3.21}$$
$$\Delta x_v = R(1 - cos(\omega T)) \tag{3.22}$$
$$\Delta y_v = R sin(\omega T) \tag{3.23}$$
$$\tag{3.24}$$

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \\ \Omega_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \\ \Omega_n \end{pmatrix} + \begin{pmatrix} \frac{v}{\omega} \cdot [sin(\Omega_n + \omega T) - sin(\Omega_n)] \\ -\frac{v}{\omega} \cdot [cos(\Omega_n + \omega T) - cos(\Omega_n)] \\ \omega T \end{pmatrix} \tag{3.25}$$

The CTRV model, as introduced by [73], is a reasonable good method to

model a cars motion[1]. During the experiments a steady acceleration will be assumed, therefor the car moves with a constant velocity, which satisfies the requirements by the CTRV model. Our sensors are able to deliver T, v, and ω, therefor the CTRV model can be used. [74] and [75] shows that the CTRV is a comprehensive motion model with a better error performance compared to Constant Acceleration (CA), Constant Velocity (CV), Constant Turn Rate (CTR), Constant Turn Rate and Acceleration (CTRA) or an other linearized model. The model expands the state space by the cars velocity v and rotation rate ω, see equation 3.26. x and y represents the position $[m]$, φ the heading in $[rad]$, v the velocity $[\frac{m}{s}]$ and ω turn rate $[\frac{rad}{s}]$.

$$\vec{x}_t = \begin{pmatrix} x & y & \varphi \end{pmatrix}^T \tag{3.26}$$

G. Steinbauer NOTE: eq 3.26 needs to fit formular 3.33

The next time step (T represents time [s] since last update) can be calculated with equation 3.27.

$$\vec{x}_{t+T} = \begin{pmatrix} x_{(t)} + \frac{v}{\omega}[sin(\varphi_t + \omega T) - sin(\varphi_t)] \\ y_{(t)} - \frac{v}{\omega}[cos(\varphi_t + \omega T) - cos(\varphi_t)] \\ \omega T + \varphi_t \end{pmatrix}^T \tag{3.27}$$

Equation 3.27 looks partial similar to equation 3.25. A much more detailed model of a cars motion, including considerations for acceleration and sideslip, can be derived using Maggi[2] equations, refer to [77]. Within this thesis the car will operate at constant low speed and therefor higher dynamics like sideslip, acceleration drift, etc. do not need to be considered.

3.2.3 resampling process

As suggested by [78], equations 3.28 and 3.29 are used to decide if the particle set (number of particles is M) set is degenerated and need to resample, w_t^i being the weight of particle i at time t. If necessary (if Effective Sample Size (ESS) drops below a parameterizable threshold). the method described within Section 3.2.4 issued to determine which particle to resample. The process will copy the best particles state (depending on resampling method) and replace bad particles.

1 the Ackerman model is assumed
2 see [76]

$$cv_t^2 = \frac{var(w_t^i)}{E^2(w_t^i)} = \frac{1}{M}\sum_{i=1}^{M}(M \cdot w_i - 1)^2 \tag{3.28}$$

$$dist = \frac{M}{1 + cv_t^2} \tag{3.29}$$

Which particles will be resampled is decided by the resampling methode, see Section 3.2.4.

3.2.4 resampling methods

H. Altinger NOTE: describe different aproaches; advantages; disadvantages; focus on reason for slot-strategy
G. Steinbauer NOTE: try to show evolution of solution; give reasons for next step

In general therms, a particle may be judged by it's weight p. A particle described as "good" satisfies the condition and will sustain its state. A "bad" particle is the opposite and will be set to a new state (weight, position, etc.) within the next execution-step $t + 1$.

Stratified

Follows the formulas suggested by [32] to calculate the number of good particles, see equation 3.30.

$$dist = \frac{1}{\sum_{n=1}^{particles}(w_n^2)} \tag{3.30}$$

If the number of effective good particles is too small, the filter has degenerated. Within the next step only particles above a desired threshold β_{eff} will stay, all others will be resampled global.

slot strategy

This methde has been designed to be able to use and support good particles. As suggested by [32], if $pf_{ess} \leq \beta_{resample}$ resample is required. pf_{ess} see equation 3.31.

$$pf_{ess} = \frac{num_{valideParticle}}{1 + \sum_{i=1}^{N_s} p_{particle}^2} \tag{3.31}$$

The highest weighed particle will serve as the reference. All particles between $p = 1$ to $p = p_{select}$ (see equation 3.32) will be good particle (n_{good}). All others ($n_{resample}$) will be resampled, but at least $n_{globalresample}$ to prevent degeneration of the filter. All $n_{resample}$ particle will be equally distributed around each n_{good}. A sampling area around each good particle can be defined with ζ_x and ζ_y, see Figure 3.16.

$$p_{select} = max_{i=1}^{N_s} p_{particle} \cdot \beta_{threshold} \tag{3.32}$$

$\beta_{resample}$, $\beta_{threshold}$, ζ_x, ζ_y, $n_{globalresample}$ are the methodes adjustable parameters.

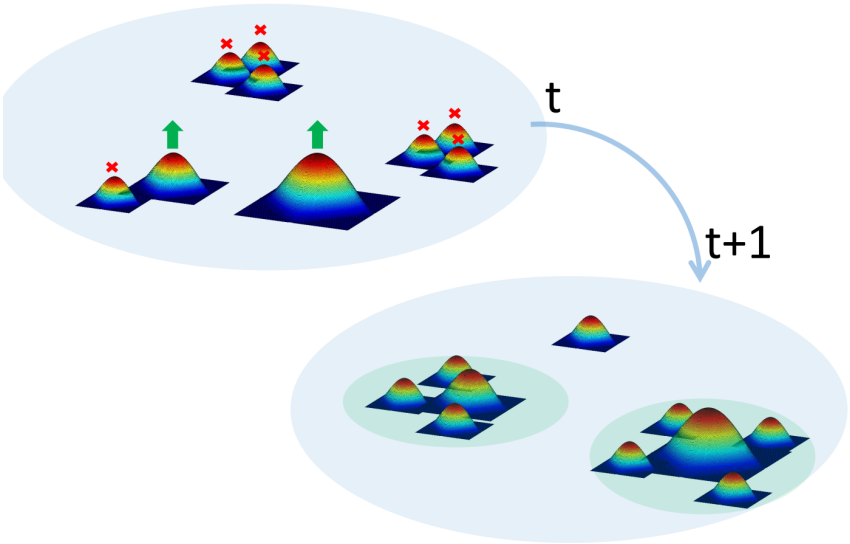


Figure 3.16: $n_{resample}$ particle (red cross) will be spread around (green circle) n_{good} particles (green arrow), $n_{globalresample}$ particles will be spread globaly (blue circle)

3.3 Kalman filter

H. Altinger NOTE: basics of implementation
H. Altinger NOTE: used models and derived matrix

Based on the mathematical basics presented within Section 2.3.2 a Kalman filter has been implemented to merge the odometry position with the particle filters position. Equation 3.33 shows the state change; the current internal state will be exchanged with the new odometry data. Based on equation 2.23 this causes the transition matrix A_t to be zero and B_t to be the unity matrix.

G. Steinbauer NOTE: formal correct introduction: A = one, odo

and pf as measurement update
G. Steinbauer NOTE: dimensions need to fit definition at 3.26

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \\ \varphi_{t+1} \end{pmatrix} = \begin{pmatrix} x_{odo} \\ y_{odo} \\ \varphi_{odo} \end{pmatrix} \tag{3.33}$$

$$A_t = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.34}$$

$$B_t = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.35}$$

The measurement update is defined as 3.36, based on equation 2.25 this causes the matrix C_t to be the unity matrix.

$$\vec{z}_t = \begin{pmatrix} x_{pf} \\ y_{pf} \\ \varphi_{pf} \end{pmatrix} \tag{3.36}$$

$$C_t = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.37}$$

Leftover are the system noise and the measurement update error. Equation 3.38 and 3.39 define the derived form of R_t and Q_t.

$$R_t = \begin{pmatrix} \sigma_x^{odo} & 0 & 0 \\ 0 & \sigma_y^{odo} & 0 \\ 0 & 0 & \sigma_\varphi^{odo} \end{pmatrix} \tag{3.38}$$

$$Q_t = \begin{pmatrix} \sigma_x^{pf} & 0 & 0 \\ 0 & \sigma_y^{pf} & 0 \\ 0 & 0 & \sigma_\varphi^{pf} \end{pmatrix} \tag{3.39}$$

Based on [2], Table 3.1 the Kalman filter has been implemented. The filters

state can be calculated according to equation 3.40 and is represented by μ_t and the covariance Σ_t

$$\begin{aligned}
\overline{\mu}_t &= A_t\mu_{t-1} + B_t u_t \\
\overline{\Sigma}_t &= A_t\Sigma_{t-1}A_t^T + R_t \\
K_t &= \overline{\Sigma}_t C_t^T (C_t\overline{\Sigma}_t C_t^T + Q_t)^{-1} \\
\mu_t &= K_t(z_t - C_t\overline{\mu}_t) \\
\Sigma_t &= (I - K_t C_t)\overline{\Sigma}_t
\end{aligned} \tag{3.40}$$

4 experiments

4.1 the environment

The following section will give a brief overview to the used environment. This will contain the building to be tested within, the car as the sensor carrier and the ground truth to get reference data to evaluate the system. The building is the same for real data and for the simulation.

