Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Bachelor Thesis

Multi-modal Localization using Wi-Fi Signal Strength and 2D Range Finder

vorgelegt von

Benjamin Scholz

10.11.2016

Universität Hamburg

Fakultät für Mathematik, Informatik und Naturwissenschaften

Fachbereich Informatik

Studiengang: Bachelor of Science Informatik

Matrikelnummer: 6428073

Erstgutachter: Dr. Norman Hendrich

Zweitgutachter: Lasse Einig

Abstract

English:

Localization is an important problem in robotics, that is needed for many navigation tasks. A sensor that is often used for this problem is the 2D laser range finder. It usually results in a precise localization as long as the robot's pose is roughly known at the start. Is this not the case the problem is specifically called global localization. Especially in rooms with a symmetric layout or rooms without any unique characteristics, a global localization using only a 2D laser range finder fails in many cases. Our approach here is to use Wi-Fi signal strength to infer an approximate position estimation. In this thesis we use Gaussian process regression with pre-recorded signal strength data of the environment in order to realize a Wi-Fi position estimation. The Wi-Fi position estimation is used to give a Monte Carlo localization with a 2D laser range finder an approximate initial position at the start in order to improve the global localization in buildings. The result is a higher success rate for the global localization when using the Wi-Fi position estimation compared to the approach only using the laser range finder.

Deutsch:

Die Standortbestimmung ist ein wichtiges Problem in der Robotik. Sie wird für eine Vielzahl von Navigationsaufgaben benötigt. 2D Laserentfernungsmesser sind beliebte Sensoren für diese Aufgabe. In den meisten Fällen führt dies zu einer präzisen Lokalisation, zumindest wenn die Position des Roboters beim Start bekannt ist. Wenn dies nicht der Fall ist, spricht man auch von einer globalen Lokalisation. Insbesondere in Räumen mit symmetrischem Strukturen oder Räumen ohne auffallende Eigenschaften, führt eine globale Lokalisation, die nur einen 2D Laserentfernungsmesser nutzt, zum Misserfolg. Unser Ansatz ist in diesen Fällen Wi-Fi-Signalstärken zu verwenden, um daraus eine Position zu approximieren. Mit Hilfe von Gaußprozessregression setzen wir eine Wi-Fi Positionsschätzung um. Hierfür werden vorher Signalstärken in der Umgebung aufgenommen. Diese Wi-Fi-Positionsschätzung wird dann zusammen mit einer Monte Carlo Lokalisation, die einen 2D Laserentfernungsmesser verwendet, benutzt um die globale Lokalisation in Gebäuden zu verbessern. Das Resultat ist eine höhere Erfolgsrate für die globale Lokalisation, im Vergleich zum Ansatz der nur den 2D Laserentfernungsmesser verwendet.

Contents

Abbreviations

AMCL	**A**daptive **M**onte **C**arlo **L**ocalization
BFGS	**B**royden **F**letcher **G**oldfarb **S**hanno
CSV	**C**omma **S**eperated **V**alues
GPS	**G**lobal **P**ositioning **S**ystem
L-BFGS	**L**imited-memory **B**royden **F**letcher **G**oldfarb **S**hanno
LIDAR	**Li**ght **D**etection **A**nd **R**anging
MAC-address	**M**edia **A**ccess **C**ontrol-address
RGB-D	**R**ed **G**reen **B**lue-**D**epth
ROS	**R**obot **O**perating **S**ystem
Rprop	**R**esilient back**prop**agation
SSID	**S**ervice **S**et **Id**entifier
Wi-Fi	**Wi**reless **Fi**delity

Glossary

BFGS An optimization algorithm, that can be used to find minima or maxima of functions. It uses both first order- and second order-derivative information, while not explicitly computing the second order-derivative.

control data Can be provided by the commands given to a robot, so for example its velocity. Alternatively often provided by sensors like odometers. Control data gives us information about the change of state of a robot.

environment measurement data Provided by sensors like laser range finders, cameras or sonars. The environment measurement data gives us information about the current state of the environment of a robot.

Gaussian process A collection of random variables, any finite number of which have a joint Gaussian distribution [Rasmussen and Williams, 2005, p. 13].

global localization Global localization describes localization without any prior information about the current position. That means a robot would have to figure out its current position at the start of the localization.

GPS A global positioning system that is using satellites. It can be used to determine the current location with an error of a few meters.

gradient descent An algorithm that uses first-order derivatives in order to find a local minimum of a function.

hyperparameter A hyperparameter is a parameter that can be learned. There is no need to set it manually.

kernel When used with Gaussian processes, kernels are also referred to as covariance function. They are used to measure the difference between two inputs. They can lift the inputs into a higher space, while the computations still happen in the input space. Thus one can save computation time by applying kernels. They can be used whenever an operation can be represented by a dot product.

kidnapped robot problem Describes the problem when a robot is taken and brought to a different location while a localization is running. The problem is that the robot is now localized in the wrong position and thus the localization fails.

L-BFGS The limited-memory variant of the BFGS algorithm.

laser range finder Sensor that uses laser beams to detect the distance of obstacles.

LIDAR Short for "Light Detection and Ranging". It is a type of sensor, that is using light in order to measure the distance to obstacles in the environment. It can be used to create 2D or 3D representations of areas.

log-likelihood It is the natural logarithm of a likelihood function. In some cases they are easier to work with than the original likelihood function. The logarithm is monotonically increasing and has the maximum at the same point as the likelihood function. Often the partial derivatives of the log-likelihood function is easier to compute than the partial derivatives of the likelihood function. Thus it is attractive to use in order to optimize hyperparameters by searching the maximum of a function.

MAC-address Short for media access control-address. This address is a commonly used identifier in networks. So for example a Wi-Fi receiver has its own MAC-address as does the access point. The MAC-address is unique for each interface.

Monte Carlo localization Robot self-localization using probability. Its basis is the recursive state estimation implemented with a particle filter.

odometer An odometer is a common sensor used in order to measure the distance traveled.

particle Particles are generally a small part of larger system. Here particles are used to refer to samples that represent parts of a state belief.

posterior A posterior is also called posterior probability distribution. It is a distribution created from the prior distribution after taking further evidence into account that wasn't considered when creating the prior.

prior A prior is also called prior probability distribution. It is a distribution that was created by making certain assumptions before taking further evidence into account. When the distribution is transformed by taking that further evidence into account, it is called posterior, or posterior distribution.

recursive state estimation Using incoming data a current state is computed recursively, by using the previous state. The state is represented by a probability density function.

regression Inferring the relationship between different variables from available scalars from said variables. In case of the example of the linear regression the result is a linear function. In the example of Gaussian process regression the result is a Gaussian process.

RGB-D sensor A sensor that is able to sense colors and depth.

Rprop Short for resilient backpropagation. Using first-order derivatives the algorithm can be used to find a local minimum. It only uses the sign of resulting partial derivatives when searching for the minimum.

SSID Short for service set identifier. It is used by Wi-Fi access points. The identifier is not unique. It is used to signal different access point that belong to the same Wi-Fi network.

training set A set of data that was recorded in order to discover the relationship between different variables. To discover those relationships regression can be used.

Chapter 1

Introduction

1.1 Motivation

Localization is an important problem in robotics. Even simple navigation tasks in households require a mobile robot to be able to determine its own position in the environment. It needs to be able to move from point A to point B. Being able to do this, a robot could for example bring a fresh cup of coffee and return the used cup back to the kitchen. It can help with cleaning tasks [Pinheiro et al., 2015]. It can also be highly important in industrial environments. Hamburg has one of the world's largest and busiest ports. In order to bring the shipping containers from one place to the other autonomous transport vehicles can be used. These are just a few of many real world examples, where localization is used.

Another modern example of using localization is the autonomous vehicle. The Global Positioning System (GPS) [Misra and Enge, 2011] can give us a good idea about the position of the car. But GPS is too inaccurate for the high demands of autonomous vehicles. Levinson et al. [2007] for example used Light Detection and Ranging (LIDAR) [Weitkamp, 2005] in addition to the GPS to vastly improve the localizations accuracy. The LIDAR is a vision based sensor that is similar to a radar. Instead of radio waves it uses light beams. It can be used to map out nearby obstacles. The LIDAR is not well suited to estimate the position on a large map with no prior information, so the GPS can be used to provide a rough position estimate. On the other hand as already explained the GPS is inaccurate, so once a rough position is estimated, the LIDAR can help to narrow the position down.

The 2D laser range finder is a specific kind of LIDAR, that is often used for localization. With the laser range finder the robot is able to detect the distance of obstacles near itself. There are many situations where a robot can not rely on GPS. In many buildings its signal is obstructed. Localizing without any prior information about the robot's position is called global localization. There are existing approaches to do this with just a vision-based sensor like the LIDAR or a 2D laser range finder. In order to localize a robot we use a map of the robot's

environment that was created beforehand. Usually only a small part of the map is taken into account, because we have a rough idea about the position. At the start of the global localization using a laser range finder the whole map has to be taken into account, until after some time a possible position was determined. But this method has the downside that it is often unreliable and leads to localization failures that are hard to recover from. This is especially true for environments that don't have many unique characteristics. When localizing with a laser range finder one usually creates a map with the existing obstacles and the outline of the room. Often many parts of these maps look very similar. The fewer distinctive features the map has, the more likely it is that the global localization with a laser range finder fails. Many places indoors can cause those problems. Many rooms are symmetric, so it can be hard to differentiate between the walls. Many buildings have rooms with the same outline, like for example many office-environments. Office environments can also be quite large, and the larger the space that has to be taken into account the more likely it is, that a global localization only using a laser range finder will fail. So this can cause problems especially in buildings.

In most buildings there are a lot of different Wi-Fi [IEEE, 2012] networks. The networks MAC-address provides a unique identifier for each signal. The signal strengths can be a good indicator about how far the wireless access point is away. And while one signal alone wouldn't be enough to infer the position, multiple signals from different access points can be used to do just that.

This could be used to enhance the localization with laser scanners. When at the start no position estimation is given, it could be computed with the Wi-Fi data. Also in case of a localization failure the Wi-Fi signals could be used to infer the correct position.

This combines the strengths of both the position estimation with Wi-Fi signals and the localization with a laser scanner. The Wi-Fi signals have a unique identifier and will only be observed in a certain range on a map. The outline of a wall could be nearly identical in different places, but overall the localization with a laser scanner is very accurate once the true position was determined.

1.2 Related Work

One of the most popular methods for localization is the Monte Carlo localization. We are going to use this method, both for localization via a 2D range finder, and parts of it also to estimate the position via Wi-Fi. Thrun et al. [2005] give an extensive overview over the Monte Carlo localization.

In order for the Monte Carlo localization to work, one needs a measurement model for the sensor that is used. Thrun et al. [2005] provide models that can be used for a 2D range finder. In order to estimate the position with a Wi-Fi receiver, one needs to implement a different measurement model.

There have been different approaches to solving the localization problem with Wi-Fi signals. Serrano et al. [2004] used a propagation model to estimate the signal strength at a given point on the map. The advantage of this technique is that only the positions of the access points have to be known. But it is a lot more inaccurate than other approaches. The problem is that inside a building the Wi-Fi signal is often obstructed by obstacles, like walls, furniture or humans. This makes the propagation model very inaccurate and thus also leads to inaccurate position estimations.

A different approach to the problem uses a map that was created beforehand. Biswas and Veloso [2010] recorded Wi-Fi signals a in grid like fashion, recording in a certain interval. In order to interpolate the Wi-Fi signal from the recordings, linear interpolation was used. The drawback here is that the Wi-Fi signals have to be recorded at certain positions on the map. This makes the process of creating the map more complicated and time consuming.

In order to create something akin to a map from recorded Wi-Fi data, using regression proved to be useful. As it turns out Gaussian processses are well suited for this kind of problem. Rasmussen and Williams [2005] give an extensive overview of how they can be used in machine learning.

Ferris et al. [2006] use Gaussian processes for building the map and for localization. This means the signal strength can be recorded at random places. Thus the needed data for creating the map can be recorded while the robot drives around or even does other tasks. In our approach we want to determine whether such a method can be used for global localization in an office environment with a 2D laser range finder and a wheeled robot. There are already existing investigations into a global localization with Wi-Fi data, but in different contexts. Duvallet and Tews [2008] used it in an industrial environment to enhance the previously developed laser beacon localization [Tews et al., 2007]. The most striking difference here is the environment, which is large and spacious compared to the often cramped spaces used as offices. Also the number of Wi-Fi signals that can be observed in office spaces is usually a lot higher than in industrial environments. Ito et al. [2014] used Wi-Fi for global localization for an iPhone 4 with an additional RGB-D sensor, a sensor is able to produce color- and depth-images. For the RGB-D sensor they used a map created from the floor-plan, that is less accurate then the maps usually used. Here an obvious disparity to our test environment is the platform used. While the phone is used in conjunction with an RGB-D sensor, which is comparable to a 2D laser range finder, the phone does not have any wheels that can provide odometry. Here they resort to an alternative for the odometry method using the RGB-D sensor.

1.3 Chapter Overview

In Chapter 2 we will explain the basics of probability based localization, with the so called Monte Carlo localization [Dellaert et al., 1999] as an example. Furthermore

we will explore how a Wi-Fi receiver can be used with the Monte Carlo localization and explain how Gaussian process regression [Rasmussen and Williams, 2005] can be used for this purpose. Chapter 3 details the implementation of a Wi-Fi position estimation using the knowledge from Chapter 2 about Monte Carlo localization and Gaussian process regression. In Chapter 4 experiments and results regarding the usage of the Wi-Fi position estimation will be discussed. In chapter 5 we will reflect the results and come to a conclusion. Lastly chapter 6 provides an outlook for future applications and possibilities.

Chapter 2

Basics

This chapter summarizes the basics for robot self-localization. At first in section 2.1 we will give an example of data produced by a laser range finder. Next we will show how uncertainty can complicate the task of localization and how probability can be helpful to approach this problem. We will introduce the recursive state estimation and building on this move on to an implementation of it using a particle filter. Afterwards we will show how said particle filter can be used for robot self-localization with the Monte Carlo localization. In the next section we will explain the problem of global localization and then introduce the kidnapped robot problem and a possible solution.

In section 2.2 we will look into the aspects of Wi-Fi that are important for the Wi-Fi position estimation.

Section 2.3 will revolve around the Wi-Fi sensor model, that can be used with the Monte Carlo localization. We will define what a Gaussian process is and derive it from a Bayesian treatment of the linear regression.

2.1 Localization

2.1.1 Localization with a Laser Range Finder

The goal of localization is that a mobile robot is able to determine its own position. For this purpose we use a map of the environment of the robot. In our case the robot used is a TurtleBot 2 (http://www.turtlebot.com, accessed on 02.11.2016) shown in figure 2.1. This particular robot uses wheels to move around and in order get information about the state of its environment it uses a Kinect camera [Andersen et al., 2015] and a 2D laser range finder. The robot also has sensors that provide us with information about its movement. An odometer is used to measure the distance traveled and a gyro to measure the angular velocity. Now using the information provided by the various sensors the objective is to infer the location of the robot.

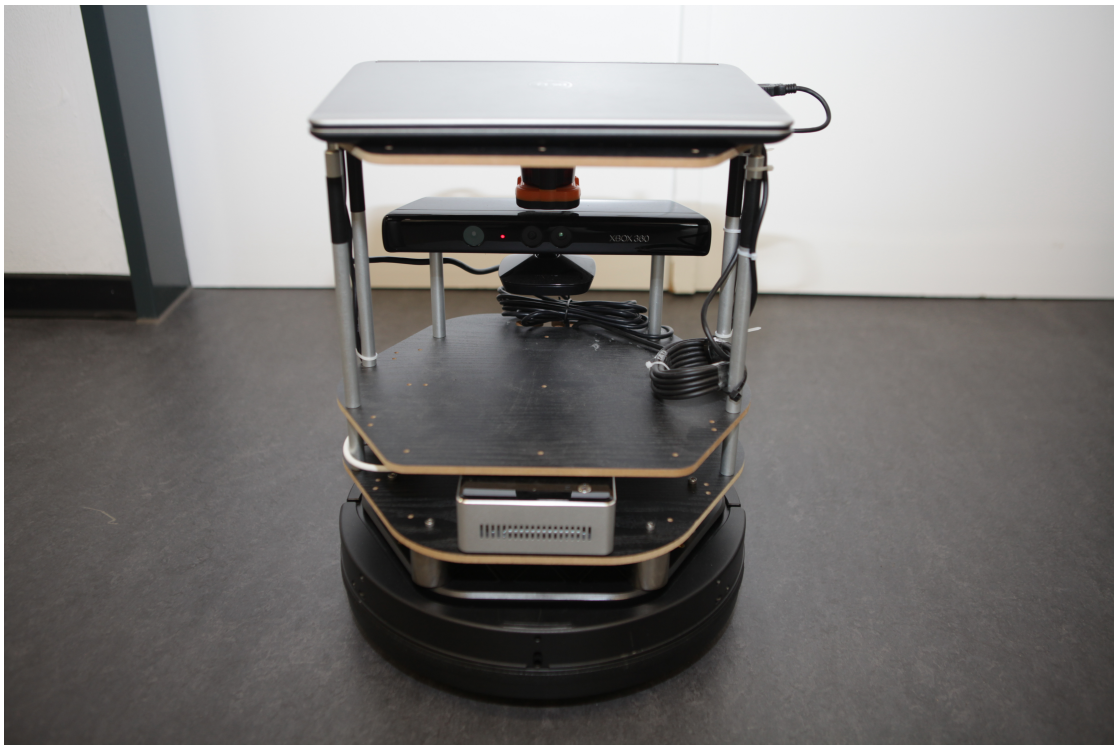

FIGURE 2.1: The TurtleBot used for testing the software and carrying out the experiments in chapter 4.