4.1.1 building

Various tests have bin done within the presented garage, which is one of the target areas for the whole project. Figure 1.1 shows a sketch of the environment. It is a typical subterranean garage house beneath an office building. The entry is a narrow ramp with concrete walls to the left and right. As common for many subterranean garage houses there are some pillars between the parking lots. During the tests some of the parking lots have been blocked by other cars, some workshop tools, etc. similar to some private garage houses.

The 3D model shown within Figure 1.1 is originated from the architects blue print. To keep the model as simple as possible, all details have been removed, e.g. water tubes, electrical installation cables, etc. Elements like fire extinguisher, electrical switches, etc. have never been mapped, therefore they are note part of the blueprint. The softwaremodule *Cad map server* reduces this 3D information and generates a 2D map. This is used to initially load the occupancy grid map. Figure 4.1 visualizes this map.

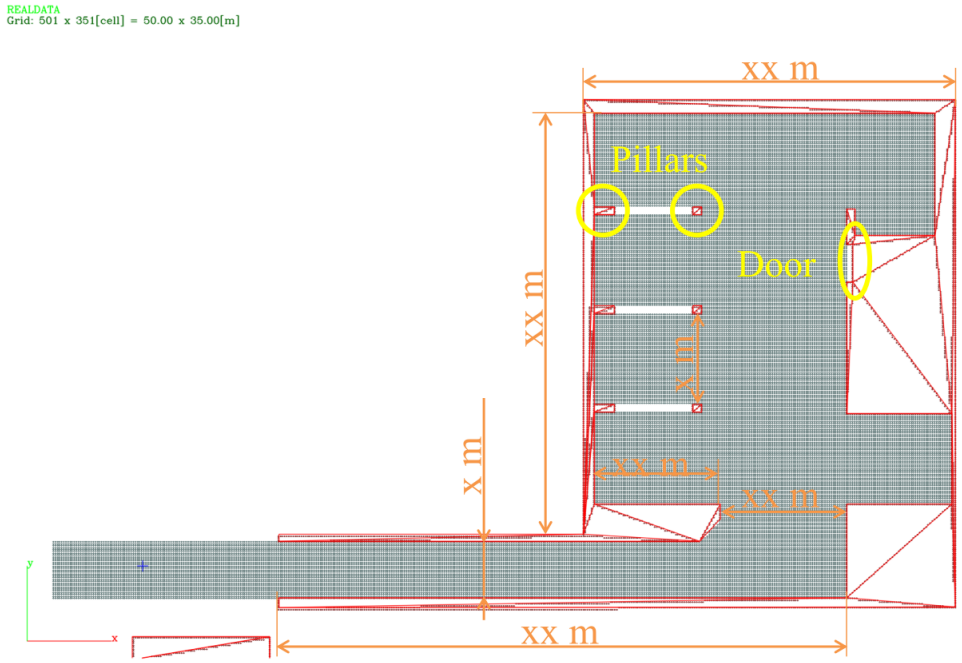


Figure 4.1: 2D blueprint generated from a 3D CAD model. Red lines mark the extracted wall lines (all 3D elements have been triangulated), the green surfaces the drivable areas, withe regions are not drivable. This represents the initial loaded map for the used subterranean garage house. The ramp is longer than in the 3D model to portrait the total length of the ramp.

As marked in Figure 4.1 the garage house is very narrow. The ramps width is around $2.5m$, the parking lots $5m$ x $2.4m$. The car has to move turning circles of 90 degree and $4m$ in diameter.

Occupancy grid map

As seen within Figure 4.1 the world is represented using an occupancy grid map, see Section 2.2.1. The discretization of the world has got a direct influence to the achievable accuracy to position quality. All obstacles will be entered into the grid, and all data can only be gained within the discrete world. The grid cells size can be adjusted, but is indirection quadratic proportional to the consumed memory. During all experiments the cell size has been set to $10x10cm$ due to computational limits.

Updating the map is realized by logodds, see equation 2.17. The updating sensors probability will increase or decrease the cells occupancy. The initial believe and an a-priori information for every grid can be initialized with data from a blueprint.

4.1.2 the car

H. Altinger NOTE: describe the basic sensors of normal A7(C7)

The car is a production like Audi A7, compare datasheet [79], model-year 2012. The standard sensors can be seen within Figure 4.2. During all tests the car has been equipped with the built in ultrasonic sensors, see Section 4.1.2.

Figure 4.2: The Audi A7 is equipped with different kinds of sensors: cameras (monocular daylight and infrared), radar (short and long range), IMU and ultrasonic. Due to their mounting position and sensor principles they can cover different areas and ranges. Source: Audi AG, Self-study program

SARA

H. Altinger NOTE: basics about accelerometer, gyros

A modified Sensor Array Audi (SARA) unit is located at the cars center of mass. It uses its own build in 6 DOF accelerometers & gyros and the Electronic Stability Control (ESC) wheel encoders to calculate an odometry position. A detailed description of the implemented position calculation is described by [68]. The system is able to deliver position updates at 100 Hz. It can be seen as an odometry with the support of an IMU. Both sensor types are merged with an EKF.

Compared against a ground truth the system delivers a position accuracy up to xxx cm and a standard error of xxx cm, see [80].

ultrasonic sensors

H. Altinger NOTE: principle of ultrasonic distance measurement, update rates, multi reflections, ...

A standard production A7 is equipped with up to 12 ultrasonic sensors, see [81]. The operational frequency is around 70 kHz. 4 sensors are mounted inside the front bumper and 4 at the back. Two sensors are mounted on each side of the car. See Figure 4.3 for position, sensor id and coverage area. Table 4.1 lists the maximum range and opening angles for the sensors.

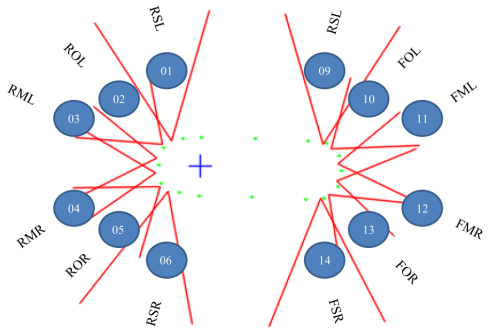


Figure 4.3: The car is equipped with 12 ultrasonic sensors. This picture shows each sensor with its ID to be used within the system. The naming convention as follows: First letter for **R**rear or **F**ront, second for **M**iddle, **O**uter or **S**ide and third for **R**ight or **L**eft (olways seen when looking into driving direction ot the vehicle). E.g. **RML** addresses the rear middle left sensor (id 3). For sensors parameters see Table 4.1

sensor	max. range	horizontal opening
	[m]	[°]
front	2.550	75
rear	2.550	75
side-front	5.000	45
side-rear	4.400	45

Table 4.1: operational parameters of 12 ultrasonic sensors

The used sensors are analog systems, which are connected to the Park Lenk Assistent (PLA) controller unit. The software has been modified to transmit the measured distances via CAN. As described within Section 4.3 the sensors achieve an average accuracy of 20 cm and need to be modeled with an average error of 20 cm.
The methode to model the sensor values within simulation and the particle filter is described within Section 3.2.1, the required parameter estimation within Section 4.3.

4.2 ground truth

To reference the estimated position a ground truth, named as OutsideIn, system has been used. It is based on environment fix LIDAR sensors, see [82]. A detailed analysis can be found within [83], a manual within [84]. The number of LIDAR sensors varies depending on the size and structural design of the test area.

The system is based on the triangulation principle described within Section 2.4.3. LIDAR sensor measures one plane surface, where a cars wheels can be identified. A RANdom Sample Consensus (RANSAC) algorithm tries to find 3 to 4 wheels of the car. With them the center of the car can be calculated. Assuming the car drives into the same direction for the first seconds a full heading can be extracted.

According to [82] the SICK LMS500-20.000pro archives an accuracy of $300mm$ for distances $65m$. As described within [83] the OutsideIn achieves an average accuracy up to $xxcm$. This is mainly caused by the implemented discretization. According to [85] the overall system achieves a repeatable accuracy of $20cm$. Within this application the OutsideIn is fusioned with the cars odometry via a Kalmanfilter.

4.3 parameter estimation for sensor model

G. Steinbauer NOTE: explain the experiment in detail! –> parameters determined via linear optimization (algo from Thrun)

G. Steinbauer NOTE: one plot of the used sensor model similar to probabilistic robotics, Figure 6.6 and 6.5

Section 3.2.1 introduces the the beam model for distance measuring sensors. This model is defined by 6 intrinsic parameters. As suggested by [2] the algorithm "**Algorithm learn intrinsic parameters**" has been used to determine the intrinsic parameters. The implemented methode uses a linear optimization. Figure 4.4 shows the measurement setup & process to obtain the data. The car is moving forward and backward against a solid wall. A LIDAR (blue cylinder) is mounted at the bonnet and measures the distance to the wall (red beam). Approximated ultrasonic sensing area (each between two purple lines) with its real beam cone (black line). The car is moved forward and backward to acquire static and dynamic distance measurements. Table 4.2 shows the optained parameters, Figure 4.7 visualize the model.

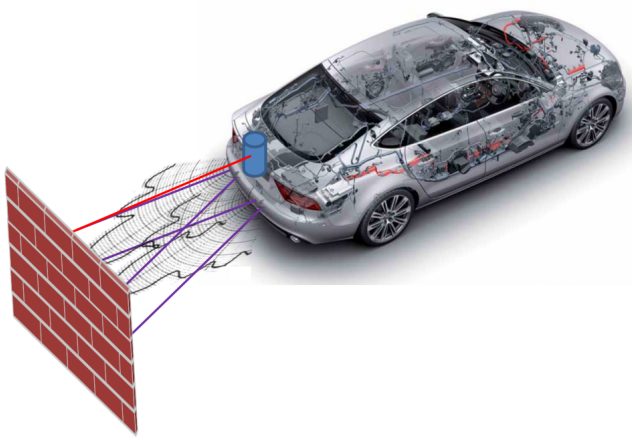


Figure 4.4: sketch ultrasonic sensor parameter estimation

Both ultrasonic sensor types have been analyzed due to their operational borders, reliability and standard error in measurement. Figure 4.5 shows raw data with standard production ultrasonic sensors, Figure 4.6 with high performance ultrasonic sensors. As the tested high performance ultrasonic sensor is within a very early development the high noise influence need to be reduced in further development released.
Figure 4.7 shows the obtained sensor model for the standard production and high performance sensors. At a compareable distance the high performance sensor delivers a less accurate real distance than the standard production. The high performance sensor has got less random, short and maximum measurements which can be seen in comparison within Figure 4.7.

Within Figure 4.5 Rear Middle Left (RML) addresses the rear center sensor on the left side, *ref.* is the reference distance acquired by the laser, $diff = |ref - rml|$ The ultrasonic sensor has got a max. distance around 2.5 m and a minimum distance of 0.1 m. Clearly one can see false ultrasonic readings if the true distance is outside this borders. Within its operational borders there is a standard measurement error between 0.1 m and 0.2 m with an average update frequency of 7.69 Hz, which fits the manufactures specifications, see [81].

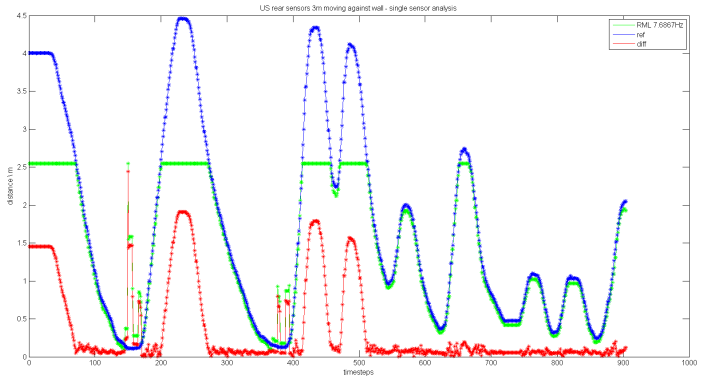


Figure 4.5: sensor test standard production ultrasonic sensor

Within Figure 4.6 RML addresses the rear center sensor on the left side, ref. is the reference distance acquired by the laser, $diff = |ref - rml|$ The ultrasonic sensor has got a max. distance around 4 m and a minimum distance of 0.15 m. Within its operational borders there is a standard measurement error between 0.1 m and 0.2 m (but with peaks up to 1.5 m) with an average update frequency of 15.1 Hz, which fits the manufactures specifications, see [86]. The High performance ultrasonic sensors are much more noisy than the standard production sensors (see Figure 4.5), but with a higher measuring distance and update frequency.

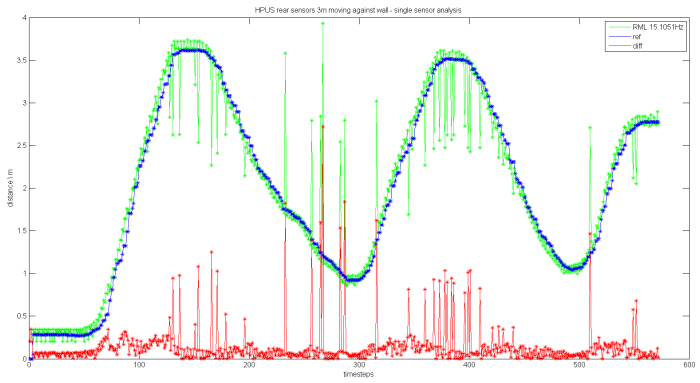


Figure 4.6: sensor test high performance ultrasonic sensor

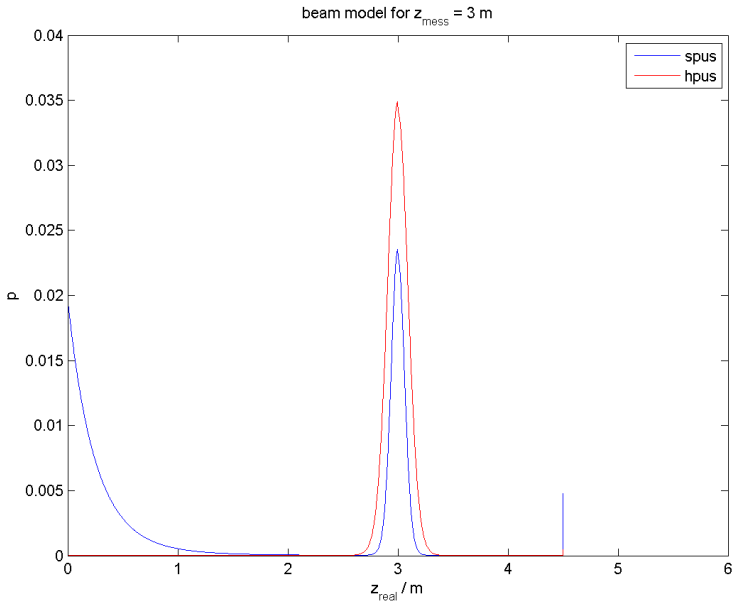


Figure 4.7: With the obtained beam model parameters a real distance of 3m can be model according to the plot.

parameter	standard ultra sonic sensor	high performance ultra sonic sensor
z_{hit}	0.2564	0.3008
z_{short}	0.1614	0.0211
z_{max}	0.1686	0.0763
z_{rand}	0.1245	0.1055
σ_{hit}	0.0992	0.1525
λ_{short}	1.5020	0.3559
$weighted_{average}$	0.7110	0.8223

Table 4.2: determined ultrasonic intrinsic parameters

4.4 results

The following section is separated into two parts, one for simulation one for real data. The simulation is used to determine the best parametrization for the particle filter. During the real data runs only the good parameters from the simulation are used. Both parts will present their results with tables (4.3 and 4.5) at the end of the subsection.

4.4.1 simulation

To determine, test and verify optimal parameters for the particle filter a simulation module has been written. The module uses an occupancy grid map, see Section 2.2.1, preloaded with an a priori CAD map to generate reference data. Any valide trajectory within the boarders of the map can be used as an input. A virtual vehicle will move along this trajectory and generates the following data:

- ground truth data
- odometry data
- ultrasonic measurement data

Section 4.4.1 will give an overview about possible simulation parameters. Simulated trajectories can be seen within Section 4.4.1.

parameters

The ground truth data will along the given trajectory without any noise. The main parameter for the temporal resolution is the step size, meaning movement per time base. Velocity and angular rotation will be calculated according this. Accuracy and update frequency can be adjusted. The simulation area is assumed to be plain, no change in height (e.g. a ramp) is simulated. The speed of the car is assumed to be low, therefor no occurrence of slip and drift is assumed.

The odometry data is based on the ground truth, but with the add of Gaussian distributed noise:

- σ_x $\cdots$ position error in x direction
- σ_y $\cdots$ position error in y direction
- σ_φ $\cdots$ orientation error
- σ_v $\cdots$ jitter for velocity
- σ_ϖ $\cdots$ jitter for angular velocity

In addition the simulated odometry data can be modeled with an increasing drift in x and y direction to simulate adding motion error via time. A constant adding drift is assumed, this error values will not decrease or compensate during one simulation run. Due to an assumed planar environment no environmental caused side effect the the sensors, e.g. changing drift when moving up a ramp,

is modeled.

To simulate ultrasonic data the CAD map is used. Every mapped object will generate an ultrasonic echo if within range of a sensor. The ground truth position of the virtual vehicle is used to calculate the true distance to the nearest object for every sensor. Random Gaussian noise (σ_{err}) will be add to this distance. To model device failures specific sensors can be deactivated during the whole simulation or random sensor reading will be suppressed temporary. The number of false readings per update cycle can be specified. Within the simulation the sensors physic nature is amused to behave according to the beam model, compare Section 3.2.1. During every update cycle the simulated response quality will be adjusted according to the beam models probability for the true distance.

H. Altinger NOTE: discuss about error parameters

H. Altinger NOTE: discussion (reference to sensors sensor models) about chosen values

G. Steinbauer NOTE: assumptions?

G. Steinbauer NOTE: any papers to simulation?

G. Steinbauer NOTE: use simplified beam model to simulate; –> Gaussian distribution already done; set various sensor measurements to random values (similar to pshort and pmax)

obtained trajectories

The following section will give an exemplary overview to obtained trajectories from simulation runs. . All data have been simulated within the environment shown in Figure 4.1. The same plots can be compared to real data, refer to Section 4.4.2. For figures 4.8 to 4.13 the following parameters have been used according to known mean and standard deviation by the odometry sensor, which has been determined by experiments, see [80].