The 2D laser range finder is a popular choice for this kind of task. The prices for these kind of sensors are reasonable. It produces reliable data. It gives us the distance to obstructions in the robot's environment. The laser range finder uses laser beams. The beams get reflected by the obstructions back to the laser range finder. The range finder then measures the time it takes for the beams to come back and uses this information in order to infer the distance of the objects in its vicinity. The further away the object, the longer it takes for the beam to get reflected back to the laser range finder.

It also has advantages over cameras for this particular task. Cameras produce pictures, which give us a lot more complex information to work with. The distance to objects can be interpreted and processed very easily. In pictures we would have to filter for the desired information.

Figure 2.2 shows the kind of data a laser range finder produces. Here the black circle is the robot. The red squares show us the distance to obstacles in the robot's environment. Here the robot is localized on a map, so the position of the robot on the map mirrors its position in the real world. This map is an obstacle map, so it shows where we would expect obstacles in the environment. The map is also divided into small grid cells, that can be either occupied, free or unknown. Here black means occupied, light gray means free and dark gray means unknown. As one can see the laser range finder data mostly matches our expectations. The obstacles found by the laser range finder match the obstacles shown on the map,

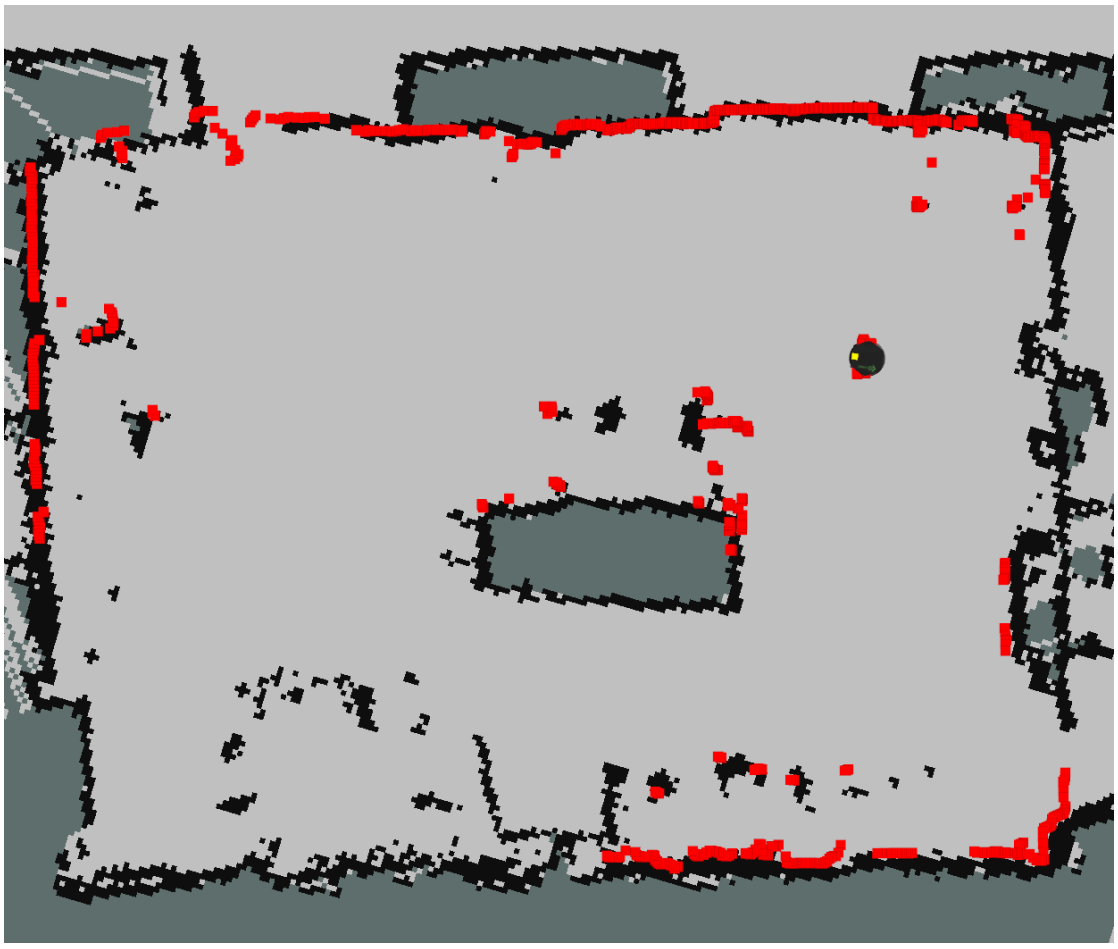

FIGURE 2.2: Example of localization with a laser range finder. The black circle is the robot. The robot is on an obstacle map that was created beforehand, so the other black lines signify obstacles. The red squares are the laser range finder data. As we can see the red squares mostly fit to the obstacles.

so it is likely that the robot is located on the correct position on the map. So sensors like the laser range finder can be used to check if the robot is localized in the correct position by comparing the produced data with the expectations based on the map. Sensors like odometers and gyros can be used to move the robot accordingly on the map. So when the robot drives a few meter forward, the same should happen on the map.

As laid out the robot has different kinds of sensors that provide us with information. Our goal is to infer the position of the robot on a map of the environment using that data. But for a variety of reasons we can not be 100% certain about the data sensors produce.

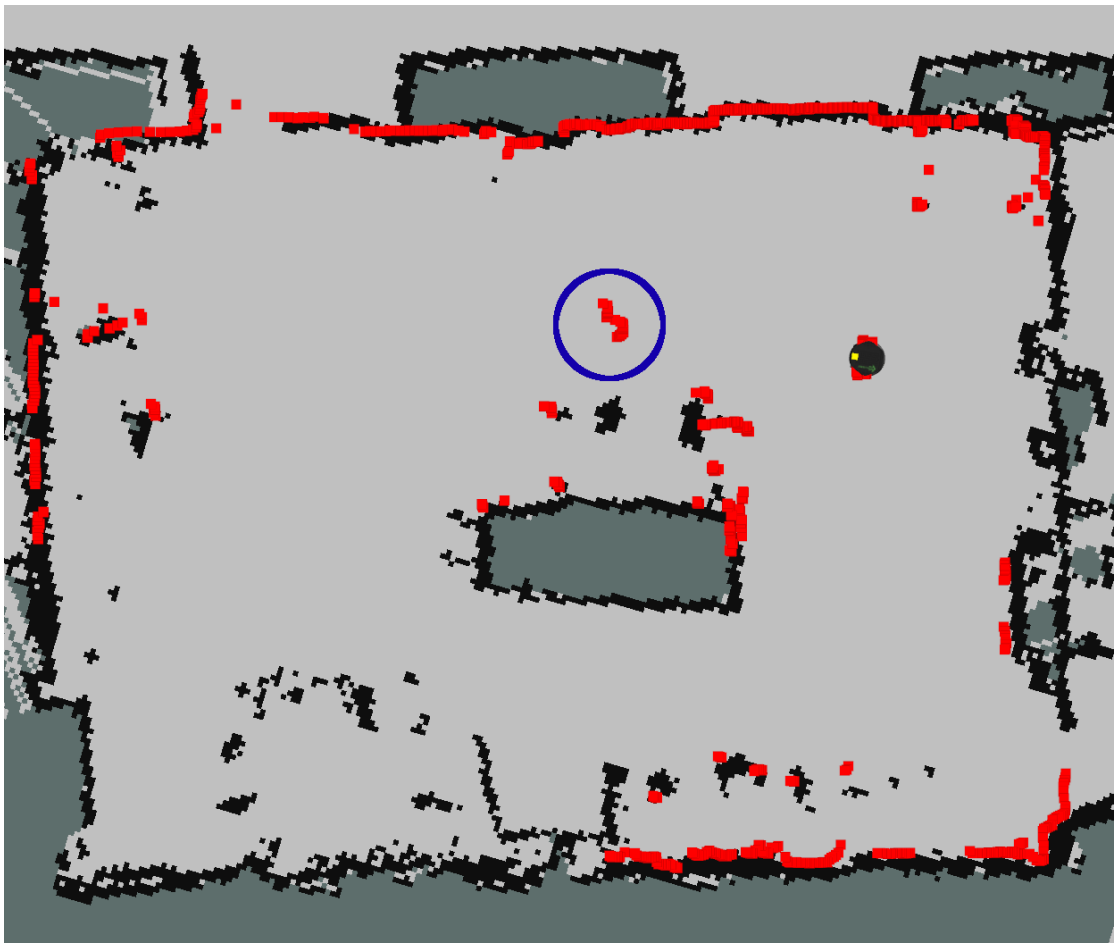

FIGURE 2.3: This is an example of how uncertainty plays a role in localization. The figure is very similar to figure 2.2, but signified by the blue circle there are measurements created by the laser range finder that do not fit to the map. The reason in this case was simply that a human was standing there.

2.1.2 Uncertainty

Uncertainty plays a big role in robotics. By dealing with the real world this inherently introduces unpredictability. The robot's environment is dynamic and can change constantly. Even the robot itself changes its own environment by acting in it. See for example figure 2.3. It is similar to figure 2.2, but here measurements appear that don't fit the expectation we would have given the map the robot is located on. The robot is still located on the right position that mirrors its position in the real world. What changed is the environment. Someone was standing in the room and the data produced by the laser range finder mirrors that circumstance. A similar example would be when furniture is moved to a different place. Suddenly the data will not match in a particular place. It would be annoying when every small change in the environment would break our localization. Robots usually have a variety of sensors. They are used to get information about the state of their environment. The sensors used in robotics have certain limitations. They can only provide an inexact interpretation of the world around them and this is

something we need to keep in mind. When an odometer tracks a robots path it will never be exact. When a laser scanner measures the distance to an obstacle we have to expect that there will be an error margin. The sensors have to deal with physical limitations when trying to capture the state of their environment. Often they are subject to noise. In localization 2D maps get used very often. They are a model of the real world. But once again it is only an approximate model that strips away a lot of information from the real world and thus introduces more uncertainty [Thrun et al., 2005, p. 3-4].

Probability can be a helpful tool to circumvent this problem. When localizing instead of trying to find the pose of the robot directly we test how likely it is that the robot is in different hypothesized poses. So, using all the information we get from the sensors we compute which poses are the most likely ones [Thrun et al., 2005, p. 5]. In the following section we will show an algorithm that uses probability to deal with the uncertainty in our sensor data.

2.1.3 Recursive State Estimation

Thrun et al. [2005] divide the data usually collected by robots into two different categories:

1. Environment measurement data, denoted $z_{t_1:t_2}$ for data from time t_1 to time t_2
2. Control data, denoted $u_{t_1:t_2}$ for data from time t_1 to time t_2

The first one gives us information about the current state of the environment, provided by sensors like cameras, laser range scanners or Wi-Fi receivers. The second kind of data gives us information about the change of state. These are values that are given to the robot to execute some action, so for example a velocity. Sensors like odometers, a sensor that measure the distance traveled, could be classified as producing environment measurement data, but in practice this kind of data is often used as control data [Thrun et al., 2005, p. 22-23].

The current time step is denoted as t, so the current state is denoted x_t. At this point we will rely on probability for the reasons already explained. The probability of state x_t is the following: $p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t})$. Here all control data up to time step t is taken into account and all environment measurement data up to the previous time step $t-1$ is taken into account. But if we assume that state x_t is a sufficient summary of everything that happened before time step t, then we can omit most data:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{2.1}$$

The probability that a certain measurement was observed can be described by $p(z_t|x_{0:t}, z_{1:t-1}, u_{1:t})$. Once again we can omit most data:

$$p(z_t|x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t|x_t) \tag{2.2}$$

Equation 2.1 is known as the state transition probability and equation 2.2 is known as the measurement probability. The state transition probability takes the control data into account. As we will see, for localization this means it is used to model the movement of the robot. The measurement probability on the other hand takes the measurement data into account. For localization this means it is used to reflect the information the robot has about its environment at the moment [Thrun et al., 2005, p. 24-25].

The state of a robot is represented by so called belief distributions.

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \tag{2.3}$$

Equation 2.3 is often referred to as prediction. It doesn't take the last environment measurement into account.

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{2.4}$$

The posterior from equation 2.4 on the other hand does take the last measurement into account [Thrun et al., 2005, p. 25-26]. So $\overline{bel}(x_t)$ can be seen as the prior, that is computed with incomplete information and $bel(x_t)$ is the corresponding posterior, that is created by also using the last measurement data. As can be seen in algorithm 1 the posterior from equation 2.4 is computed by using the prediction from equation 2.3.

Once again, we don't want to take all past data from all time steps into account. So we are going to simplify both equations for our purposes. $bel(x_t)$ can be simplified by using Bayes rule and equation 2.2:

$$\begin{aligned} p(x_t|z_{1:t}, u_{1:t}) &= \frac{p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1:t})} \\ &= \eta p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t}) \\ bel(x_t) &= \eta p(z_t|x_t)\overline{bel}(x_t) \end{aligned} \tag{2.5}$$

Here η is a normalizer, stemming from the fact that $p(z_t|z_{1:t-1}, u_{1:t})$ is independent of the state x_t and so no matter what value x_t takes on in $p(x_t|z_{1:t}, u_{1:t})$, it does not influence η.

$\overline{bel}(x_t)$ can be simplified for our purposes, using the theorem of total probability and equation 2.1 [Thrun et al., 2005, p. 31-33].

$$\begin{aligned} p(x_t|z_{1:t-1}, u_{1:t}) &= \int p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t})p(x_{t-1}|z_{1:t-1}, u_{1:t})dx_{t-1} \\ &= \int p(x_t|u_t, x_{t-1})p(x_{t-1}|z_{1:t-1}, u_{1:t-1})dx_{t-1} \\ \overline{bel}(x_t) &= \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx_{t-1} \end{aligned} \tag{2.6}$$

In the second step in equation 2.6 we make use of the fact that u_t is not needed to infer the probability of state x_{t-1} and therefore simply omit it. 2.6 and 2.5 are used in the Bayes filter algorithm.

Algorithm 1 Bayes_filter [Thrun et al., 2005, p. 27]

1: **procedure** BAYES_FILTER($bel(x_{t-1}), u_t, z_t$)
2: **for** all x_t **do**
3: $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx_{t-1}$
4: $bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)$
5: **end for**
6: **return** $bel(x_t)$
7: **end procedure**

The algorithm gets the distribution of belief $bel(x_{t-1})$ from the last time step, the new environment data z_t and control data u_t from this time step t. The belief gets updated. At first with the control data, to make sure that the change of state gets reflected in the distribution before we apply our knowledge from the environment data to our distribution. In line 4 the belief gets updated with the measurement data too. Now all available data was used to create the new belief distribution. The belief distribution can be used to infer the state x_t.

Figure 2.4 shows a simple example how the algorithm could be used for localization. It shows a robot that is able to detect if it stands in front of a door. It reflects the nature of the control data and measurement data. Whenever the robot moved and the control data is used to update the belief, the belief still spikes in the correct position, but the spike becomes weaker. We have to take into account that the odometer could give us imprecise measurements. This is the reason why we use the measurement data too. Even if we had the correct position in the beginning, when we only use control data, after driving around for some time the localization will be off by a huge margin. The measurement data is there to correct this. If we look at both control data and measurement data in a certain time interval the uncertainty induced by moving around is small enough that we can compensate for it by using the measurement data.

The figure also provides a good example as to why we use probabilities. In figure 2.4 (b) the robot senses a door. There are three different doors, but the robot can't be sure which door it is, so it simply applies the same probabilities to all three doors instead of randomly choosing one. Then it also can't be sure about the exact position in front of these doors because the data could be imprecise. So in front of each door the distribution has bell-shaped spikes to reflect this.

Another good example why probabilities work so well can be observed in (c). The robot moved and of course the control data is also imprecise. So the spikes move according to the control data, but they get weaker and more spread out.

Now (d) shows a nice example how the data from the previous time steps get taken into account. The spikes move according to the control data. Now the

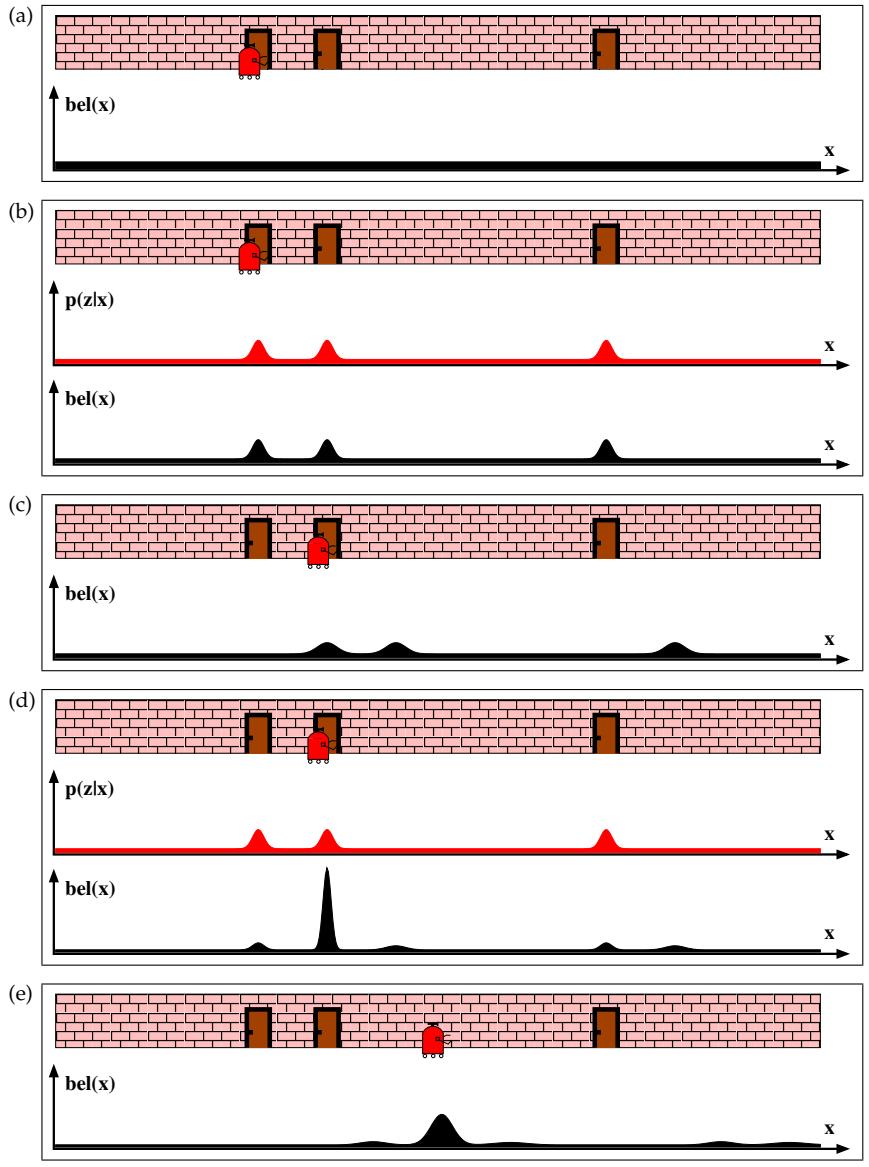

FIGURE 2.4: A simple example to explain how the Bayes filter works. In (a) the belief is evenly distributed. In (b) the robot observes the door and the measurement data reflects this and updates $bel(x)$ accordingly. In (c) the robot moves to the right and the beliefs structure shifts to the right as well, but the peaks become lower, because the control data could be imprecise. In (d) the robot takes the new measurement data into account. The belief clearly spikes at the correct position. In (e) the robot moved again. The spike is still in the correct position, but it is lower [Thrun et al., 2005, p. 6].

robot observes a door once again, just like in (b). But because the belief has only a spike in front of the correct door, the new belief has a high and distinct spike in front of the correct door after taking the measurement data into account.

2.1.4 Particle Filter

The particle filter is a non-parametric solution to implement the Bayes filter. Here the posterior distribution $bel(x_t)$ is represented by finitely many samples [Thrun et al., 2005, p. 85].

Specifically it is represented by a set of particles, that is denoted as:

$$X_t = x_t^{[1]}, x_t^{[2]}, ..., x_t^{[M]} \tag{2.7}$$

Each particle represents a possible state hypothesis at time t. M is the number of particles [Thrun et al., 2005, p. 96-97].

Algorithm 2 Particle_filter [Thrun et al., 2005, p. 98]

```
1:  procedure PARTICLE_FILTER(X_{t-1}, u_t, z_t)
2:      X̄_t = X_t = ∅
3:      for m = 1 to M do
4:          sample x_t^[m] ~ p(x_t | u_t, x_{t-1}^[m])
5:          w_t^[m] = p(z_t | x_t^[m])
6:          X̄_t = X̄_t + ⟨x_t^[m], w_t^[m]⟩
7:      end for
8:      for m = 1 to M do
9:          draw i with probability ∝ w_t^[i]
10:         add x_t^[i] to X_t
11:     end for
12:     return X_t
13: end procedure
```

So the belief $bel(x_t)$ is approximated by particle set X_t. The probability for a state hypothesis to be included is ideally proportional to its Bayes filter posterior $bel(x_t)$ [Thrun et al., 2005, p. 98].

$$x_t^{[m]} \sim p(x_t | z_{1:t}, u_{1:t}) \tag{2.8}$$

Just like in the Bayes filter the posterior X_t is computed recursively from the set X_{t-1}.

On line 4 of algorithm 2 the equivalent to $\overline{bel}(x_t)$ is computed. The state $x_t^{[m]}$ is generated based on particle $x_{t-1}^{[m]}$ and control data u_t. Based on $p(x_t | u_t, x_{t-1}^{[m]})$ new particles are sampled. This is done for every particle in the set X_{t-1}. In this step the most recent information from the control data gets added.

Line 5 is then the equivalent to computing $bel(x_t)$. $w_t^{[m]}$ is the weight of particle m at time t. The higher the weight the more likely it is that the state of the particle is representative of the real state.

On line 9 new particles get drawn. This happens according to the weights w_t. The higher the weight, the higher the chance of the particle to be drawn. The particles that were unlikely to represent the real state won't get drawn and only the ones that have a high weight survive.

2.1.5 Monte Carlo Localization

The particle filter can be used for localization [Thrun et al., 2005, p. 252]. Figure 2.5 shows a simple example. This figure is similar to figure 2.4. But here the particles and their weights are used to represent the belief distribution. In (a), (c) and (e) only the distribution of the particles is shown, but not their weights. It shows how after some iterations the particles are clustered around the positions that are most likely the real position. But this also involves the disadvantage that after some iterations the particles will only be clustered around one spot, so we only observe this particular part of the whole belief. This is both an advantage and a disadvantage. It means we save computing power, because we only compute a small part of the whole belief distribution. But it also means that after some iterations we will only observe one particular point of the belief and disregard the rest. In certain situations this can lead to localization failures, which can't be resolved unless we tweak the algorithm. How this happens and how those situations can be salvaged will be explained in the following section 2.1.7.

There is control data that is usually provided by an odometer. The measurement data is usually provided by sensors like laser range finders. In our specific case the Wi-Fi receiver will be another source of measurement data.

A sample motion model and a measurement model are needed for the Monte Carlo localization to work. The sample motion model takes u_t and $x_{t-1}^{[m]}$ as input. It takes the odometer's data into account and samples new particles based on that. In simpler terms this just means that the particles are moved according to what we can infer from the control data available. If a robot drove a few meters to the right the particles should move to the right as well. This step does not take the measurement data into account yet.

That happens in the next step. The measurement model takes the newly sampled particle $x_t^{[m]}$ and measurement data z_t and computes a weight. The higher the weight, the more likely is it according to the model that the particle is representative of the real pose.

These models are different for different kind of sensors. Later on we will show the used measurement model for the Wi-Fi receiver in-depth.

Algorithm 3 is the Monte Carlo localization. Note how similar it is to the particle filter from algorithm 2. The only difference is how the new set of particles is

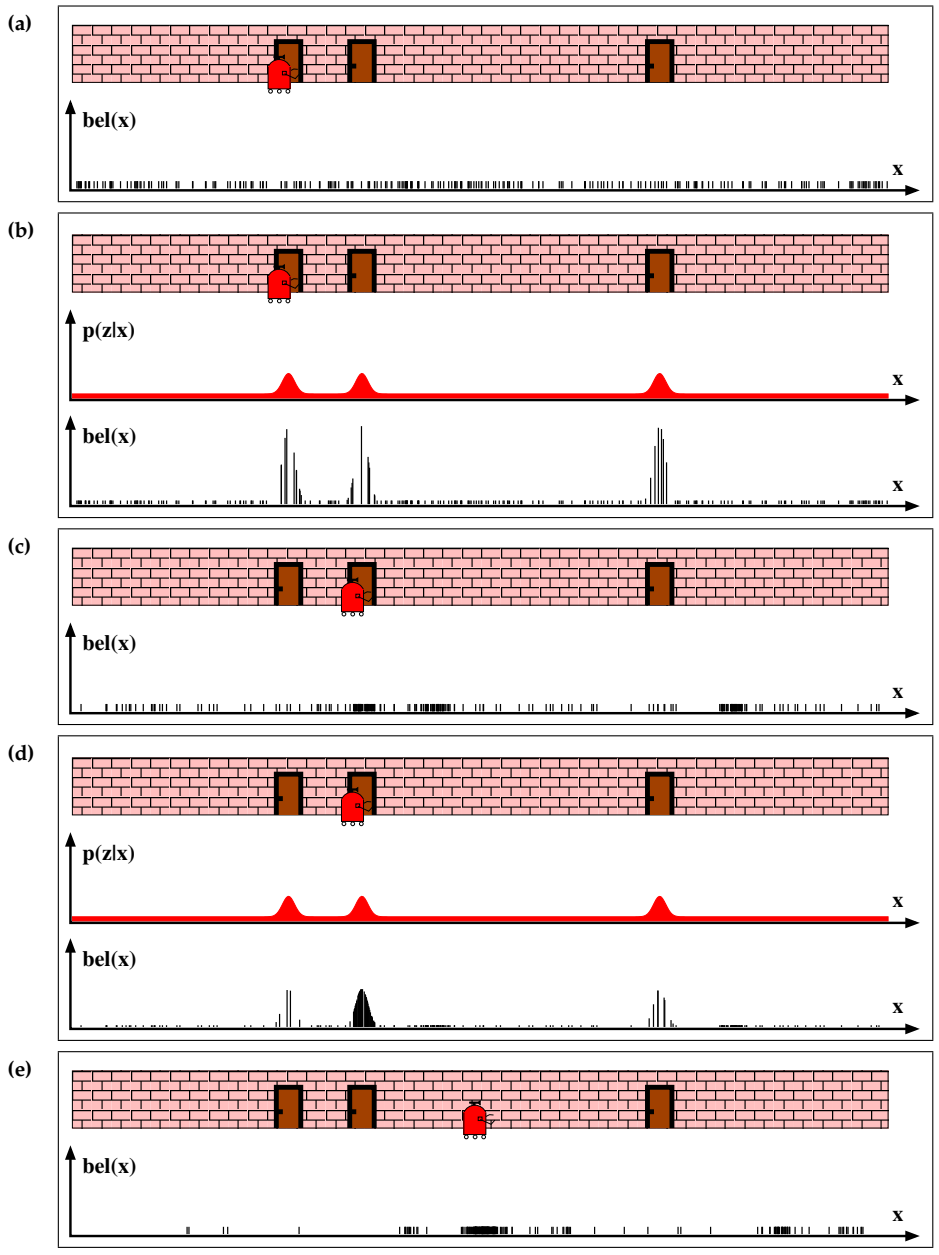

FIGURE 2.5: This is a simple example of the localization realized with a particle filter. It is similar to figure 2.4, but this time the belief is not a curve but instead represented by the distribution of particles [Thrun et al., 2005, p. 251].

Algorithm 3 Monte_Carlo_Localization [Thrun et al., 2005, p. 252]

1: **procedure** MONTE_CARLO_LOCALIZATION(X_{t-1}, u_t, z_t, m)
2: $\bar{X}_t = X_t = \emptyset$
3: **for** $m = 1$ to M **do**
4: $x_t^{[m]} =$ **sample_motion_model**$(u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} =$ **measurement_model**$(z_t, x_{t-1}^{[m]}, m)$
6: $\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: **end for**
8: **for** $m = 1$ to M **do**
9: draw i with probability $\propto w_t^{[i]}$
10: add $x_t^{[i]}$ to X_t
11: **end for**
12: **return** X_t
13: **end procedure**

sampled in line 4 and how the weights are computed in line 5. This happens according to the sample_motion_model and the measurement_model.

2.1.6 Global Localization

There is local localization and there is global localization. In the local localization problem the position at the beginning is already known. The particles are already clustered around the correct position at the beginning. The task then is just to track the position from there on out. In the global localization problem we have no information about the robot's location at the start.

A common approach to solve this problem is to spread the particles over the entire map at the start. After some iterations of weighing and sampling the particles they are supposed to be clustered around the real state. Sometimes this approach does not localize the robot in the right position and there are a number of circumstances that can increase the likelihood of such a global localization failure.

Many buildings have multiple rooms with similar structures. This can lead to the creation of multiple clusters and the pose is ambiguous. Many rooms are symmetric. This too can lead to clusters on multiple locations.

2.1.7 Kidnapped Robot Problem

The kidnapped robot problem occurs when, while performing localization, the robot is taken and placed at a different location. It won't be able to register where it went in most cases. Wheeled robots usually use odometers for the control data and once the robot is taken up and not on the ground they won't register anything anymore.

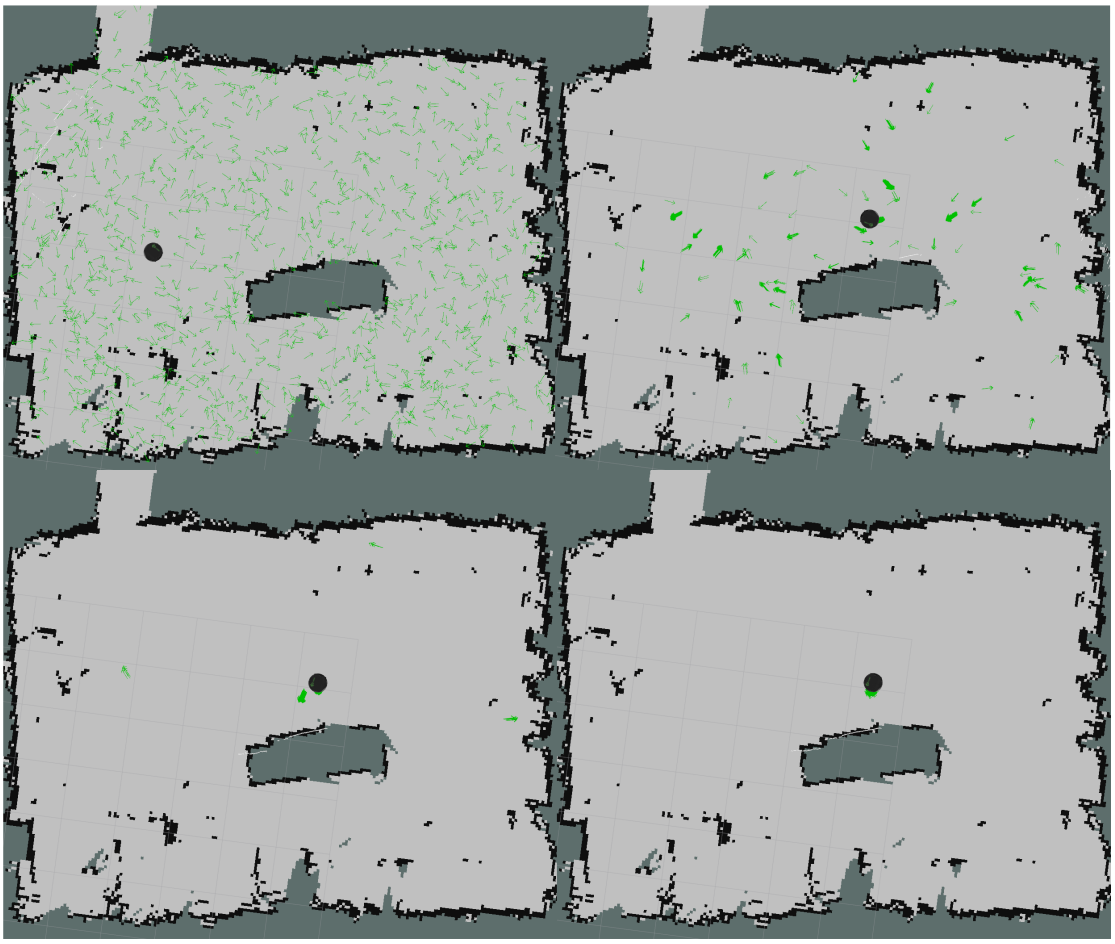

FIGURE 2.6: Example of global localization in our laboratory. The black circle represents the robot. The green arrows represent the position and orientation of the particles. In the upper left pictures the particles are spread over the entire state space. In the subsequent pictures the robot rotates to observe its environment. Slowly the particles converge to the correct location on the map and thus it is possible to infer the correct position of the robot.

This leads to problems when the particles are already clustered around one pose. When sampling new particles this cluster will persist and there will not be any particles on the rest of the map. To recover from such failures one can introduce a certain number of random particles into the set [Thrun et al., 2005, p. 256].

We could introduce a certain number of random particles in every iteration. But we can also use the measurement model and the generated weights as an indicator how likely the current pose is. For this purpose we calculate the average weight of the current iteration [Thrun et al., 2005, p. 257].

$$\frac{1}{M}\sum_{m=1}^{M} w_t^{[m]} \approx p(z_t|z_{1:t-1}, u_{1:t}, m) \tag{2.9}$$

This already gives us a good idea, but we don't want to rely on only a single

iteration to decide the induction of random particles. There could be an unusually high amount of noise or other reasons for inaccurate measurements. So we take the average over multiple iterations into account. We keep track of two different values:

$$\begin{aligned} w_{slow} &= w_{slow} + \alpha_{slow}(w_{avg} - w_{slow}) \\ w_{fast} &= w_{fast} + \alpha_{fast}(w_{avg} - w_{fast}) \end{aligned} \tag{2.10}$$

α_{slow} and α_{fast} are decay rates and constants that have to be set beforehand. w_{slow} is the long term measurement probability and w_{fast} is the short term probability. This means that past weighting averages have a higher influence on w_{slow} then they do on w_{fast}.

During the resampling process random particles are drawn with probability:

$$\max\{0.0, 1.0 - w_{fast}/w_{slow}\} \tag{2.11}$$

The higher the long term probability w_{slow} is compared to w_{fast} the more likely it is that random particles get introduced. That means the lower the recent weights are compared to the older ones, the higher the probability that random particles are drawn [Thrun et al., 2005, p. 258-259].

2.2 Wi-Fi

A Wi-Fi signal is spread from a base station known as access point. Each signal carries a unique identifier called media access control (MAC) address, which can be used to set the different signals apart. The Service Set Identifier (SSID) on the other hand isn't unique and there are often multiple access points distributing Wi-Fi signals under the same SSID. This makes the MAC-address relevant and the SSID irrelevant for the Wi-Fi position estimation [IEEE, 2012].

The Wi-Fi signal strength is measured in dBm. This is a logarithmic unit of measurement that is based on order of magnitude, rather than a linear scale. The reason for this is that the signal strengths deteriorate very quickly. According to our own experience the range of the signal is between about -90 dBm up to -40 dBm. The higher the number the better the signal. For our purposes the stability of the data transfer via the Wi-Fi connections is unimportant. We are only interested the received signal strength.