- sensor model: beam model
- resampling: slot resample methode
- distribution: odo
- $\sigma_x = 0.001$ m
- $\sigma_y = 0.001$ m

- $\sigma_{\varphi} = 0.00873$ °
- $\sigma_{v} = 0.5$ $\frac{m}{s}$
- $\sigma_{\varpi} = 0.005$ $\frac{rad}{s}$
- $drift_x = 0.01$ $\frac{m}{s}$
- $drift_y = 0.01$ $\frac{m}{s}$
- blind sensors: one per update cycle

A complete list of parameters used during simulation runs can be found within Table 4.3

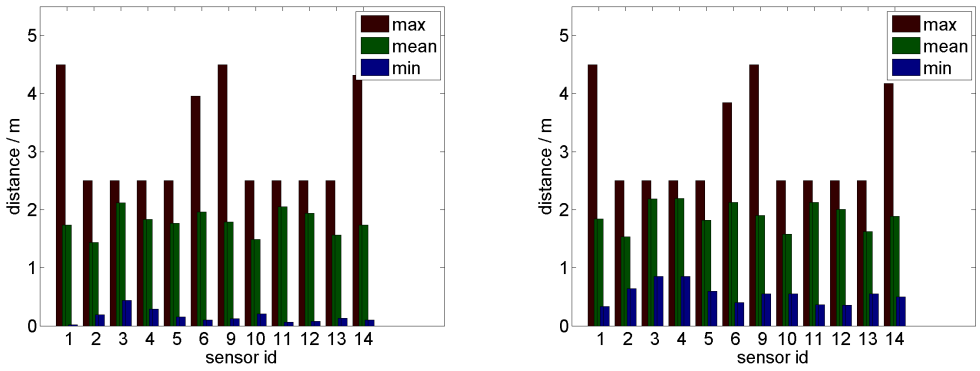
Comparing the ultrasonic sensor responses within Figure 4.8 between the obtained (simulated) measurement data (listed as real) and the true distances (listed as virtual). Clearly one can see higher distances and less readings at minimum range. This is caused by the simulation because similar behavior could be observed with real data. This has got a significant influence to the sensor model.

As described within Section 4.4.1 the simulated odometry has got a drift in x and y which can be seen within Figure 4.9 and compared against real data within Figure 4.15.

Figure 4.11 compares the resulting position from a Kalman merged odometry with the particle filters position to the ground truth, Figure 4.11 shows the same data as an overlay. Clearly one can see a good lateral position. The scene, as known from the blueprint (see Figure 4.1) shows a very narrow ramp which causes this good lateral, but unclear longitudinal position. There are too many good particles with similar readings, because only the side sensors can add measurement data to be weighted. By mischance a lot of particles will deliver such data. After the first curve the scene is unique enough to fix the position.

Figures 4.12 and 4.13 shows the position quality analysis. Two criteria are used, quadratic distances error and orientation error between the systems position and the ground truth at the same time. According to the explanation above the lateral position can not be determined which causes the high displacement error. After the first unique scene the quadratic distance error decreases down to less than 1m.

When analyzing the weight of every particle Figure 4.13 a random weight distribution over the whole particle set. Good particles will survive more than one resampling step, but will not live forever.



(a) (b)

Figure 4.8: Comparing sensor distance responses. The bars show the min, average and max reported distance from simulated data (real, subFigure 4.8(a)) and virtual (true distance, subFigure 4.8(b)) measurements during a simulation run.

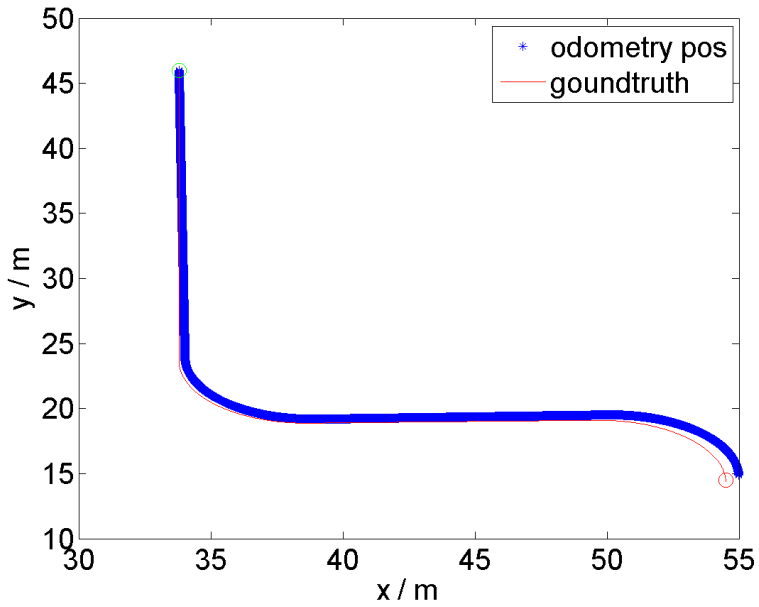


Figure 4.9: simulated odometry compared to ground truth during a simulation run

The resulting plain position from the particle filter can be seen within Figure 4.10. Clearly one can see a wide area of position hypothesis around the ground truth position due to uncertanties (mainly caused by the noisy sensor data) the particle filter has got. Using a Kalmanfilter to fusion odometry with this positon will result in a smaller band around the true position, compare Figure 4.11.

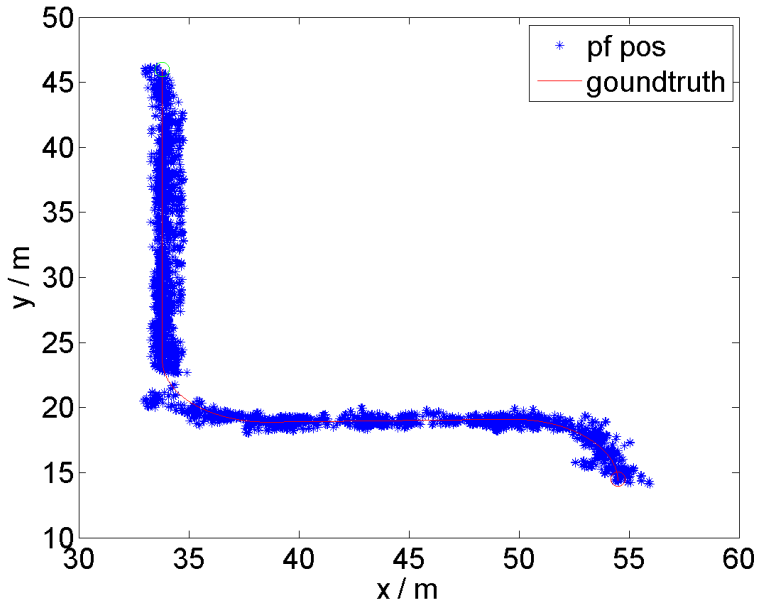


Figure 4.10: particle filter position during a simulation run

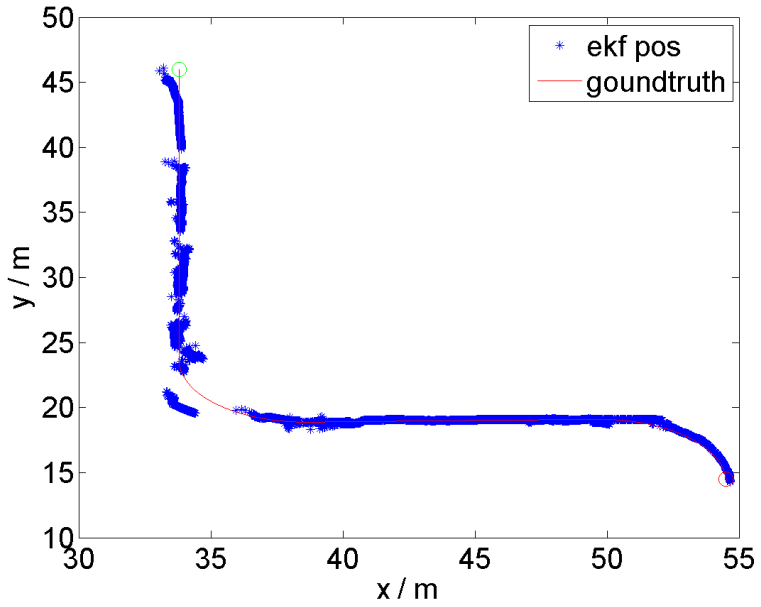
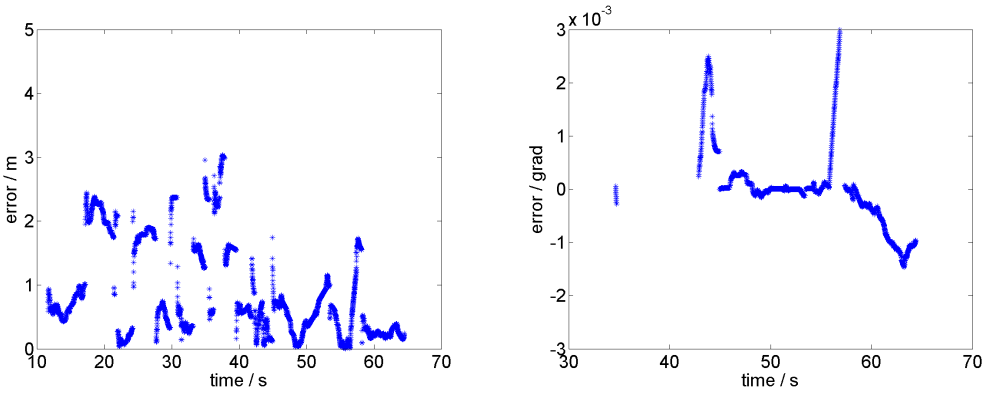