The access points send their signal on so called channels. Wi-Fi signals are spread on different frequency ranges. The ranges usually used are 2.4 GHz and 5 GHz. These frequency bands are then further divided into channels. So each Wi-Fi signal is spread on a specific channel. The more Wi-Fi signals in a close range send on the same channel, the more they interfere with each other.

Our goal is to fetch the MAC-address and the corresponding signal strength from every available Wi-Fi network, not only the one the robot is connected to. This

can be achieved by actively scanning the different Wi-Fi channels. In order to receive the data a probe request is sent to the available access points. Each access point answers. This has to be done on each supported channel. Sending out the request, waiting for the answer and repeating that procedure for each channel means that it can take a few seconds to complete a whole scan.

2.3 Wi-Fi Sensor Model

2.3.1 Overview

As discussed earlier, in order for the Monte Carlo localization to work with a certain type of sensor, we need to provide a measurement model for it. Our goal is to create one for a Wi-Fi receiver. For this purpose we chose to create a map of the Wi-Fi signal strengths. Doing this for Wi-Fi signals is different from creating a map of obstacles for example. For an obstacle map a point on the map can only have 3 different states: free, occupied and unknown. But Wi-Fi signals change constantly from position to position. This makes it more complicated to create an accurate map. It isn't feasible to record the Wi-Fi signal strengths on every single point of the map. Like we discussed the scanning process alone will take a few seconds. Thus we choose to record a fair amount of data points and use regression to interpolate a likely value for the Wi-Fi signal strengths at every point on the map. Wi-Fi signals can be unpredictable in the way they spread. With no obstacles between robot and access point it is reasonable to expect the signal to constantly get weaker the further we go away. But in buildings there are walls and other obstacles that can obstruct and deflect the signal.

Gaussian processes are a very flexible solution for this kind of problem. They are non-parametric and using hyperparameter optimization it is possible to fit them to the Wi-Fi signals. As we will show with a Gaussian process it is possible to compute a Gaussian distribution for every coordinate on the map. So given a coordinate x and y we can compute a Gaussian distribution with a mean and variance. The resulting Gaussian distribution is a model of the probability of the occurrence of a signal strength at that coordinate. This property makes it very well suited to use it as measurement model. We are not interested in the most likely signal strength observed at a certain point on the map, but in the probability that a given signal strength is observed. This is needed in order to compute the weights for particles of the Monte Carlo localization, so a Gaussian process is very well suited for this task.

In the further subsections we will explain how Gaussian processes work and how we can use them to create the Wi-Fi map and how we can use them for the measurement model.

2.3.2 Gaussian Processes

We have a dataset D of n observations of the form $D = \{(\mathbf{x_i}, y_i)|i = 1, ..., n\}$. In our example the $\mathbf{x_i}$ is a coordinate on the map and y_i is the associated Wi-Fi signal strength. Now in regression we want to infer the function values for new inputs. So we want to get a function f from the dataset D. But how do we infer function f? We need to make previous assumptions about the function because otherwise all functions that cross all training inputs and the corresponding values would be equally valid. The two common methods to achieve this are: [Rasmussen and Williams, 2005, p. 2]

1. Restrict function f to certain classes of functions.
2. Put prior probabilities on all possible functions.

When using the first solution depending on what classes are chosen, they can be a bad fit. For example one can imagine that linear functions would be a bad fit for Wi-Fi data. They are not flexible enough. On the other hand it can happen that the classes chosen lead to overfitting when they are too flexible [Rasmussen and Williams, 2005, p. 2].

In the second solution we would need to put a prior on all possible functions. The possible functions are infinitely many, so this seems like a difficult task. This is where the Gaussian process can help us.

A stochastic process is a generalization of the probability distribution. While the probability distribution concerns scalars and vectors, the process concerns functions [Rasmussen and Williams, 2005, p. 2]. Here we will focus on processes that are Gaussian. The reason for that is that is that it makes the computations required a lot easier. One can think of a function $f(x)$ as a vector, where each entry specifies a value for a specific input x. These vectors would be infinitely large. When we want to infer a function value from a Gaussian process at finitely many points it is as if we had taken the infinitely many other points taken into account, even though we ignore them [Rasmussen and Williams, 2005, p. 2].

To give a better intuition on how inferring the function f from the dataset D works we will give an example in form of figure 2.7. As one can see in (a) the functions are drawn without any observations. In (b) two observations were made. The functions go through the observed values. The thick line signifies the mean prediction. But this method also gives as a variance at each input value. The closer the input value is to an observation, the lower the standard deviation indicated by the gray shadows. The obvious reason is that we are more certain about the true function value the closer we are to the observations we made.

When using Gaussian processes the form of the drawn functions is determined by a covariance function [Rasmussen and Williams, 2005, p. 4]. This function determines the functions that are considered for inference. So a different covariance function would have created a different mean and variance and the functions drawn

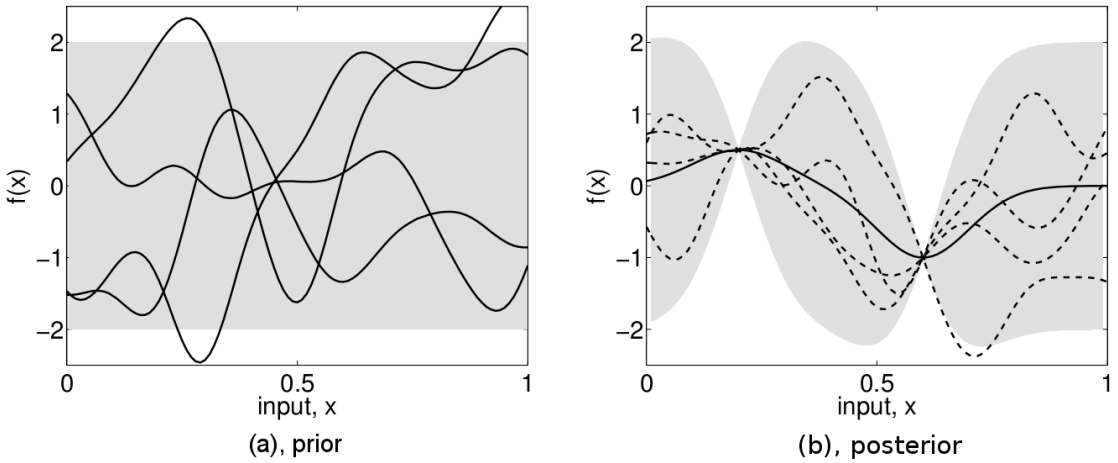

FIGURE 2.7: An example of how the prior distribution of functions works. In (a) functions were drawn from the prior. As one can see they have similar characteristics. In (b) the posterior is shown. Two points were observed. The solid line is the mean prediction. The shaded region is twice the standard deviation at each input value x [Rasmussen and Williams, 2005, p. 3]

.

from the prior in (a) would have looked different. This makes choosing a fitting covariance function important. A better fit produces a more accurate result. But covariance functions usually aren't static. There are parameters. But these can be learned, that is why they are called hyperparameters. So while we still have to choose a fitting covariance function for the Gaussian process they can be fit to the data by determining the right hyperparameters. How to do this will be discussed later in this section. In the following section we will define what a Gaussian process is and how it can be used for regression.

Regression Regression is a method to determine a relationship between a number of variables, using data that was recorded beforehand. So in our example we want to know the relationship between the Wi-Fi signal strength and the coordinates on the map. We are going to use Gaussian process regression to predict the likelihood of Wi-Fi signals at certain coordinates based on the Wi-Fi signals that we observed beforehand. To explain how we do this with Gaussian processes we are going to start with a simpler regression and go on from there.

We are going to look at the linear model from a Bayesian perspective.

Once again we have a training set D of n observations in the form of $D = \{(\mathbf{x_i}, y_i)|i = 1, ..., n\}$. To make things easier we are going to introduce vector $\mathcal{X}$ and y to form $D = (\mathcal{X}, y)$.

Now the standard linear regression model with Gaussian noise has the following form:

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{x}^{\mathbf{T}}\mathbf{w} \\ y &= f(\mathbf{x}) + \varepsilon \end{aligned} \tag{2.12}$$

Here $f(\mathbf{x})$ is the function value. $\mathbf{w}$ is a vector of weights, which act as the parameters of the linear function. y are the observed function values. Because the real values won't be exactly on the graph of $f(\mathbf{x})$ the noise term ε is added [Rasmussen and Williams, 2005, p. 8].

The noise term is of the form of a Gaussian distribution with zero mean and variance σ_n^2.

$$\varepsilon \sim \mathcal{N}(0, \sigma_n^2) \tag{2.13}$$

From this we infer the likelihood $p(\mathbf{y}|\mathbf{X}, \mathbf{w})$, so the probability density of the observations $\mathbf{y}$ given the weights $\mathbf{w}$ and the training inputs $\mathbf{X}$ [Rasmussen and Williams, 2005, p. 9].

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}, \mathbf{w}) &= \prod_{i=1}^{n} p(y_i|\mathbf{x_i}, \mathbf{w}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left(-\frac{(y_i - \mathbf{x_i}^T\mathbf{w})^2}{2\sigma_n^2}\right) \\ &= \frac{1}{(2\pi\sigma_n)^{n/2}} \exp\left(-\frac{1}{2\sigma_n^2}|\mathbf{y} - \mathbf{X}^T\mathbf{w}|^2\right) = \mathcal{N}(\mathbf{X}^T\mathbf{w}, \sigma_n^2 I) \end{aligned} \tag{2.14}$$

We can form the likelihood to a Gaussian distribution with mean $\mathbf{X}^T\mathbf{w}$, which is $f(x)$, and a variance of $\sigma_n^2 I$, which resembles the noise term ε.

Because we use the Bayesian formalism we need to specify a prior over the weights. This prior expresses what we believe their nature is like before we look at the observations. For this we use another Gaussian distribution. It will have a mean of zero and the variance is the covariance matrix Σ_p [Rasmussen and Williams, 2005, p. 9].

$$\mathbf{w} \sim \mathcal{N}(0, \Sigma_p) \tag{2.15}$$

To be able to predict new values, we need to know the weights. This means we want to infer the value of the weights from the training set. This is called the posterior and has the form $p(\mathbf{w}|\mathbf{X}, \mathbf{y})$. We can use Bayes' theorem in order to achieve this.

$$\begin{aligned} \text{posterior} &= \frac{\text{likelihood} \cdot \text{prior}}{\text{marginal likelihood}} \\ p(\mathbf{w}|\mathbf{y}, \mathbf{X}) &= \frac{p(\mathbf{y}|\mathbf{X}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y}|\mathbf{X})} \end{aligned} \tag{2.16}$$

$p(\mathbf{y}|\mathbf{X})$ is independent of the weights and therefore it is just a normalizing constant. Using the theorem of total probability it takes on the following form:

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{x}, \mathbf{w})p(\mathbf{w})d\mathbf{w} \tag{2.17}$$

Now leaving out the normalizing constant and only concentrating on the likelihood and prior we are able to form the posterior into a normal distribution, by using

equations 2.14 and 2.15.

$$\begin{aligned} p(\mathbf{w}|\mathbf{X},\mathbf{y}) &\propto \exp\big(-\frac{1}{2\sigma_n^2}(\mathbf{y}-\mathbf{X}^T\mathbf{w})^T(\mathbf{y}-\mathbf{X}^T\mathbf{w})\big)\exp\big(-\frac{1}{2}\mathbf{w}^T\Sigma_p^{-1}\mathbf{w}\big) \\ &\propto \exp\big(-\frac{1}{2}(\mathbf{w}-\bar{\mathbf{w}})^T(\frac{1}{\sigma_n^2}\mathbf{X}\mathbf{X}^T+\Sigma_p^{-1})(\mathbf{w}-\bar{\mathbf{w}})\big) \end{aligned} \tag{2.18}$$

Here $\bar{\mathbf{w}} = \sigma_n^{-2}(\sigma_n^{-2}\mathbf{X}\mathbf{X}^T + \Sigma_p^{-1})^{-1}\mathbf{X}\mathbf{y}$. Now the resulting Gaussian distribution is the following:

$$p(\mathbf{w}|\mathbf{X},\mathbf{y}) \sim \mathcal{N}(\bar{\mathbf{w}} = \frac{1}{\sigma_n^2}A^{-1}\mathbf{X}\mathbf{y}, A^{-1}) \tag{2.19}$$

with $A = \sigma_n^{-2}\mathbf{X}\mathbf{X}^T + \Sigma_p^{-1}$ [Rasmussen and Williams, 2005, p. 9].

Now we want to predict new values. We have a new input $\mathbf{x}_*$ with the function value $f_* \triangleq f(\mathbf{x}_*)$. In order to achieve this we use the posterior distribution and average over all possible parameter values [Rasmussen and Williams, 2005, p. 11].

$$\begin{aligned} p(f_*|\mathbf{x}_*,\mathbf{X},\mathbf{y}) &= \int p(f_*|\mathbf{x}_*,\mathbf{w})p(\mathbf{w}|\mathbf{X},\mathbf{y})d\mathbf{w} \\ &= \mathcal{N}(\frac{1}{\sigma_n^2}{\mathbf{x}_*}^T A^{-1}\mathbf{X}\mathbf{y}, {\mathbf{x}_*}^T A^{-1}\mathbf{x}_*) \end{aligned} \tag{2.20}$$

This is called the predictive distribution. With this distribution one can predict the function values for new inputs $\mathbf{x}_*$. But of course this solution is still very limited. It still only considers linear functions and won't be flexible enough for the Wi-Fi data.

But there is a trick we can apply in order to fix this flaw. We can simply project the inputs into high dimensional space by using a set of basis functions and apply the linear regression model in that space. A simple example would be $\phi(x) = (1, x, x^2, x^3, ...)$. As long as the function is independent of $\mathbf{w}$ one can still apply the linear regression to it [Rasmussen and Williams, 2005, p. 11].

We will look closer at the basis function at a later point, for now it is simply given by $\phi(x)$. This changes the model for linear regression as follows [Rasmussen and Williams, 2005, p. 12].

$$f(\mathbf{x}) = \phi(\mathbf{x})^T\mathbf{w} \tag{2.21}$$

Now applying the basis function to predictive distribution from equation 2.20 can easily be done, by just applying the basis function to the inputs.

$$f_*|\mathbf{x}_*,\mathbf{X},\mathbf{y} \sim \mathcal{N}(\frac{1}{\sigma_n^2}\phi(\mathbf{x}_*)^T A^{-1}\Phi\mathbf{y}, \phi(\mathbf{x}_*)^T A^{-1}\phi(\mathbf{x}_*)) \tag{2.22}$$

Here $\Phi = \Phi(\mathbf{X})$ and $A = \sigma_n^{-2}\Phi\Phi^T + \Sigma_p^{-1}$.

One drawback here is A^{-1}. We would have to invert A which is a $N \times N$ matrix, where N is the size of the feature space. But we can rewrite the formula

[Rasmussen and Williams, 2005, p. 12].

$$f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y} \sim \mathcal{N}(\phi_*^T \Sigma_p \Phi (K + \sigma_n^2 I)^{-1} \mathbf{y}, \phi_* \Sigma_p \phi_* - \phi_*^T \Sigma_p \Phi (K + \sigma_n^2 I)^{-1} \Phi^T \Sigma_p \phi_*) \tag{2.23}$$

Here $\phi(\mathbf{x}_*) = \phi_*$ and $K = \Phi^T \Sigma_p \Phi$.

Now we will introduce function $k(\mathbf{x_p}, \mathbf{x_q})$ which is called the covariance or kernel function.

$$\begin{aligned} k(\mathbf{x_p}, \mathbf{x_q}) &= \phi(\mathbf{x_p})^T \Sigma_p \phi(\mathbf{x_q}) \\ &= \psi(\mathbf{x_p}) \cdot \psi(\mathbf{x_q}) \end{aligned} \tag{2.24}$$

The fact that we can rewrite this function as a dot product makes it possible to apply the so called kernel trick [Rasmussen and Williams, 2005, p. 12]. We can define a function $k(\mathbf{x_p}, \mathbf{x_q})$ and then replace all occurrences by it. The kernel trick is used to lift the inputs into a higher dimensional space, while making all computations still in the input space. This saves a lot of computation time and memory. So we are going to place the kernel function where we can in equation 2.23.

$$f_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y} \sim \mathcal{N}({\mathbf{k}_*}^T (K + \sigma_n^2 I)^{-1} \mathbf{y}, k(\mathbf{x}_*, \mathbf{x}_*) - {\mathbf{k}_*}^T (K + \sigma_n^2 I)^{-1} \mathbf{k}_*) \tag{2.25}$$

with $\mathbf{k}_*$ denoting a vector of covariances between the test point and the training points, and K being the covariance matrix of all training points, so $K = K(\mathbf{X}, \mathbf{X})$.

Equation 2.25 now enables us, given the training data $\mathbf{X}$ and $\mathbf{y}$, to compute a Gaussian distribution for a new data point $\mathbf{x}_*$. For the Wi-Fi data that would mean that given previously recorded coordinates $\mathbf{X}$ and corresponding Wi-Fi signal strengths $\mathbf{y}$ from one access point and a new coordinate $\mathbf{x}_*$ we would could compute Gaussian distribution. Now given a newly observed Wi-Fi signal strength from the same access point, we would be able to determine a likelihood that it was observed at coordinate $\mathbf{x}_*$, simply by using the computed Gaussian distribution. This is exactly what we need to compute the weights for the Monte Carlo localization from section 2.1.5.

Now we have everything we need to fully define the Gaussian process and then predict values from it.

According to Rasmussen and Williams [2005, p. 13] a Gaussian process is fully defined as follows:

$$\mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{2.26}$$

The mean function $m(\mathbf{x})$ is usually defined as 0.

The kernel function $k(\mathbf{x}, \mathbf{x}')$, also called covariance function, is an important aspect for the Gaussian process. There are many possibilities to choose from for kernel functions. The functions considered as kernel functions have to be symmetric and positive semi-definite.

But while there are many functions that could be used as kernel, we are going to focus on one of the most popular ones. The radial basis function kernel. (RBF

kernel). Ferris et al. [2006] showed that this kernel is very well suited for the usage with Wi-Fi signal strengths.

$$k(x_p, x_q) = \sigma_f^2 \exp(-\frac{1}{2l^2}(x_p - x_q)^2) + \sigma_n^2 \delta_{pq} \tag{2.27}$$

Here δ_{pq} is the Kronecker delta, that is 1 whenever $p = q$ and 0 else. This particular kernel function results in a very smooth graph. There are three so called hyperparameters. These are variables, but they don't have to be set manually, but can actually be learned. How will be discussed in the next section. The hyperparameters here are the lengthscale l, the signal variance σ_f^2 and the noise variance σ_n^2.

Hyperparameter Optimization The Gaussian process regression is a parameterless regression method. But we have to deal with so called hyperparameters. The advantage here is that we can use optimization algorithms in order to find values that are working well. In order to find those values we take the log-likelihood and maximize it. The log-likelihood function gives us an indication how well the Gaussian process fits to the data. The higher the value of the log-likelihood function the better the closer the Gaussian process resembles the given data.

The log-likelihood is defined by the following function [Rasmussen and Williams, 2005, p. 113].

$$log\, p(\mathbf{y}|\mathbf{X}, \theta) = -\frac{1}{2}\mathbf{y}^T(K + \sigma_n^2 I)^{-1}\mathbf{y} - \frac{1}{2} log\, |K + \sigma_n^2 I| - \frac{n}{2} log\, 2\pi \tag{2.28}$$

Here θ are the hyperparameters l, σ_n^2 and σ_f^2. In order to use optimization algorithms we also need the partial derivatives of the function [Rasmussen and Williams, 2005, p. 114].

$$\frac{\partial}{\partial \theta_j} log\, p(\mathbf{y}|\mathbf{X}, \theta) = \frac{1}{2} tr\left((K^{-1}\mathbf{y})(K^{-1}\mathbf{y})^T \frac{\partial K}{\partial \theta_j} \right) \tag{2.29}$$

So, we need a method that uses the information from the log-likelihood function and its gradient, and maximizes the value by adjusting the hyperparameter values. Some examples of algorithms that are suitable to optimize the hyperparameters are the gradient descent [Shewchuk, 1994], BFGS or L-BFGS [Liu and Nocedal, 1989]. Blum and Riedmiller [2013] propose to use resilient backpropagation (Rprop) to solve this problem. They show that the algorithm has a similar performance as L-BFGS. But it has the advantage over L-BFGS and similar methods that it is easier to implement.

Like most methods the algorithm requires not only the function itself, but also the gradient. But it doesn't use the second order derivatives or an approximation thereof, which leads to shorter computation times per iteration [Blum and Riedmiller, 2013].

In every iteration the hyperparameters θ are updated depending on the sign of the derivative:

$$\theta_i^{(t+1)} = \theta_i^{(t)} - sign\left(\frac{\partial J^{(t)}}{\partial \theta_i}\right)\Delta_i^{(t)} \tag{2.30}$$

Δ_i is the update-value. Depending on the change of the sign the hyperparameter, Δ_i is either increased by a factor of $\eta^+ > 1$ or decreased by a factor $0 < \eta^- < 1$.

$$\Delta_i^{(t)} = \begin{cases} \eta^+ \cdot \Delta_i^{(t-1)}, \text{ if } \frac{\partial J}{\partial \theta_i}^{(t-1)} \cdot \frac{\partial J}{\partial \theta_i}^{(t)} > 0 \\ \eta^- \cdot \Delta_i^{(t-1)}, \text{ if } \frac{\partial J}{\partial \theta_i}^{(t-1)} \cdot \frac{\partial J}{\partial \theta_i}^{(t)} < 0 \\ \Delta_i^{(t-1)}, \text{ else} \end{cases} \tag{2.31}$$

The initial update value is set to Δ_0 and is bounded by Δ_{min} and Δ_{max}. The parameters have to be specified, but there are values that work for most cases [Blum and Riedmiller, 2013].

So in order to get good values for the hyperparameters we need to apply the algorithm to the partial derivatives as specified in equation 2.29. Like most optimization algorithms it tries to minimize the function value, but we need to maximize it. The higher the log-likelihood is the better the fit. So we simply use the negative log-likelihood and its negative partial derivatives.

The algorithms will run for a set number of iterations or until every partial derivative is 0. At every iteration we check the negative log-likelihood and compare it with the best result we found yet. If the new result is smaller we save the current hyperparameters. At the end of this process the Gaussian process will fit well to the training data [Blum and Riedmiller, 2013].

2.4 Chapter Summary

At the beginning of this chapter we explained the basics of localization. We moved from the uncertainty involved in robotics to the recursive state estimation to its implementation with a particle filter. At the end of section 2.1 we showed the Monte Carlo localization that uses particle filters. For the Monte Carlo localization to actually work we need sensor models. For a Wi-Fi receiver Gaussian processes are well suited. So, in the second half of this chapter we determined what a Gaussian process is and showed how it can be used for regression by deriving it from the linear regression. Even though the Gaussian process regression is a parameterless method, there are hyperparameters that have to be optimized. We explained how this can be done using Rprop.

In the next chapter we will use this knowledge to explain our implementation of the Wi-Fi position estimation, by using Gaussian processes and the Monte Carlo localization.

Chapter 3

Implementation

This chapter discusses the implementation of the parts needed for a functioning Wi-Fi position estimation. In section 3.1 we will introduce the Robot Operating System(ROS) and explain its basic components. In section 3.2 we will explain how the required Wi-Fi data gets fetched and published for other parts of the program. Section 3.3 gives a short overview about the used implementation of the Monte Carlo localization for the 2D laser range finder. In section 3.4 we will show how the published Wi-Fi and position data from the robot is collected and stored for further use with the Wi-Fi position estimation. The section 3.5 details the implementation of the Gaussian process for the Wi-Fi position estimation. And at last section 3.6 will tie the previous chapters together and explain the Wi-Fi position estimation as a whole.

3.1 ROS

The Wi-Fi position estimation was implemented with the ROS-Framework and run on a TurtleBot 2 (http://www.turtlebot.com/, accessed on 02.11.2016). ROS [Fernandez et al., 2015] [Quigley et al., 2009] is short for Robot Operating System. It supports a wide variety of robots and was designed so that different robots and environments have a common basis to make it easier to share.

The software in ROS is divided into packages. So each package usually has a different functionality or purpose. An example used in this project is the AMCL package and the created package for the Wi-Fi position estimation. These packages contain nodes. The nodes are processes. Before the nodes can be executed a master has to be started. On this master the nodes are registered, so that different nodes can see each other and can communicate with each other.

Another important part of the ROS-framework for basic functionality is the parameter server. It can be used in order to supply a node with manually set parameters when the node is started. This makes it easier to customize the nodes.

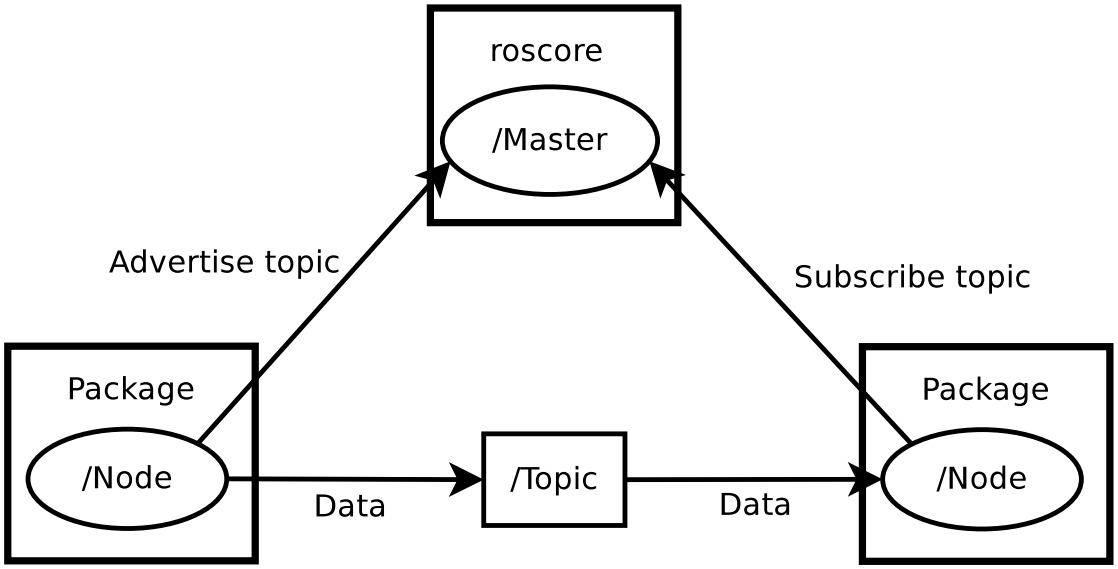

FIGURE 3.1: Relationship of the basic parts of the ROS environment.

For the purpose of communication between the nodes there are topics and services. Topics can be either published or subscribed to. For example, there is the AMCL node, that publishes a pose estimation and a Wi-Fi publisher node, that publishes the Wi-Fi signal strengths, and the Wi-Fi data recorder is subscribed to these nodes so it can save the signal strengths and the corresponding position on the map. A service can be offered or called by a node. For example AMCL offers global localization via service. This means services are only performed, whenever they are called, while the data published on topics usually gets published constantly.

The ROS framework offers a wide variety of packages for a wide variety of robots. The advantages of this are that we don't have to build everything from the ground up. There is already existing software that is maintained by a large community. Different packages can be re-used for different robots, as long as these robots run ROS.

There are some other ROS packages and nodes that were used in this project. The rosbag package (http://wiki.ros.org/rosbag, accessed on 31.10.2016) can be used to record data and store it to play it back later. For the nodes there is no difference if the robot is actually producing them right now or if they are just played back from a rosbag. This was for example used to carry out experiments to apply different algorithms to the same set of data.

Another package we used is RViz (http://wiki.ros.org/rviz, accessed on 31.10.2016). It was used to visualize the Wi-Fi data and the Gaussian process for various figures in this thesis.

Starting these nodes can be done via the command line. The typical way to do this is the command "rosrun package_name node_name". Another option is creating a launch file and starting it with the roslaunch-package (http://wiki.ros.org/roslaunch,

accessed on 01.11.2016). These files are written in the XML-format. They can contain parameters that are used by the node launched. A launch file can be started using the terminal command "roslaunch package_name launch_file.launch".

3.2 Wi-Fi Data Publisher

The first step to realize the position estimation is to actually get the Wi-Fi data. For this we use an active Wi-Fi scan. We wait for the result and handle the data. Then the SSIDs, MAC-addresses and signal strengths are published. This is procedure gets repeated until the node is stopped. As already discussed the scan can take up to a few seconds. Of course this means that the rate at which the node publishes new Wi-Fi data is limited by the time the scan takes as well.

The node publishes a topic called "/wifi_state". It contains the Wi-Fi signal strength in dBm, the MAC-addresses and the SSIDs.

In order to get this data the following terminal command is used:

```
$ sudo iw dev wlan0 scan
```

Here wlan0 is the used network interface. This value can be different on other systems. This command starts a scan to find all Wi-Fi networks that can be found. It provides a lot of information about every network and among them is the signal strength and the MAC-address.

Without further specification the command will scan all channels, but it is also possible to specify a frequency that corresponds to a Wi-Fi channel. This will limit the Wi-Fi networks that can be found, but it is also a lot faster. Here is a simple example:

```
$ sudo iw dev wlan0 scan freq 2412
```

Different frequencies correspond to different Wi-Fi channels. 2412 MHz for example is the frequency of channel 1.

In our experience the complete scan takes about 3-5 seconds, depending on the Wi-Fi receiver used. The scan using a specified frequency only takes a fracture of a second. A short test resulted in approximately 20 scans per second, but once again this depends on the Wi-Fi receiver used. Since we want the data from all available Wi-Fi access points only the complete scan was used.

3.3 AMCL

We use the AMCL package (http://wiki.ros.org/amcl, accessed on 25.10.2016) for the localization using the 2D range finder. The package contains an implementation of the Monte Carlo localization and furthermore provides the needed measurement model for the laser range finder. The package implements the following specific algorithms from Thrun et al. [2005]:

- sample_motion_model_odometry (A sample motion model using the odometry.)
- beam_range_finder_model (A measurement model using the laser range finder or Kinect.)
- likelihood_field_range_finder_model (A measurement model using the laser range finder or Kinect.)
- Augmented_MCL (The Monte Carlo Localization with the mechanic from section 2.1.7)
- KLD_Sampling_MCL (The Monte Carlo Localization with a mechanism that varies the number of particles depending on how certain the Monte Carlo Localization is about the localization.)

The AMCL node needs a map of the environment the robot is supposed to be localized in. The map is typically created beforehand for example by using a package like gmapping (http://wiki.ros.org/gmapping, accessed on 01.11.2016). The map can be provided to AMCL by using the "map_server" package (http://wiki.ros.org/map_server, accessed on 01.11.2016). So a map_server node loading the correct map has to be started before starting the AMCL node. When started, AMCL publishes the pose of the robot determined by the localization. This pose contains the x and y coordinate and the orientation as a quaternion on the given map. The poses also contain a covariance matrix, that represents the spread of the particles around the determined position.

It should be noted that no matter how the robot is moved, AMCL will continue to localize the robot as long as the wheels are used to move it. So only when the robot is picked up and moved, AMCL is not able to track the position.

The package has multiple use cases in this project. For one it is used for the data collection, so that we know where on the map the data was recorded. It is also used in conjunction with the Wi-Fi position estimation, to test how well it works as an aid for the global localization.

The last use case is to use the Wi-Fi position estimation in case of a localization failure. We can use the method discussed in section 2.1.7 as an indicator if there is a localization failure. Using the value from equation 2.11 we can infer the quality of the localization. Usually AMCL does computes this value, but does not publish

it so that other nodes can use it. AMCL was slightly modified, so that it publishes that value under the identifier "/amcl_failure_probability". Here the higher the given value, the more likely it is that there is a localization failure.

3.4 Data Collection

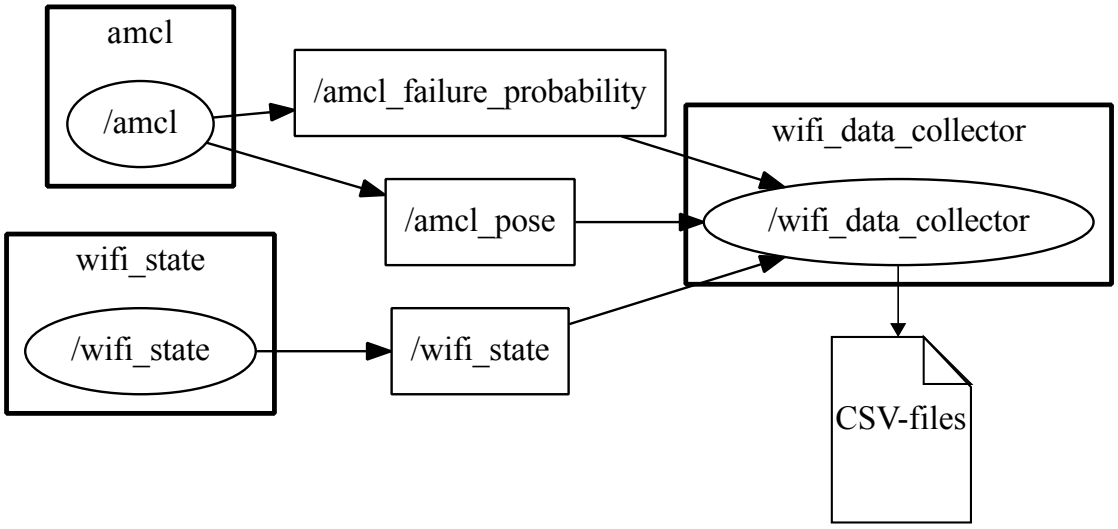

FIGURE 3.2: Overview over the topics the Wi-Fi_data_collector subscribed to and the related nodes. The /amcl_pose and /wifi_state topics are publishing the data that is supposed to be stored. The /amcl_failure_probability topic gives us an indication if there is a localization failure and in that case when a certain threshold is exceeded it stops recording the data. The data is then stored in a number of CSV files.

In order to collect the data, the aforementioned Wi-Fi data publisher is used in combination with AMCL. We need to know the position of the robot on the map, if we want to use the data later on to build a map of the signal strengths. AMCL publishes the estimated pose of the robot and the Wi-Fi data publisher the measured MAC-addresses and signal strengths. So the data collection node updates the pose whenever there is a new one and whenever new Wi-Fi data is published it stores the data together with the pose in comma separated value (CSV) files. For each MAC-address there is a separate file.

There are some options to customize the data collector. One is if the data is supposed to be collected only when standing still or if the robot should collect the data constantly. Another option is to load existing CSV files, so that the data collector can add new data points to them.

In order to collect the data the mobile robot needs to move around. In order to do this the "move_base" package (http://wiki.ros.org/move_base, accessed on 03.11.2016) can be used. One can provide a goal consisting of coordinates from the used map and an orientation and then the robot will move to this location. Another

way to move the robot is the "turtlebot_teleop" package (http://wiki.ros.org/turtlebot_teleop, accessed on 03.11.2016). Here one has to control the movement of the robot for example via a keyboard or via a joystick.