Figure 4.11: overlay particlefilter and odometry kalman vs. groundtruth - simulation



(a) (b)

Figure 4.12: distance and orientation error from particlefilter and odometry via kalman - simulation

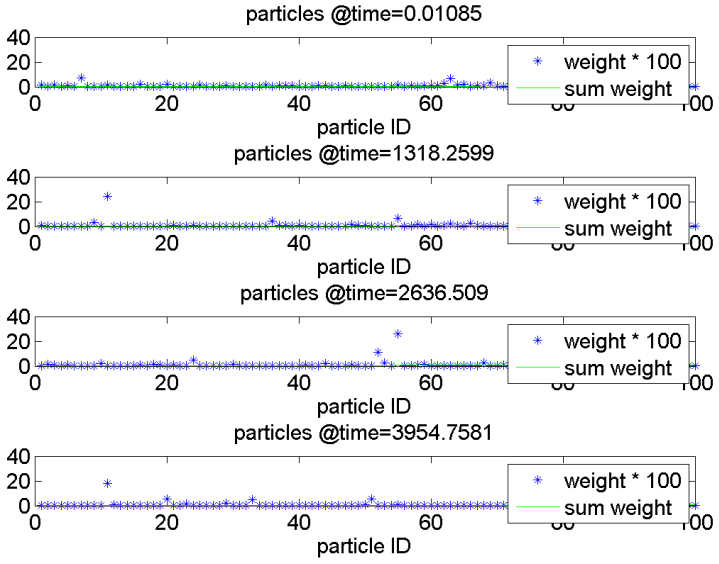


Figure 4.13: weightanalysis - simulation. For better visibility the value for weight and age have been scaled by 10 duo to the hight summed weight over fall particles. This weight has been extracted before the normalization step.

G. Steinbauer NOTE: add plot: simulation generates ground truth for odo & sonar measurements; vertical: bins for expected distances; horizontal: bins for real measurements; every row should contain similar results to beam models graph; create this plot for simulated data and real data (for real data use LMS as ground truth)

Table 4.3 shows a representation of used parameters and archived position accuracy during simulation runs. Comparing the various parameters one can see a decreasing position accuracy whit an increasing number of particles. The best results can be achieved with the slot resample strategy (see Section 3.2.4).

All data where aquisitated at the same simulated environment.

particles	e_{rr}^{-}			φ_{err}			simple Model	beam Model	slot resample	residual resample	global distribution	odometry distribution	ζ_x distribution [1]		ζ_y distribution		ζ_φ distribution [2]	
	min	mean	max	min	mean	max												
	m	m	m	°	°	°							m		m		°	
100	0	0.4791	3.0311	-0.1106	-0.0349	0.0044	✓	✗	✓	✗	✗	✓	0.2		0.2		0.175	
200	0	1.0534	7.2954	-0.1344	-0.0355	0.0041	✓	✗	✓	✗	✗	✓	0.2		0.2		0.175	
500	0	3.9383	44.2013	-0.1131	-0.0312	0.0763	✓	✗	✓	✗	✗	✓	0.2		0.2		0.175	
100	0	0.24	3.1616	-0.1151	-0.0333	0.0497	✗	✓	✓	✗	✗	✓	0.2		0.2		0.175	
200	0	1.511	9.3707	-0.1355	-0.0361	0.0462	✗	✓	✓	✗	✗	✓	0.2		0.2		0.175	
500	0	3.283	16.1638	-0.1166	-0.0368	0.0484	✗	✓	✓	✗	✗	✓	0.2		0.2		0.175	
100	0	5.8501	32.5439	-0.1178	-0.0351	0.0037	✓	✗	✓	✗	✓	✗	0.2		0.2		0.175	
200	0	6.1017	31.2621	-0.1121	-0.0347	0.0134	✓	✗	✓	✗	✓	✗	0.2		0.2		0.175	
500	0	6.9413	34.9366	-0.1612	-0.034	0.0742	✓	✗	✓	✗	✓	✗	0.2		0.2		0.175	

Table 4.3: particle-filter parameters and resulting performance from simulation data, part I, for part II see Table 4.4

1 ζ_x,ζ_y are set according to the grid cell size used, see Section 4.1.1

2 ζ_φ is set according to mean and standard deviation from odometry, see [80]

particles	e_{rr}^{-}			φ_{err}			simple Model	beam Model	slot resample	residual resample	global distribution	odometry distribution	ζ_x distribution [1]	ζ_y distribution	ζ_φ distribution [2]
	min	mean	max	min	mean	max									
	m	m	m	°	°	°							m	m	°
100	0	14.8455	58.3996	-0.0817	-0.0115	0.0614	✗	✓	✗	✓	✗	✓	0.2	0.2	0.175
200	0	12.5269	38.947	-0.1415	-0.0411	0.0103	✗	✓	✗	✓	✗	✓	0.2	0.2	0.175
500	0	11.2283	28.3315	-0.1527	-0.0305	0.0675	✗	✓	✗	✓	✗	✓	0.2	0.2	0.175
100	0	2.5276	29.5528	-0.1132	-0.0351	0.0039	✗	✓	✓	✗	✓	✗	0.2	0.2	0.175
200	0	1.4755	27.6012	-0.1132	-0.0358	0.0247	✗	✓	✓	✗	✓	✗	0.2	0.2	0.175
500	0	1.0042	35.4301	-0.1108	-0.0399	0.0006	✗	✓	✓	✗	✓	✗	0.2	0.2	0.175

Table 4.4: particle-filter parameters and resulting performance from simulation data, part II, for part I see Table 4.3

1 ζ_x,ζ_y are set according to the grid cell size used, see Section 4.1.1
2 ζ_φ is set according to mean and standard deviation from odometry, see [80]

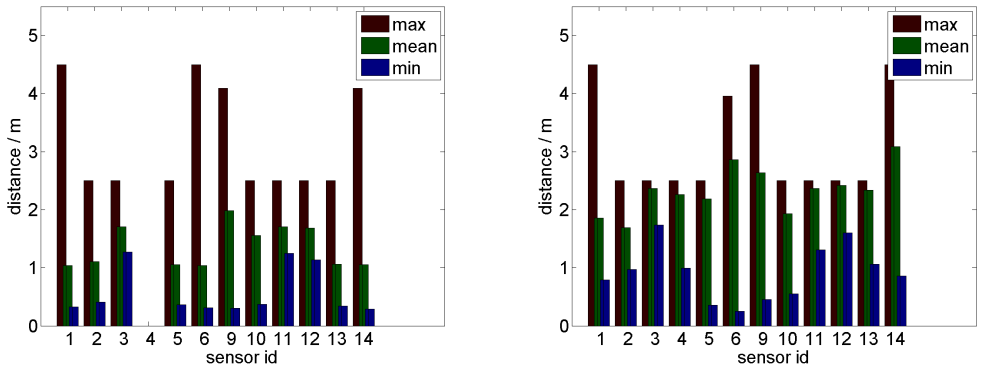
4.4.2 real environment

This section will show results from experiments with real data. The vehicle has been driven manually through the environment presented within Figure 4.1. Odometry and ultrasonic measurement data are on-board. Ground-truth data is generated off-board, see Section 4.2. Figures 4.14 to 4.19 are exemplary plots from one run. The used parameters can be seen within Table 4.5, last entry for 2.000 particles. Table 4.5 shows resulting performance data with good parameters known from simulation, compare Section 4.4.1.

Comparing real ultrasonic data with true distance data, see Figure 4.14, one can see typical problems. e.g. one sensor is dirty and does not report any data. The mean reported distance (from all measurements during the whole run) is short due to short readings, misreadings, etc. The same causes the difference within the minimum distances between real and true distances. This behavior need to be considered within the sensor model, compare Section 3.2.1. As known from Table 4.3 an enhanced sensor model delivers a higher position accuracy.

Figure 4.15 shows an overlay between real odometry data and ground truth position. Clearly one can see an error in rotation which causes a quadratic position error of up to $2.1m$. Using the particle filter with a Kalman filter and the odometry Figure 4.16 can be achieved. The fusioned data (as described within Section 3.1 the data represent a Kalman fusion between odometry and particle filter position) is slightly better (mean quadratic error of $0.228m$ compared to $0.72m$, max. quadratic error of $1.435m$ compared to $1.8712m$) than odometry only. Quadratic error values within Table 4.5 are based on data from Figure 4.18. In contrast Figure 4.17 shows the same data only from odometry.

Analyzing the particle weights one can see within Figure 4.19 some dominant particles. This fits the implemented slot resample strategy.



(a) (b)

Figure 4.14: Comparing sensor distance responses. The bars show the min, average and max reported distance from real data (b) and virtual (true distance (b)) measurements during on run.

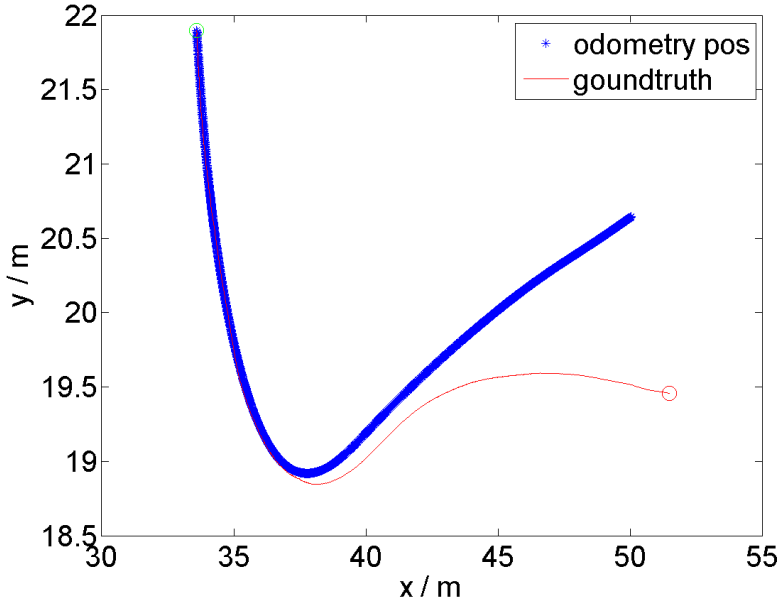


Figure 4.15: real odometry compared to ground truth during one run

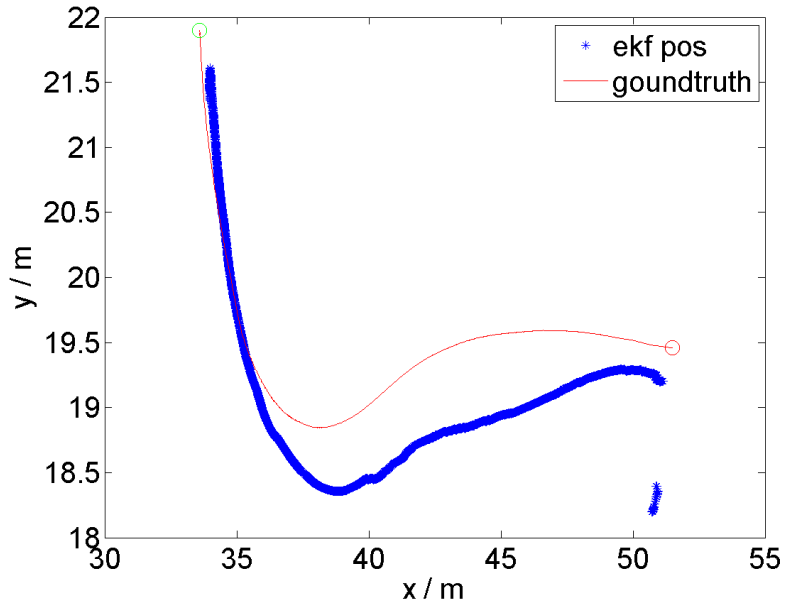
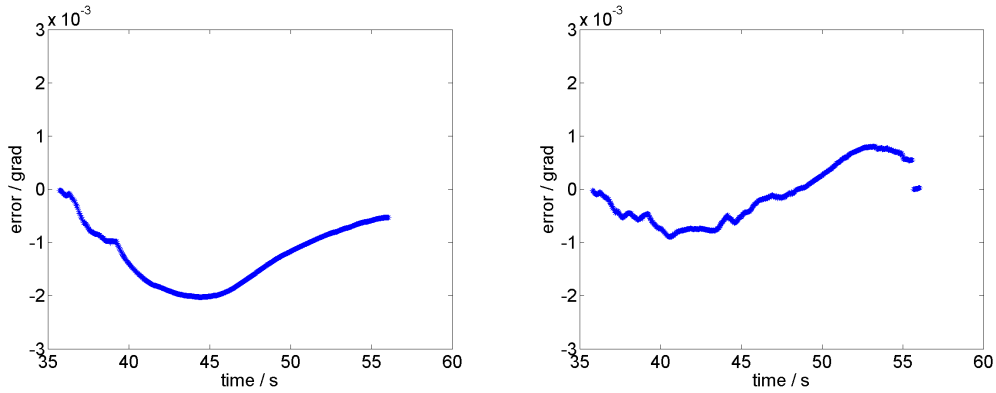
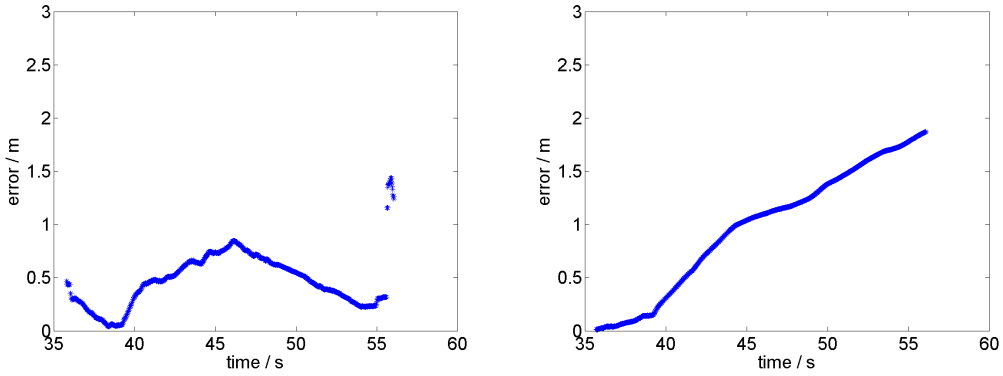


Figure 4.16: Kalman filtered position from odometry and particle filter position during one run



(a) (b)

Figure 4.17: distance and orientation error from particlefilter and odometry via kalman - real data



(a) (b)

Figure 4.18: distance and orientation error from particlefilter and odometry via kalman - real data

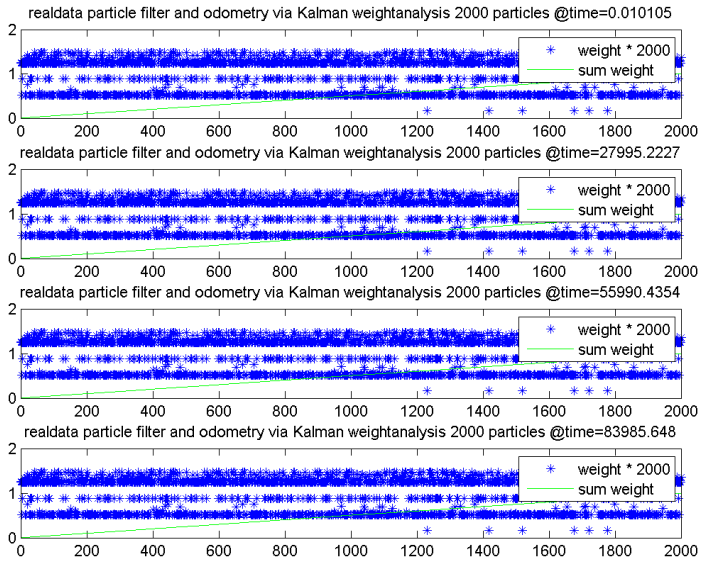


Figure 4.19: weightanalysis - real data. For better visibility the value for weight has been scaled by 2000 due to the normalized weight.

Analyzing Table 4.5 one can see an increasing accuracy (quadratic position error) with a higher number of particles. All data within this table where acquired using the last good position as center for resampling. This method can be used to enhance odometry position. In combination with the slot resample method and the beam model this is known from simulation to achieve the best results, compare Section 4.4.1.

particles	e_{rr}				φ_{err}			simple Model	beam Model	slot resample	residual resample	global distribution	odo distribution	ζ_x distribution [1]	ζ_y distribution	ζ_φ distribution [2]
	min	mean	max		min	mean	max									
	m	m	m		°	°	°							m	m	°
100	0	0.2973	6.4718		-0.0518	-0.0009	0.0533	✗	✓	✓	✗	✗	✓	0.2	0.2	0.175
200	0	0.2559	3.2056		-0.0009	0.0006	0.0345	✗	✓	✓	✗	✗	✓	0.2	0.2	0.175
500	0	0.1846	2.114		-0.0319	0	0.0014	✗	✓	✓	✗	✗	✓	0.2	0.2	0.175
1000	0	0.1884	1.45		-0.0207	0	0.0014	✗	✓	✓	✗	✗	✓	0.2	0.2	0.175
2000	0	0.2279	1.435		-0.0009	0	0.0008	✗	✓	✓	✗	✗	✓	0.2	0.2	0.175

Table 4.5: particlefilter parameters and resulting performance from real data

1 ζ_x,ζ_y are set according to the grid cell size used, see Section 4.1.1

2 ζ_φ is set according to mean and standard deviation from odometry, see [80]

5 Conclusion

Knowing from literature, see [20], sonar sensors can be used to navigate within mapped environments. Later literature ([2] and [25]) suggest particle filter to solve a global localization problem, even within unmapped environments when using SLAM. Starting with results from Table 4.3 it is possible to use sonar sensors in combination with a particle filter to calculate a position hypothesis. The system described within Section 3 is able calculate such a position hypothesis based on odometry and sonar distance readings.

Comparing results from simulation, see Section 4.4.1, with real world data, see Section 4.4.2, the real hypothesis is not as accurate and reliable as the simulated. Partially this result is caused by the nature of ultrasonic sensors, like low spatial resolution, high noise and false readings. Using a simple or an enhanced sensor model like the beam model has got a big impact on the resulting position accuracy, compare Table 4.3.