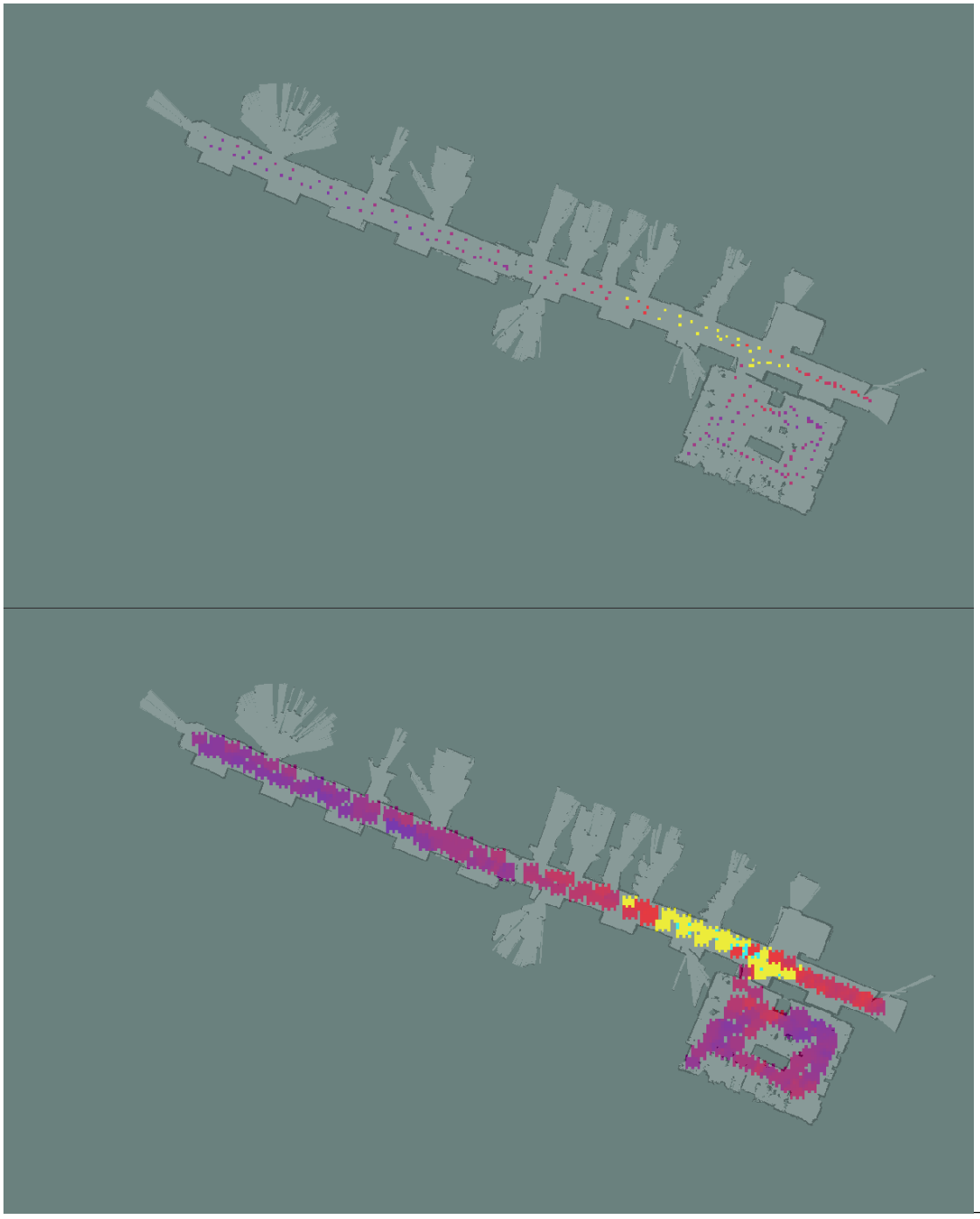

FIGURE 3.3: Both pictures represent Wi-Fi data from one access point, recorded on the hallway of the TAMS research group at the university of Hamburg. Above each dot is a certain recording at the position on the map. Beneath we interpolated the data, to highlight the behavior of the signal strength. The colors from strong to weak signal strength: yellow, light blue, red, purple, blue.

3.5 Gaussian Process

The model from section 2.3 was used to implement the Gaussian process. A change was made to the formula for the kernel. The reason for this is that we want to prevent the hyperparameters from taking on negative values. This means we either have to apply an optimization algorithm that works for constrained problems, or make sure the parameters can't take on negative values. The first solution would mean we would have to use a more complicated algorithm, therefore we decided to take the second approach.

The covariance function from equation 2.27 is changed to the following form:

$$cov(y_p, y_q) = \sigma_f \exp\left(- \frac{|x_p - x_q|^2}{l} \right) + \sigma_n \delta_{pq} \tag{3.1}$$

In order to make sure that the hyperparameters can only take on positive values they are substituted by the following terms:

$$\begin{aligned} \sigma_f &= exp(2\theta_1) \\ l &= exp(\theta_2) \\ \sigma_n &= exp(\theta_3) \end{aligned} \tag{3.2}$$

This leads to changes for the gradient too:

$$\begin{aligned} \frac{\partial cov}{\partial \theta_1} &= 2\sigma_f exp\left(- \frac{|x_p - x_q|^2}{l} \right) \\ \frac{\partial cov}{\partial \theta_2} &= -\sigma_f exp\left(- \frac{|x_p - x_q|^2}{l} \right)\left(- \frac{|x_p - x_q|^2}{l} \right) \\ \frac{\partial cov}{\partial \theta_3} &= \sigma_n \delta_{pq} \end{aligned} \tag{3.3}$$

Now the optimization algorithm is applied to θ_1, θ_2 and θ_3. When these become negative, σ_f, l and σ_n will stay positive.

The Gaussian process is not a node, but a simple class. In order to create a Gaussian process usually a path to a folder with CSV-files is provided. These files then contain training data, that is, the positions and corresponding signal strengths. Each MAC-address usually has its own CSV-file named after itself, so that the Gaussian process can distinguish them.

The implementation of the Gaussian process was done with the Eigen-library (http://eigen.tuxfamily.org/, accessed on 28.10.2016). There are many matrix- and vector-operations needed and the Eigen-library is well suited for these kind of problems, as it provides matrix- and vector-classes with a number of useful algorithms for them.

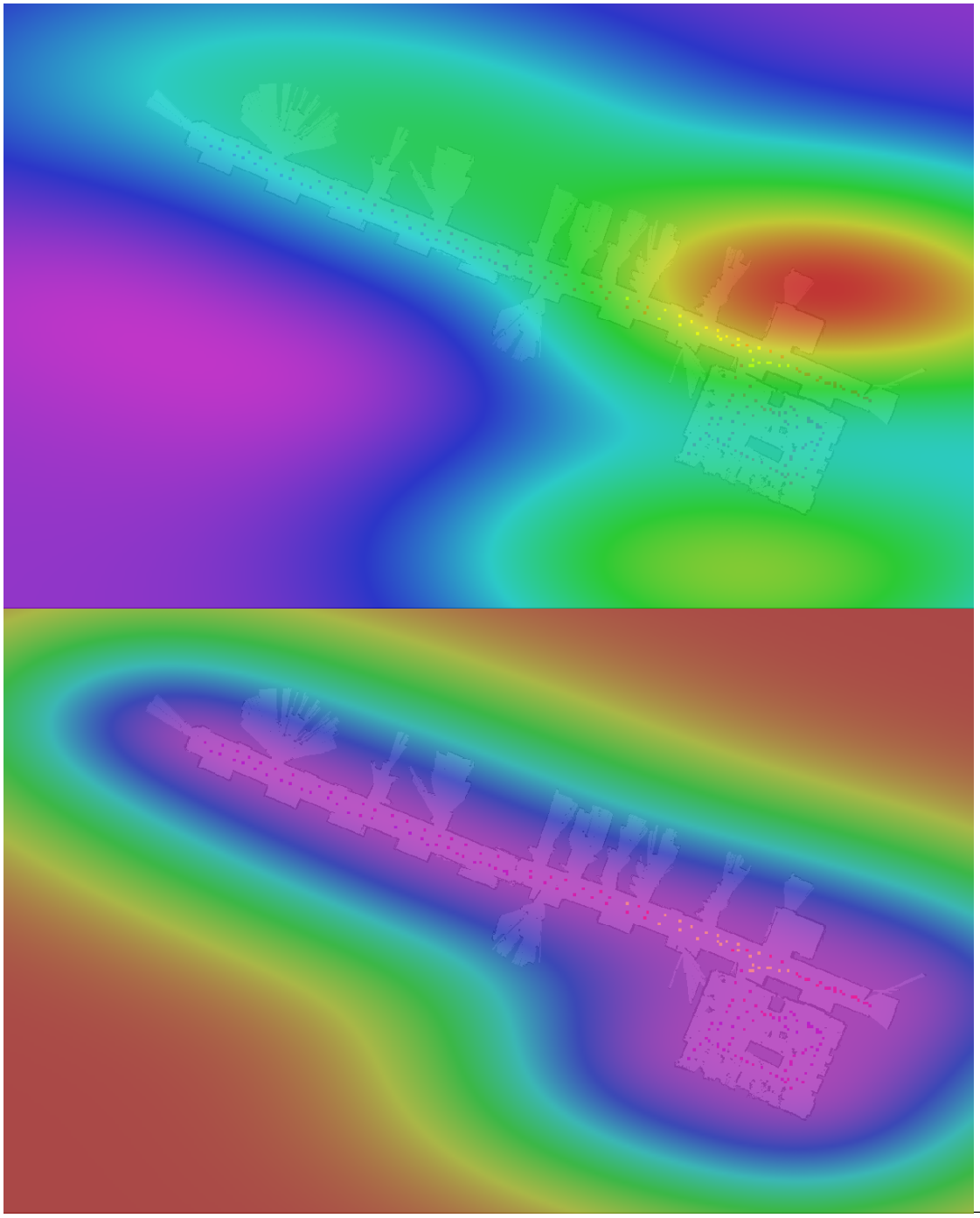

FIGURE 3.4: The picture above represents the mean of a Gaussian process that was trained for the data from figure 3.3. The colors from high to low: red, yellow, green, light blue, dark blue, purple. Below is the corresponding variance of the same process. As one can see the variance is very low close to the region where the data was recorded. This happens because we are more confident about the signal strengths in these regions. The further away we go, the higher the variance.

For the optimization of the hyperparameters the Rprop-algorithm detailed in section 2.3.2 was implemented in a separate class. So once a Gaussian process was created, the Rprop-class can be used in order to train the Gaussian process, so that it fits the training data.

Another aspect to consider is the nature of the Wi-Fi signal strength data and how it affects the Gaussian process. We are using a constant mean function of 0. In practice that means that when moving away from any training data, the mean of the Gaussian process will converge to 0.

We decided to normalize the data to a range from 0 to 1. In order to do this we decided to use a maximum of 0 dBm and a minimum of -100 dBm. In our experience these values are not reached in real world scenarios. This is especially important for the minimum. A value of -100 dBm would be 0 after normalization and this would mean it would be indistinguishable from a value far outside the range of the training data. This is a scenario we want to avoid. Another reason for the normalization is that the used optimization algorithm, the already discussed Rprop-algorithm, works better with the normalized values. The resulting operation for the normalization is the following:

$$\begin{aligned} \mathbf{y}' &= \frac{\mathbf{y} - (-100)}{0 - (-100)} \\ &= \frac{\mathbf{y} + 100}{100} \end{aligned} \tag{3.4}$$

This operation is both applied to the training data and to the new incoming data, when a weight for a particle is computed.

3.6 Wi-Fi Position Estimation

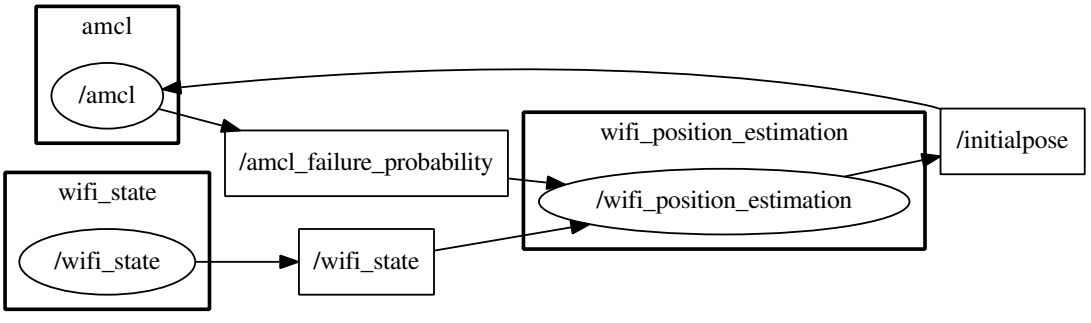

FIGURE 3.5: A basic diagram of the Wi-Fi_localization and the subscribed and published topics.

The Wi-Fi position estimation uses the data that was collected to create a number of Gaussian processes. For every network a new Gaussian process is created.

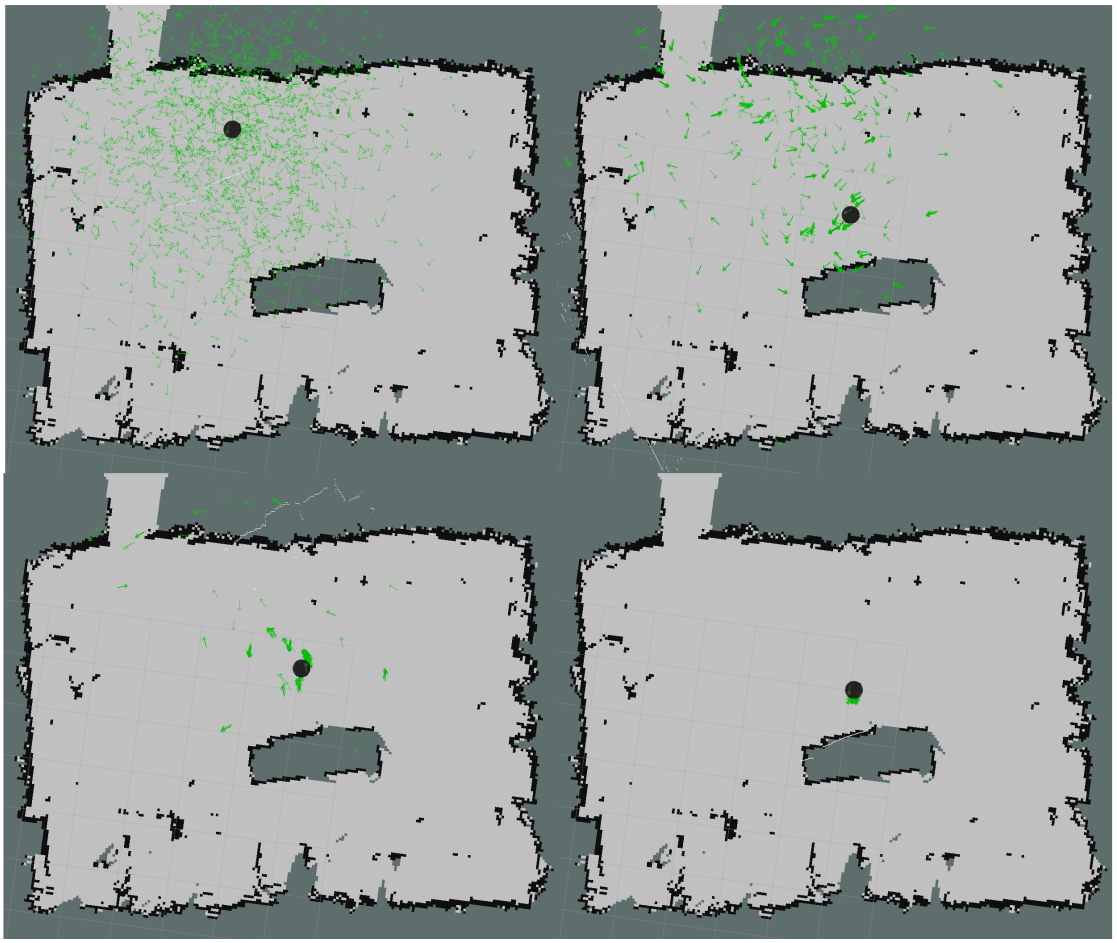

FIGURE 3.6: Example of using the Wi-Fi position estimation for global localization in our laboratory. The black circle represents the robot and the green arrows the position and orientation of the particles of AMCL using the 2D laser range finder. In the upper left picture the Wi-Fi position estimation was performed already. Compared to figure 2.6 one can see that the particles are a lot more concentrated and not spread over the entire state space. The position is off by a certain margin. In the following pictures the robot is turning around itself to observe its environment with the 2D laser range finder. The particles are quickly clustered around the correct location and the thus the correct position of the robot can be inferred.

Since we only try to estimate the position we basically only replicate the first step of the global Monte Carlo localization. So particles are spread across the map.

The Wi-Fi state publisher node will provide us with the current Wi-Fi signal strengths of the different networks. We will use this data with the coordinates from the particles to compute the weights for each particle.

The processes get the coordinates, and the signal strength that corresponds to the MAC-address of the data that the particular Gaussian process was created with. For each particle the weights are computed by multiplying the computed values from each MAC-address. So for n MAC-addresses we get the following formula to

compute the weight:

$$w(x_*) = \prod_{i=1}^{n} p(z_t^{[i]}|x_*) \tag{3.5}$$

At the end the highest weight for a particular position determines which position is the most likely one. This position is then sent to AMCL. AMCL will use it as a new start position and will proceed the localization from those coordinates.

AMCL will reset its own particles and spread them around that location. How far the particles are spread is determined by a covariance matrix. These values are not computed, but set manually.

So the first use case would be the manual initiation of the position estimation in order to give AMCL a seed for the localization. This would be similar to a global localization. The robot doesn't know where it is on the map and has to deduce its own position with the available data.

Another use case would be in case of a localization failure. To determine a localization failure we use the "/amcl_failure_probability" published by AMCL using the 2D laser range finder.

We specify a threshold. If the computed value from equation 2.11 is bigger than the threshold, the Wi-Fi position estimation is triggered. A position will be computed and sent to AMCL as a new start position.