The used sensor set, as described within Section 4.1.2, has got too short maximum reading distances and too wide horizontal opening. This results within a low spatial resolution and a short sensing distances. As a result it is only possible to generate position hypothesis within small environments. Small is hereby defined as enough obstacles, e.g. walls or other mapped objects, within the sensor range. In such a case the sensor is always able to read a distance which can be used to gauge position hypothesis (the particles).

6 further work

The enhancement when moving from the simple sensor model to the ”Beam Models of Range Finders”, as described within [2], is promising. Further work should be done do refine this model. In addition a way to calibrate the sonar sensors to prevent false readings and reduce the noisy readings would be interesting. Another approach would be to determine the sensor models parameters for every single sensor and use it to calculate the position hypothesis probability.

The sensor model can be enhanced to better approximate the physical propagation of the ultrasonic sound wave and the resulting sensitivity area of the sensor.

Precisely looking at Figure 4.6 one can see a positive distance error when moving toward an obstacle, and a negative distance error when moving away. May this is caused by the Doppler Effect, which can be handled by a specialized filter. A adjustable band-pass filter can be used to reduce the distance outliers. This can lead to an enhanced sensor preprocessing system.

The sensor set itself can be changed. If using more sensors with smaller horizontal opening and longer maximum distance readings the position hypothesis can be improved. The High Performance Ultrasonic Sensor (HPUS) shows less false readings than the Standard Production Ultrasonic Sensor (SPUS) and delivers a better reading quality. As seen within Figure 4.7 the sensor can be more accurate.

To reduce false position hypothesis in case of multi hypothesis a weighted and history based combination of good particles can be used instead of the implemented average sum.

A total different approach could be to use location based informations. When moving along the long narrow ramp, compare Figure 4.1, the particle filter delivers a good lateral position, but a bad longitudinal position. Embedding this information into the map an enhanced sensor fusion filter could adopt its state transitions and updates.

Bibliography

[1] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Particle filters for mobile robot localization." *Proc. of the Sixteenth National Conference on Artificial Intelligence*, pp. 470–498, 1999.

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

[3] WHO, "Global health risks," *WHO Library Cataloguing-in-Publication Data*, 2009.

[4] EUSTAT, "Road safety evolution in the eu," 2010.

[5] S. Austria, "Straßenverkehrsunfälle 2010," *Statistik Austria*, 2011.

[6] Gleitsmann, "Verkehrsbericht 2010," *Polizeipräsidium Osthesse*, 2010.

[7] various, "Special issue: Special issue on the darpa grand challenge, part 1," *Journal of Field Robotics*, vol. 23, no. 8, pp. 461 – 652, 2006.

[8] ——, "Special issue: Special issue on the darpa grand challenge, part 2," *Journal of Field Robotics*, vol. 23, no. 8, pp. 655 – 835, 2006.

[9] ——, "Special issue on the 2007 darpa urban challenge, part i," *Journal of Field Robotics*, vol. 25, no. 8, 2008.

[10] ——, "Special issue on the 2007 darpa urban challenge, part ii," *Journal of Field Robotics*, vol. 25, no. 9, 2008.

[11] ——, "Special issue on the 2007 darpa urban challenge, part iii," *Journal of Field Robotics*, vol. 25, no. 10, 2008.

[12] K. Mak, "Advanced driver assistance systems: Assessing opportunities and challenges," *Strategy Analytics*, 2007.

[13] E. commission for europe, "Convention on road traffic," *United Nations*, 1968.

[14] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y.-W. Seo, S. Singh, J. Snider, A. Stentz, W. R. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson, "Autonomous driving in urban environments: Boss

and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425 – 466, 2008.

[15] K. OHNO, T. TSUBOUCHI, B. SHIGEMATSU, and S. YUTA, "Differential gps and odometry-based outdoor navigation of a mobile robot," *Advanced Robotics*, vol. 18, no. 6, pp. 611–635, 2004.

[16] P. Waldmann and D. Diehues, "der bmw tracktrainer - automatisiertes fahren im grenzbereich auf der nürnburgring nordschleife," *TÜV SÜD Tagungen*, 2013.

[17] J. Funke, P. Theodosis, R. Hindiyeh, G. Stanek, K. Kritatakirana, C. Gerdes, D. L. M., Hernandez, B. Muller-Bessler, B., and Huhnke, "Up to the limits: Autonomous audi tts," in *Intelligent Vehicles Symposium (IV), 2012 IEEE*, june 2012, pp. 541 –547.

[18] Berlin, *DIN 70000: Road vehicles: Veicle dynamics and road-holding ability.*, 1994.

[19] J. Bloomenthal and J. Rokne, "Homogenouse corrdinates," The University of Calagary, Tech. Rep., 2003.

[20] A. Elfes, "Sonar-based real-world mapping and navigation," *Robotics and Automation, IEEE Journal of*, vol. 3, no. 3, pp. 249 –265, june 1987.

[21] S. Thrun, "Learning occupancy grid maps with forward sensor models," *Autonomous Robots*, vol. 15, pp. 111–127, 2003.

[22] D. Fox, J. Hightower, D. S. L. Liao, and G. Borriello, "Bayesian filters for location estimation," *IEEE Pervasive Compting*, vol. 03, pp. 24–33, 2003.

[23] R. Kalman, "A new approach to linear filtering and prediction problmes," *Transactions of the ASME Journal of Basic Engineering*, 1960.

[24] *A Comparison between Extended Kalman Filtering and Sequential Monte Carlo Techniques for Simultaneous Localisation and Map-building.* ARAA, 2002.

[25] S. Thrun, "Particle filters in robotics," in *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI).* UAI 2002, 2002, pp. 511–518.

[26] F. Gustafsson, "Particle filter theory and practice with positioning applications," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 25, no. 7, pp. 53 –82, july 2010.

[27] D. Crisan and A. Doucet, "A survey of convergence results on particle filtering methods for practitioners," *Signal Processing, IEEE Transactions on*, vol. 50, no. 3, pp. 736 –746, mar 2002.

[28] A. Doucet, N. de Fretias, and N. Gordon, *Sequential Monte Carlo Methods in Practice.* Springer, 2001.

[29] F. Daum and J. Huang, "Curse of dimensionality and particle filters," in *Aerospace Conference, 2003. Proceedings. 2003 IEEE*, vol. 4, 3 2003, pp. 1979 – 1993.

[30] S. Lenser and M. Veloso, "Sensor resetting localization for poorly modelled mobile robots," in *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA*. IEEE, 2000, pp. 1225 – 1232.

[31] D. Fox, "Kld-sampling: Adaptive particle filters," in *In Advances in Neural Information Processing Systems 14*. MIT Press, 2001, pp. 713–720.

[32] N. G. M. Sanjeev Arulampalam, Simon Maskell and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *Transactions on Signal processing*, vol. 50, pp. 174–188, February 2002.

[33] S. Julier and J. Uhlmann, "A new extension of the kalman filter to nonlinear systems," in *Proceedings of SPIE*, ser. 3, vol. 3, 1997, pp. 182 – 193.

[34] D. Blackwell, "Conditional expectation and unbiased sequential estimation," *The Annals of Mathematical Statistics*, vol. 18, pp. 105–110, 1947.

[35] M. H. D. Groot, *Probability and Statistics*, ser. 3rd edition. Addison-Wesley, 2001.

[36] T. Dean and K. Kanazawa, "A model for reasoning about persistence and causation," Rown University, Tech. Rep., 1989.

[37] W. Shu and Z. Zheng, "Performance analysis of kalman-based filters and particle filters for non-linear non-gaussian bayesian tracking," *Proceedings of the 16th IFAC World Congress, 2005*, 2005.

[38] K. M. A. Doucet, N. Freitags and S. Russell, "Rao-blackwellised particle filtering for dynamic bayesian networks," in *In Conference on Uncertainty in Artificial Intelligence (2000)*, 2000, pp. 176 – 183.

[39] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Janssonm R. Karlsson and P.J. Nordlund, "Particle filters for positioning, navigation, and tracking," *Signal Processing, IEEE Transactions on*, vol. 50, no. 2, pp. 425 –437, feb 2002.

[40] C. S. G. Grisettiyz and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, 2005, pp. 2432 – 2437.

[41] C. S. G. Grisetti and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, pp. 34–46, 2006.

[42] D. F. J.s. Gutmann, W. Burgard and K. Konoliege, "An experimental comparison of localization methods," in *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, vol. 2. IEEE, 1998, pp. 736–743.

[43] J. Gutmann and D. Fox, "An experimental comparison of localization methods continued," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 1. Digital Creatures Lab., Sony Corp., Tokyo, Japan, 2002, pp. 454–459.

[44] M. Bugallo and P. Djuric, "Complex systems and particle filtering," in *Signals, Systems and Computers, 2008 42nd Asilomar Conference on*, oct. 2008, pp. 1183 – 1187.

[45] O. C. R. Douc and E. Moulines, "Comparison of resampling schemes for particle filtering," in *Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis (2005)*, 2005, pp. 64 – 69.

[46] T. Bailey, J. Nieto, J. Guivant, M. Stevens, M. and E. Nebot, "Consistency of the ekf-slam algorithm," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, 2006, pp. 3562 –3568.

[47] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics*. Springer, 2008.

[48] S. Konatowski and A. Pieniezny, "A comparision of estimtaion accuracy by the use of kf, ekf and ukf filters," *Computational Methods and Experimental Measurements*, vol. XIII, pp. 779 – 789, 2007.