3.7 Using the Wi-Fi Position Estimation

In the preceding sections we explained the implementation of all the parts of the system. So this chapter will discuss the appliance of said software in the real world. The robot used is a TurtleBot 2 (http://www.turtlebot.com, accessed on 02.11.2016) with a Kinect [Andersen et al., 2015]. To complement the Kinect there is also the laser range finder "Hokuyo UTM-30LX" [las, 2012] installed. It can be seen in figure 2.2. According to Microsoft the Kinect has a scanning range of 4 meters, but the data is still usable some meters beyond [Andersen et al., 2015]. The laser range finder has a scanning range of 30 meters [las, 2012], so it is vastly superior for our purposes. The environment we tested the software in was mainly a long hallway at the TAMS research group at the university of Hamburg that can be seen in figure 3.8.

The goal now is to put the Wi-Fi position estimation to use. The first step here is to first collect data in the environment the robot is supposed to localize itself. This can be done with AMCL from section 3.3 by using the laser range finder, the Wi-Fi publisher from section 3.2 and the Wi-Fi data collector from section 3.4. So one needs to start the nodes one after another. It is also important to make sure that AMCL's localization is close to the real position, so it is advisable to set the position manually at the start. An easy way to do this is the rviz-node. It can

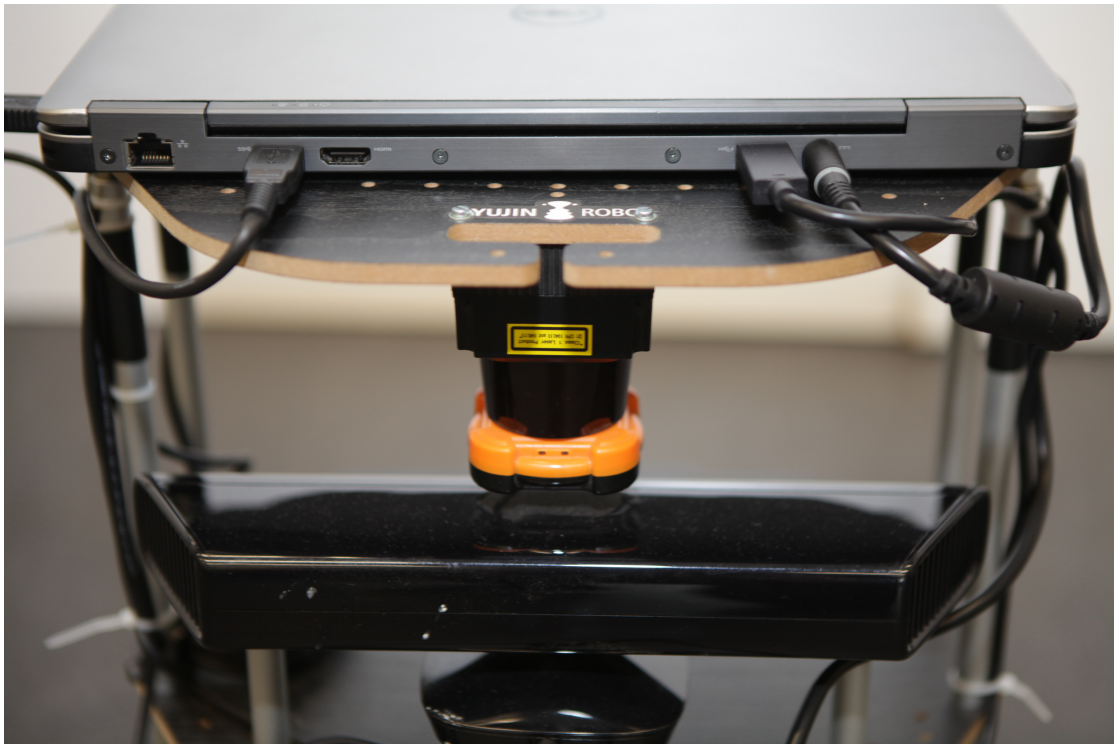

FIGURE 3.7: The laser range finder used for the TurtleBot.

FIGURE 3.8: The hallway of the TAMS research group at the university of Hamburg where we tested the software and carried out the experiments in chapter 4.

visualize the map used and offers the option to publish a new initial-pose to AMCL via clicking on the correct position on the map. The data collector offers different configurations. An important one is whether the Wi-Fi data should always be collected or only when standing still. In our tests we decided for the latter. To configure the node the launch file for the node can be customized. It contains the used parameters.

So when the nodes are started the Wi-Fi data can be collected by calling the rosservice "/start_wifi_scan" with the value "true". Now the data is recorded according to the chosen configuration. Another node not mentioned yet is the "map_traverser_node". This node gets a list of x- and y-coordinates as parameter and then drives to these positions one after the other. This node can also be used to record the data. Another method is to operate the robot via keyboard with the "turtlebot_teleop" package (http://wiki.ros.org/turtlebot_teleop, accessed on 27.10.2016). The Wi-Fi and position data is then saved in CSV-files. The Wi-Fi data can also be visualized by using occupancy grids. Calling the service "/publish_wifi_maps" publishes the occupancy grids that can be used to visualize the currently collected data. The topics the occupancy grids are published to are named after the data's MAC-address and an identifier of the nature of the visualization. There are four identifiers:

- ln (local normalized): normalized with the minimum and maximum of only this access point.
- li (local interpolated): normalized with the minimum and maximum of only this access point and interpolated.
- gn (global normalized): normalized with the minimum and maximum of all access points.
- gi (global interpolated): normalized with the minimum and maximum of all access points and interpolated.

The collected Wi-Fi data can then be used as training data for the Gaussian process. The Wi-Fi position estimation node is given a parameter that contains the path to a folder with the CSV-files containing the training data. There is a launch file that contains these parameters and should be customized to point to the correct folder containing the recorded Wi-Fi data.

Once started the node either trains the Gaussian processes created from the training data, or if this was already done when previously running the node it loads the already optimized hyperparameters. The optimized hyperparameters are saved as CSV-files as well, in a subfolder of the folder containing the training data. Once the node has been initialized fully it can be used to approximate the position of the robot. In order to send the position estimation to AMCL the service called "compute_amcl_start_point" can be used. Another parameter given to the Wi-Fi position estimation node is a threshold for the "/amcl_failure_probability" value published by AMCL. Is the value above this threshold the Wi-Fi position estimation is triggered to recover from the localization failure.

3.8 Chapter Summary

In this chapter we discussed the implementation of all parts needed for the Wi-Fi position estimation. At first we explained the ROS-framework used for the implementation of the system. Afterwards the parts needed to collect the Wi-Fi data were shown, with the Wi-Fi data publisher, AMCL and the Wi-Fi data collector. The collected data could then be used as training data. In order to implement the position estimation first the implementation of the Gaussian process was discussed and afterwards its application for the Wi-Fi position estimation node. At last we were giving information on the particular hardware used and explained how to use the implemented software.

We now have a system that is able to estimate the position via Wi-Fi data. In the next chapter we will use it in different experiments to determine how effective it is for the proposed tasks.

Chapter 4

Experiments and Results

In order to determine how well the Wi-Fi position estimation works, we carried out a number of experiments. First we will analyze the collected Wi-Fi data in section 4.1.1. In section 4.1.2 we will assess the accuracy of the Wi-Fi position estimation by itself. The following section 4.1.3 contains two experiments that determine how well the Wi-Fi position estimation can aid AMCL with a laser range finder. In the first experiment we evaluate how well it can help with the global localization. In the third and last experiment we inspect how well the localization failure detection works.

4.1 Data Analysis and Experiments

4.1.1 Wi-Fi Data

We want to examine the Wi-Fi data and look at how it behaves. Figure 4.1 shows an example of the Wi-Fi strength of one access point. The signal reaches its peak close to the access point, marked by the yellow color. The further away the data was collected the weaker the signal was. In this example the access point was in the same room and there are basically no obstacles between the access point and the robot at any position. Another example can be seen in figure 4.2. Here the signal never reaches the strength of figure 4.1. But still the peak is at the center of the recorded data. To both sides then it gets weaker until it the Wi-Fi receiver can't pick it up anymore. It is reasonable to assume that the signal originates from an access point on another floor, so either from a hallway above or below. Still even though the signal is obstructed it is evenly distributed. This is probably due to the fact that no matter where the robot recorded the signal on the hallway, the signal was always obstructed by walls and thus was weakened equally.

Now figure 4.3 paints a different picture. We can see a clear peak in the upper right corner of the room. And indeed the access point is installed there behind the wall. But the signal doesn't weaken gradually like it was on the floor. Even

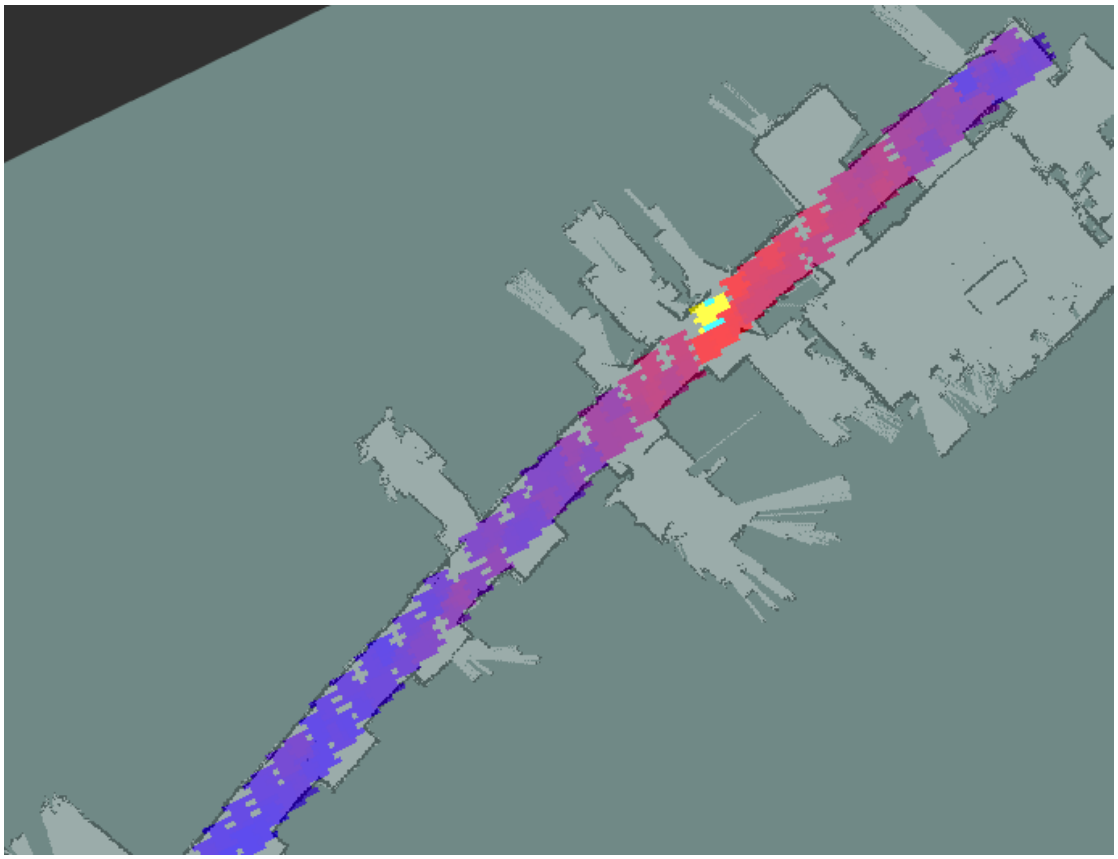

FIGURE 4.1: Visualization of Wi-Fi data collected on a hallway. The access point the signal originated from is on the same floor.

though in its entirety it does indeed get weaker the further away it was recorded, there are local peaks in different places.

This is probably due to the nature of this room. There are tables, chairs and a couch, which can all lead to the obstruction of the signal. Especially because the recording robot was relatively small.

4.1.2 Wi-Fi Position Estimation Accuracy

For the Wi-Fi position estimation to be actually useful it is important that it is accurate to a certain extent. It is not necessary to achieve accuracy on the level of a few centimeters, but it can only be useful if it can reduce the possible positions to a radius of a few meters. The first experiment was designed to compare the actual position of the robot with the estimated position from the Wi-Fi data.

To determine the actual position AMCL is used. The position was determined at the start of the localization, so that it is as accurate as possible. The test environment was a long hallway. The hallway can be seen in figure 4.1 This would be the kind of environment where the Wi-Fi position estimation could give an

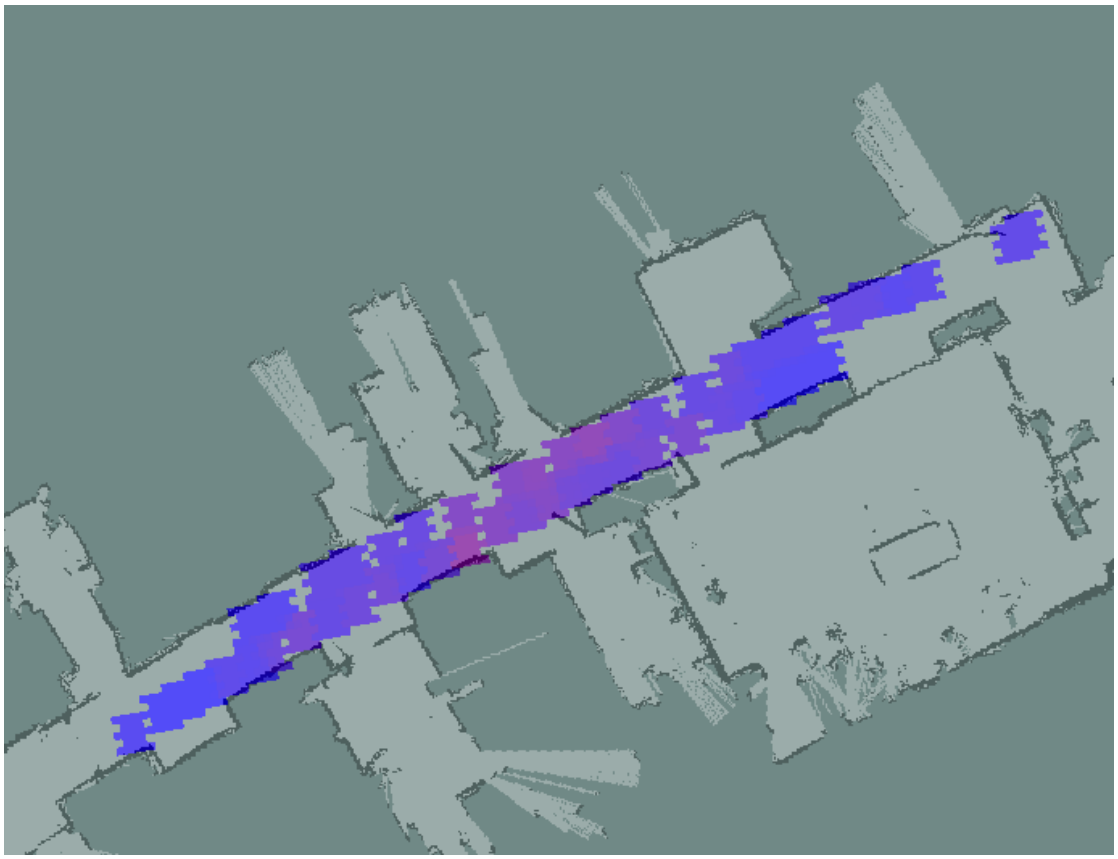

FIGURE 4.2: Visualization of Wi-Fi data collected on a hallway. In this example the access point is on a different floor.

TABLE 4.1: Error of the Wi-Fi position estimation in meters.

Try:	1	2	3	4	5	6	7	8	9	10
Error:	3.75	1.32	0.63	2.72	1.42	3.88	0.25	1.86	3.27	1.55
Try:	11	12	13	14	15	16	17	18	19	20
Error:	1.11	0.38	0.49	0.37	0.97	4.25	1.08	0.38	6.36	0.18

advantage over the usual approach to global localization because there aren't any characteristics that stand out and there is a lot of symmetry.

The robot drives to different points on the map that were chosen beforehand. When it arrived the robot will execute the Wi-Fi position estimation. The result will be compared with the coordinates given by AMCL. The difference between the two results will be computed.

The result of the experiment was, that on average the difference to the ground truth was 1.81 meters. The individual results can be seen in table 4.1.

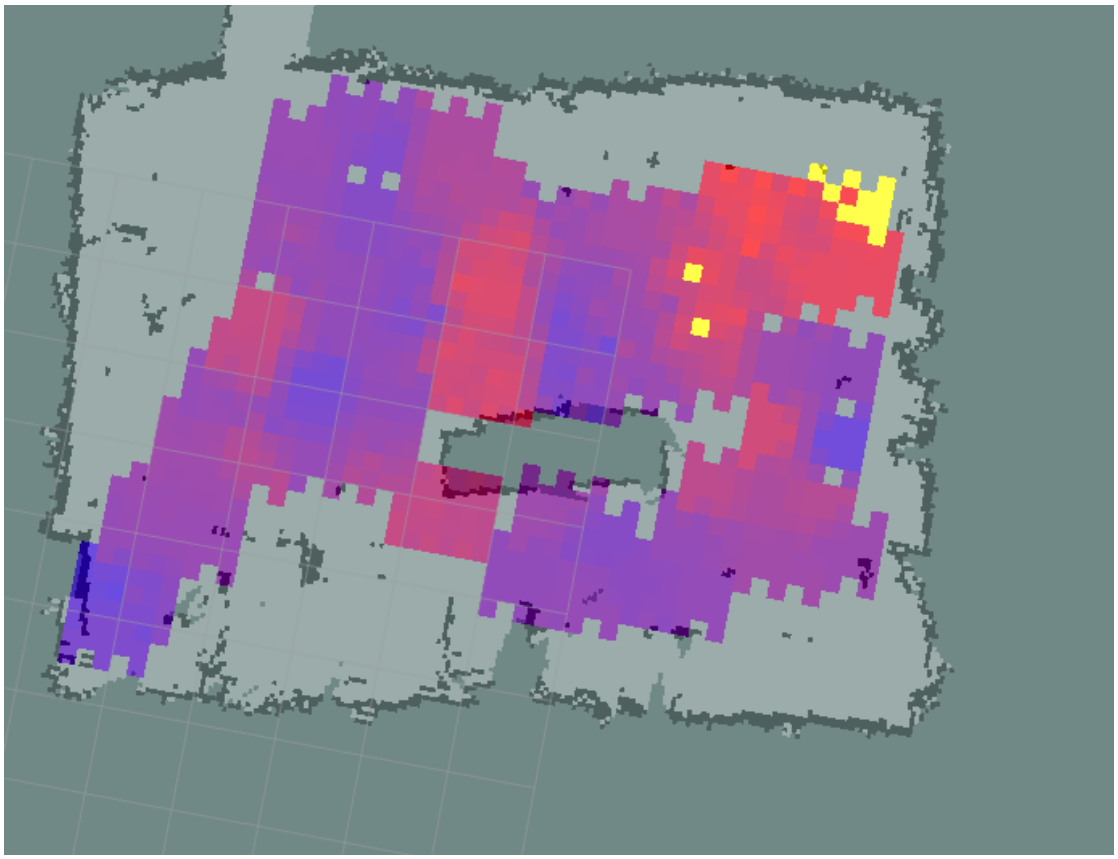

FIGURE 4.3: Visualization of Wi-Fi data collected in our laboratory. This is a room with many potential obstructions of Wi-Fi signals. This is reflected here by the way the Wi-Fi signal strength behaves. The access point is behind the wall of the upper right corner, near the maximum signal strength marked by the yellow color. It can be clearly seen that the signal is not getting gradually weaker.