[49] S. Haykin, *Kalman filtering and neural networks*, ser. chapter 7. John Wiley and Sons, 2001.

[50] K. L. B. Hoffman-Wellenhof and M. Wieser, *Principles of Positioning and Guidance*. Springer, Wien, 2003.

[51] G. Inc., *Garmin Wide Area Augmentation System*, 2012.

[52] T. Inc., *How GPS Works*, 2012.

[53] M. H. T. Luettel and H. Wuensche, "Autonomous Ground Vehicles – Concepts and a Path to the Future," *Proceedings of the IEEE*, 2012.

[54] L. N.-S. A. Suppe and A. Steinfeld, "Semi-autonomous virtual valet parking," in *AutomotiveUI*. ACM, 2010, pp. 139–145.

[55] F. S. J.M. Wille and M. Maurer, "Stadtpilot: Driving autonomously on braunschweig's inner ring road," in *Intelligent Vehicles Symposium (IV), 2010 IEEE*, june 2010, pp. 506 –511.

[56] D. Marioli, C. Narduzzi, C. Offelli, D. Petri, E. Sardini and A.Taroni, "Digital time of flight measurement for ultrasonic sensor," *IEEE Transactions of instrumentation and measurement*, vol. 41, pp. 93–97, 1992.

[57] J. Gutmann, "Robuste navigation autonomer mobiler systeme," Ph.D. dissertation, Universität Freiburg, 2000.

[58] J. S. W. Adiprawita, A.S. Ahmad and B. Trilaksono, "New resampling algorithm for particle filter localization for mobile robot with 3 ultrasonic

sonar sensor," in *In Proceedings of the 2011 internationla converence on electrical engineering and informatics*, 2011, pp. 1–6.

[59] R. Kümmerle, D. Hähnel, D. Dolgov, S. Thrun and W. Burgard, "Autonomous driving in a multi-level parking structure," in *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA-09)*, 2009, pp. 3395–3400.

[60] *Kognitive Fahrzeuge der UniBWM: Von VaMoRs zu MuCAR-3.* Munic Network, 2008.

[61] A. S. A. Schanz and K. Kuhnert, "Autonomous parking in subterranean garages-a look at the position estimation," in *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE*, june 2003, pp. 253 – 258.

[62] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Stanley: The robot that won the darpa grand challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661 – 692, 2006.

[63] B. M.-B. G. Stanek, D. Langer and B. Huhnke, "Junior 3: A test platform for advanced driver assistance systems," in *Intelligent Vehicles Symposium (IV), 2010 IEEE.* IEEE, 2010, pp. 143–149.

[64] S. A. Waldkirchen, *LMS200/211/221/291 Laser Measurement Systems*, 2006.

[65] I. Velodyne Acoustics, *HDL-64E users Manual*, 03 2008.

[66] S. Thrun and M. Montemerlo, "The GraphSLAM algorithm with applications to large-scale mapping of urban structures," *International Journal on Robotics Research*, vol. 25, no. 5/6, pp. 403–430, 2005.

[67] D. Langer and C. Thorpe, "Range sensor based outdoor vehicle navigation, collision avoidance and parallel parking," *Autonomous Robots*, vol. 2, pp. 147–161, 1995.

[68] M. Baer and M.E. Bouzouraa and C.Demiral and U. Hofmann and S.Gies and K. Diepold, "Egomaster: A central ego motion estimation for driver assist systems," in *Control and Automation, 2009. ICCA 2009. IEEE International Conference on.* IEEE, 2009, pp. 1708–1715.

[69] C. Chen and M. Rickert, *Automatic Parking Garage - ADTF Path-planning Filter*, FORTISS, 2011.

[70] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: a survey," *Network, IEEE*, vol. 18, no. 4, pp. 45 – 50, july-aug. 2004.

[71] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25 – 30, 1965.

[72] H.-H. Braess and U. Seifert, *Handbuch Kraftfahrzeugtechnik*. Vieweg, 2000.

[73] S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems*. Artech House Boston London, 1999.

[74] R. Schubert, E. Richter, and G. Wanielik, "Comparison and evaluation of advanced motion models for vehicle tracking," in *Information Fusion, 2008 11th International Conference on*, IEEE. 2008 IEEE, 2008, pp. 730–735.

[75] H. C. U. Scheunert and G. Wanielik, "Precise vehicle localization using multiple sensors and natural landmarks," in *Proceedings of the Seventh International Conference on Information Fusion*. International Society of Information Fusion, 2004, pp. 649–656.

[76] A. Bychkov and V. Suslonov, "Maggi's equations in terms of quasi-coordinates," *REGULAR AND CHAOTIC DYNAMICS*, vol. 7, no. 3, pp. 269–279, 2002.

[77] E. N. A.B. Bychakov, C. Cattani and M. Yushkov, "The simplest model of the turning movement of a car with its possible sideslip," *Technische Mechanik*, vol. 1, no. 29, pp. 1–12, 2007.

[78] I. Rekleitis, "A particle filter tutorial for mobiel robot localization," Centre for Intelligent machines, McGill University, Tech. Rep., 2004.

[79] A. AG, *Audi A7/S7 datasheet*, 2010.

[80] C. Demiral, "Fahrversuche im pgn mit c7(a7)," Audi AG, EF-56, Tech. Rep., 2012.

[81] A. AG, *4H0 919 275 - Ultrasonic Sensor, personal communication*, 2011.

[82] S. A. Waldkirchen, *Laser Measurement Systems of the LMS500 Product Family*, 2010.

[83] A. Ibisch, S. Stümper, H. Altinger, M. Neuhausen, M.-P. Tschentscher, M. Schlipsing, J. Salmen, and A. Knoll, "Autonomous driving in a parking garage: Vehicle-localization and tracking using environment-embedded lidar sensors," 2013, contributed to IV 2013.

[84] A. Ibisch, *OutsideIn LIDAR ground truth system*, Institut für Neuroinformatik, Ruhr-Universität Bochum, 2012.

[85] A. AG, *Parking Pilote - reproduceable parking accuracy during automated driving, personal communication*, 2012.

[86] V. AG, *High performance ultrasonic sensors - mark B2, personal communication*, 2010.

[87] Elektrobit, *ADTF advertising folder*, 10 2011.

[88] T. M. R. P. Toolkit, "Mrptk," 2012. [Online]. Available: http://www.mrpt.org/Particle_Filters

[89] R. O. System, "Ros," 2012. [Online]. Available: http://www.ros.org

List of Figures

List of Tables

A ADTF - Environment

Figure A.1: The Logo of the used ADTF 2.8.1

The ADTF is a commercial time triggered message passing framework. It features hardware access such as Bus-systems for modern cars. It is possible to record data at any point to be reused with a replay feature. The playback speed can be adjusted to simulate realtime behavior even when processing time is high.

The vendor, see [87], describes it the follows: "ADTF is able to capture asynchronous data from different sensor sources and provides standard components for data recording and interpretation of LIN, MOST, CAN and FlexRay bus systems."

It supports a simple interface to program so called filters to connect with the message passing system. Every filter will be triggered if new data arrives, can handle its own data procession methods and pass or transmit new data. The framework ensures time synchronous execution of developed modules nevertheless if working on-line or off-line.

The framework comes with an C++ Application Programming Interface (API) and runs on Windows or Linux.
Within the automotive development of driver assistance systems ADTF can be seen as an industrial standard supported by all mayor European car manufactures and suppliers.

B acronyms

ADTF Automotive Data and Time-triggered Framework

AMCL Adaptive Monte Carlo Localization

API Application Programming Interface

CAD Computer Aided Design

CAN Controller Area Network

CA Constant Acceleration

CTR Constant Turn Rate

CTRA Constant Turn Rate and Acceleration

CTRV Constant Turn Rate and Velocity

CV Constant Velocity

DARPA Defense Advanced Research Projects Agency

DAQ Data AcQuisition

DGPS Differential Global Positioning System

DIN Deutsche Industrie Norm, engl. german industrial norm

DOF Degree Of Freedom

DSP Digital Signal Processor

EgoMaster Egomotion Master, see [68]

EKF Exended Kalman Filter

ESC Electronic Stability Control

ESS Effective Sample Size

EUROSTAT Statistical Office of the European Communities

GIS Geographic Information System

GPS Global Positioning System

HPUS High Performance Ultrasonic Sensor

ICC Instantaneous Centre of Curvature

IMU Inertial Measurement Unit

INS Inertial Measurement System

KLD Kullback-Leibler distance

LIDAR LIght Detection And Ranging

LTR Linear Time Resampling

MCL Monte Carlo Localization

MRPT Mobile Robot Programming Toolkit, see [88]

PLA Park Lenk Assistent

PDF Probability Density Function

PF Particle Filter

RBS Reference Broadcast Synchronization

RANSAC RANdom Sample Consensus

RBPF Rao Blackwellized Particle Filter

RFID Radio Frequency IDentification

RML Rear Middle Left

ROS Robot Operating System [89]

SARA Sensor Array Audi

SIR Sample Importance Resampling

SLAM Simultaneous Localization And Mapping

SPUS Standard Production Ultrasonic Sensor

SRL Sensor Resetting Localization

SWR Select With Replacement

UKF Unscented Kalman Filter

WAAS Wide Area Augmentation System

WHO World Health Organization

Thanks