4.1.3 AMCL with Wi-Fi Position Estimation

Global Localization The proposed potential use for the position estimation was that it could give a starting position for AMCL. This experiment will answer in how far the position estimation can aid AMCL. Once again a hallway is used. A set of goals for the robot to drive to is used. When a goal was reached the two different methods for global localization are compared.

One method uses the typical global localization where the particles are spread over the entire map and then converge to one pose. The other method will use the Wi-Fi position estimation as a seed and will lead to a much smaller area for the particles to be spread out in.

When the estimation was performed the robot will turn 360 degrees and then drive to the next planned goal. From there the procedure is repeated.

TABLE 4.2: Error in meters of the global localization using the Wi-Fi position estimation.

Try:	1	2	3	4	5	6	7	8	9	10
Error:	3.63	0.01	0.08	7.15	7.85	1.83	0.05	6.48	0.01	2.04
Try:	11	12	13	14	15	16	17	18	19	
Error:	0.04	0.17	5.14	0.13	0.02	4.8	0.06	2.82	0.03	

TABLE 4.3: Error in meters of the global localization using the standard approach from AMCL.

Try:	1	2	3	4	5	6	7
Error:	20.31	34.11	9.89	7.70	29.07	28.31	0.15
Try:	8	9	10	11	12	13	14
Error:	9.68	5.09	20.22	5.24	9.04	10.34	14.38
Try:	15	16	17	18	19		
Error:	24.52	34.95	25.46	5.50	35.44		

At the arrival at the next goal the difference to the actual position was measured. Using the Wi-Fi position estimation the mean error was 2.23 meters. With the global localization the mean error was 17.34 meters.

While observing the global localization using the Wi-Fi position estimation when executing the experiment some notable behaviors occurred. All in all in 9 of 19 tries the robot was able to approximately find the correct location. In some cases the failure could be explained by the fact that the Wi-Fi position estimation was too far away from the true position. In other cases the robot localized itself in the wrong direction. Of course the reason for this is that with the Wi-Fi data we are not able to infer any information about the orientation of the robot. The hallway looks very similar on both sides, so it is easy to reconstruct as to why the localization was failing. In figure 4.2 one can see that on one side of the hallway the wall is built straight, while on the other side in a certain interval the wall gives way to a bit more space. So sometimes it happened that the robot was localizing itself next to one of these spaces, even though it was actually next to the space before or behind it. So even when we are able to narrow down the possible position of the robot on the map, the laser range finder based localization can still get confused by the structure of the room.

The global localization implemented in the AMCL package was mostly able to localize the robot on the hallway, but in just one case it was able to approximately localize the robot on the right position. The obvious reason is that the hallway's outline is very similar in a lot of places. The Wi-Fi position estimation had similar problems, but there the part of the hallway that was taken into account was already very small. AMCL spreads its particles over the whole map when starting the global localization, so naturally these problems are magnitudes higher too.

Kidnapped Robot Problem Another case where the position estimation can be used in AMCL, is when the robot was picked up while localizing and brought to a different position without noticing it. This kind of problem is called the kidnapped robot problem.

In this experiment the robot will drive up and down the hallway. The values from equation 2.10 are used to compare the weights in the long term, with the weights in the short term. This means the robot needs to be localized correctly for some time, so that it can be detected that the robot was kidnapped. So at first the robot will drive up and down the hallway with the help of AMCL using the laser range finder. We will record this with a rosbag. Once finished the rosbag can be replayed and a new instance of AMCL can be applied to the recorded data. At first we will initialize the new instance of AMCL to the correct location. After driving up and down for some time AMCL will be given an initial position at a random point on the map. So this will result in a localization failure. Meanwhile AMCL still gets the data from the robot driving up and down the hallway from the rosbag. Since AMCL now has the wrong location, the "/amcl_failure_probability" value should rise and signal this localization failure or kidnapping of the robot.

We now want to know if the Wi-Fi position estimation would be triggered. For that to happen we set a threshold of 0.1 for the "/amcl_failure_probability" topic. This value is computed by AMCL as explained in section 2.1.7.

In 10 tries the Wi-Fi position estimation was triggered every time by the time the robot was at the other end of the hallway. The method that is used is the same as in the previous experiment, so a new the new computed position is given as a seed for AMCL. The accuracy of this mechanic was already discussed in the previous chapter.

4.2 Chapter Summary

We looked at the Wi-Fi data and its behavior. When there are are no obstacles in between the access point and the robot, the signal gradually gets weaker the further the robot is away. In rooms with many obstructions we saw that the signal is less predictable. The first experiment was designed to measure the Wi-Fi position estimation's accuracy, that resulted in an average error of about 1.81 meters. In the next experiment we used this mechanic to enhance the global localization of AMCL. The computed position is given as a seed for AMCL. The result was an improvement over AMCL's usual global localization, with an error of 2.23 meters with using the Wi-Fi position estimation compared to an error of 17.34 meters without it. The last experiment was designed to test the localization failure detection. It worked in 10 out of 10 tries. In the following chapter we will evaluate the results of the experiments and draw our conclusions from them.

Chapter 5

Conclusion

The Wi-Fi position estimation can be realized with Gaussian processes. The results of the first experiment, where the accuracy of the Wi-Fi position estimation was tested, was a mean error of around 1.81 meters to the ground truth. This value is low enough to use the Wi-Fi position estimation for the global localization. It is able to provide a rough estimate of the location beforehand.

In the next experiment we determined how suitable the Wi-Fi position estimation is for the use as global localization. The result of the Wi-Fi position estimation was used as an initial position for the Monte Carlo localization with a laser range finder. The result was that on average the position differed only 2.23 meters from the ground truth. This is a big improvement over the global localization provided by AMCL, which had a mean error of 17.34 meters. In 9 out of 19 cases the global localization using the Wi-Fi position estimation was able to approximately find the true position. This too is a big improvement over the global localization from AMCL, that was only able to do this in 1 out of 19 cases.

The experiments were carried out on a hallway and especially in the second experiment the problems laser range finders can face in such in an environment became apparent. There were no outstanding landmarks in the hallway. Many parts of the map look very similar. So when AMCL was using the global localization it could happen that the robot was not able to distinguish one end of the hallway from the other end. In other cases the robot was localizing itself in the wrong part of the hallway, because many parts of the hallway's outline are very similar as well. This symmetry and lack of any outstanding features in the outline were the main causes for the difficulties AMCL's global localization was facing. The global localization using the Wi-Fi position estimation was suffering from these problems as well, but to a lesser extent. So it happened that the robot would localize itself in the wrong direction or in a part of the hallway that was a few meters away from the true position. But the Wi-Fi position estimation was able to reduce the issues caused by those circumstances.

The method to detect if the robot was kidnapped was working well and was able to correctly identify the problem in 10 out of 10 tries. So using this the Wi-Fi position estimation would also be able to help to re-localize the robot on the correct position.

In practice we foresee some drawbacks to the Wi-Fi position estimation. First of all the Wi-Fi scan to fetch the signal strengths and MAC-addresses of the access points in the environment can take a few seconds. Then the computations involved in the Wi-Fi position estimation can take a few seconds depending on the number of access points observed, training data involved and the number of particles used. For the experiments on the hallway usually a large number of access points was observed and with 1000 particles the Wi-Fi position estimation was taking 30 seconds to compute an approximate location. This is fine for an initial global localization, because the robot without a localization would not know where to go anyway. In the case of a localization failure, when the Wi-Fi position estimation is triggered by the "/amcl_failure_probability", the robot would need to stop for the time of the computation of the new initial position as well.

Chapter 6

Outlook

The Wi-Fi position estimation precision is with a mean error of 1.81 meters in the experiment good enough, so that it can be used for the global localization for AMCL with a laser range finder. The Wi-Fi signals can only be used to infer information about the position, but not about the orientation of the robot. One could for example use a compass to get this kind of information. Right now the particles, that are spread by the Wi-Fi position estimation have a random orientation. Using a compass one could infer an approximate orientation and thus would be able to spread particles that have that orientation. When carrying out the second experiment where the Wi-Fi position estimation was used for the global localization we observed that the robot would often be localized in the wrong direction on the hallway. So using a compass could clearly lead to an improvement in this case.

Another concern is the computation time of the Wi-Fi position estimation. When the global localization using the Wi-Fi position estimation is executed random positions on the map are generated that are used as particles. So every Gaussian process for each of the networks has to compute a Gaussian distribution for all those particles. Depending on the number of particles, size of the training data and the number of observed Wi-Fi networks in the moment, this can cause long computation times. One solution would be to pre-compute the Gaussian distributions when the Wi-Fi position estimation node is initialized. So the random positions are all generated beforehand and when the Wi-Fi position estimation is actually called these randomly generated positions are used. The obvious consequence here is that in case we only generate the random positions once, the position of the particles would not change in subsequent executions. A solution would be to draw a new set of random positions after a Wi-Fi position estimation was performed and calculate the new Gaussian distributions then.

Right now the robot has to stand whenever the Wi-Fi position estimation is executed. The reason for this is the already mentioned computation time, but another reason is also the time the Wi-Fi scan takes. A full scan usually takes a few seconds. The different Wi-Fi channels are scanned one after the other. When the robot moves in the few seconds that the scan is taking, then the data from the

different channels is spread over the path the robot moved on. So we do not know the actual position the specific networks where actually recorded on. But as we explained it is also possible to scan the different Wi-Fi channels separately. This is a lot faster then the whole scan, but we will also fetch less Wi-Fi networks. This makes some optimizations possible. First off we would be able to scan the different channels one after the other, but get the data from each channel instantly instead of waiting for scanning all the other channels. This would give us the ability to process the data when it is only a fraction of a second old. Another possible optimization would be to only scan a selection of Wi-Fi channels. The command used for the Wi-Fi publisher also carries information about the channel used for a network. So we would only need to scan those particular channels.

This can also help to implement the Wi-Fi sensor model directly into AMCL. So then the computed weights could be directly applied to the particles from AMCL. This can just be done by multiplying the weights computed with the measurement model for the laser range finder with the weights computed by the Wi-Fi receiver sensor measurement model. One problem here will be that the weights produced by the Wi-Fi sensor model can differ from iteration to iteration. Their value is dependent on the number of Wi-Fi access points observed. So they would have to be normalized somehow before used with AMCL. When the channels are scanned individually the data can be used instantly to compute the weight, instead of waiting a few seconds for the scan of all channels to end. This could also help with the computing time. When we scan the channels individually we observe less access points and therefore less computation time is needed for the Gaussian processes. Another factor is the number of particles. AMCL varies the number of particles. When they are clustered around one position there are less particles compared to the start of a global localization when the particles are spread over the entire map. Less particles would lessen the computation time as well.

Another factor for the computation time is the size of the training data set. If we would find a strategy that would be able to reduce that set while keeping the accuracy of the Wi-Fi position estimation and the quality of the results from the Wi-Fi sensor model relatively stable, then this would also help to reduce the computation time.

We used the weights computed by AMCL using the laser range finder for the detection of the kidnapped robot problem. One possible way to use the Wi-Fi position estimation for the detection would be to approximate the position using it and when the position computed by AMCL is further away than a certain threshold then AMCL is re-initialized using the Wi-Fi position estimation.

Appendix A

Code Overview

This is an overview over the code created for this thesis. The tree shows folders and files. In parenthesis behind the files is the number of lines the file contains.

```
wifi_localization
├── include
│   ├── csv_data_loader.h (31)
│   ├── wifi_data_collector
│   │   ├── mapcollection.h (70)
│   │   ├── mapdata.h (137)
│   │   └── subscriber.h (142)
│   └── wifi_position_estimation
│       ├── gaussian_process
│       │   ├── gaussian_process.h (119)
│       │   ├── kernel.h (68)
│       │   └── optimizer.h (41)
│       └── wifi_position_estimation.h (90)
├── launch
│   ├── kidnapped_robot_experiment.launch
│   ├── pos-est-accuracy-experiment.launch
│   ├── wifi_data_collector_csv.launch
│   ├── wifi_data_collector.launch
│   ├── wifi_data_collector_record_only_stopped.launch
│   └── wifi_position_estimation.launch
└── msg
    ├── MaxWeight.msg
    ├── WifiPositionEstimation.msg
    └── WifiState.msg
```

```
wifi_localization
├── scripts
│   ├── wifi_publisher.py (72)
│   ├── wifi.py
│   └── wifi.pyc
└── src
    ├── csv_data_loader.cpp (50)
    ├── experiments
    │   ├── map_traverser_node.cpp (137)
    │   ├── wifi_pos_est_accuracy2_node.cpp (95)
    │   ├── wifi_pos_est_accuracy_node.cpp (40)
    │   └── wifi_pos_est_kidnapping_node.cpp (182)
    ├── wifi_data_collector
    │   ├── mapcollection.cpp (105)
    │   ├── mapdata.cpp (209)
    │   ├── subscriber.cpp (109)
    │   └── wifi_data_collector_node.cpp (48)
    └── wifi_position_estimation
        ├── gaussian_process
        │   ├── gaussian_process.cpp (152)
        │   ├── kernel.cpp (51)
        │   └── optimizer.cpp (59)
        ├── wifi_position_estimation.cpp (232)
        └── wifi_position_estimation_node.cpp (21)
```

Bibliography

(2012). *Scanning Laser Range Finder UTM-30LX/LN Specification*. Hokuyo Inc. Ltd.

Andersen, M., Jensen, T., Lisouski, P., Mortensen, A., Hansen, M., Gregersen, T., and Ahrendt, P. (2015). Kinect Depth Sensor Evaluation for Computer Vision Applications. *Technical Report Electronics and Computer Engineering*, 1(6).

Biswas, J. and Veloso , M. (2010). Wifi Localization and Navigation for Autonomous Indoor Mobile Robots. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 4379–4384.

Blum, M. and Riedmiller, M. (2013). Optimization of Gaussian Process Hyperparameters using Rprop. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 339–344.

Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999). Monte Carlo localization for mobile robots. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 1322–1328 vol.2.

Duvallet, F. and Tews, A. D. (2008). WiFi Position Estimation in Industrial Environments Using Gaussian Processes. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2216–2221. IEEE.

Fernandez, E., Crespo, L. S., Mahtani, A., and Martinez, A. (2015). *Learning ROS for Robotics Programming - Second Edition*. Packt Publishing, 2nd edition.

Ferris, B., Hahnel, D., and Fox, D. (2006). Gaussian Processes for Signal Strength-Based Location Estimation. *Proc. Robotics: Science and Systems*, 442:303–310.

IEEE (2012). Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Std 802.11-2012, IEEE.

Ito, S., Endres, F., Kuderer, M., Tipaldi, G. D., Stachniss, C., and Burgard, W. (2014). W-RGB-D: Floor-Plan-Based Indoor Global Localization Using a Depth Camera and WiFi. In *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, pages 417–422.

Levinson, J., Montemerlo, M., and Thrun, S. (2007). Map-Based Precision Vehicle Localization in Urban Environments. In *Robotics: Science and Systems III, June 27-30, 2007, Georgia Institute of Technology, Atlanta, Georgia, USA*.

Liu, D. C. and Nocedal, J. (1989). On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical programming*, 45(1-3):503–528.

Misra, P. and Enge, P. (2011). Global Positioning System: Signals, Measurements and Performance Second Edition.

Pinheiro, P., Cardozo, E., Wainer, J., and Rohmer, E. (2015). Cleaning Task Planning for an Autonomous Robot in Indoor Places with Multiples Rooms. *International Journal of Machine Learning and Computing*, 5(2):86–90.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*.

Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Serrano, O., Rodero Merino, L., Matellán Olivera, V., and Cañas, J. M. (2004). Robot localization using WiFi signal without intensity map. In *V Workshop de agentes Físicos*, pages 79–88.

Shewchuk, J. R. (1994). An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, Pittsburgh, PA, USA.

Tews, A., Pradalier, C., and Roberts, J. (2007). Autonomous Hot Metal Carrier. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1176–1182.

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

Weitkamp, C., editor (2005). *Lidar (Range-Resolved Optical Remote Sensing of the Atmosphere)*. Springer-Verlag New York.

Selbstständigkeitserklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Bachelorstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel — insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen — benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ich bin mit der Einstellung der Arbeit in den Bestand der Bibliothek des Fachbereichs Informatik einverstanden.

Hamburg, den 10. November 2016

_____

Benjamin Scholz