Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

BACHELORTHESIS

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

vorgelegt von

Benjamin Scholz

Fakultät für Mathematik, Informatik und Naturwissenschaften

Fachbereich Informatik

Studiengang: Bachelor of Science Informatik

Martikelnummer: 6428073

Erstgutachter: Dr. Norman Hendrich

Zweitgutachter: Lasse Einig

Abstract

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Contents

Chapter 1

Introduction

1.1 Motivation

Localization is an important problem in robotics. It is needed in order to solve a wide range of problems. Even simple tasks in households require a robot to be able to determine its own position in the environment. It needs to be able to find from point A to point B. Being able to do this, a robot could for example bring a fresh cup of coffee and bring 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.

The Monte Carlo Localization[Dellaert et al., 1999] is one of the most widely used algorithms to solve the localization problem. It has proven itself to be stable and reliable in a wide range of situations. As [Levinson et al., 2007] showed it was even suitable for use in autonomous vehicles to enhance the location estimation precision using a LIDAR, GPS and odometry. In the case of the autonomous vehicle it is reasonable to expect that GPS will be available most of the time. It can give the vehicle a good estimation even though it can be a few meters off. In the case of the autonomous car the localization happens on a huge map and the GPS estimation is very helpful for the Monte Carlo Localization. Monte Carlo Localization works best when the position at the beginning is already known or if it at least gets a rough estimate. There are approaches to localize the robot with no prior position estimation, for example by taking the whole map into account instead of just a certain position, in the beginning. This is called global localization.

There are many situations where a robot can't rely on GPS. In many buildings its signal is obstructed. Now one could use global localization in the beginning. But this can cause problems. On many maps there just aren't enough unique landmarks to quickly figure out the robot's pose. So in many cases the robot is

not able to localize itself correctly. Once a wrong pose was determined by the localization, it can be difficult to recover from such a failure.

There are some environments that heighten the chance of the occurrence of such problems. In a long hallway, for example, many parts of the outline look similar. Also they are often symmetric and that can cause additional problems, because suddenly parts of the map are indistinguishable from each other. Also many buildings have different rooms with the exact same layout. Another situation where the global localization could fail would be when the laser scanner doesn't reach any obstacles. The robot would need to drive around and find them in order to actually localize itself.

In most buildings there are a lot of different Wi-Fi 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 Monte Carlo 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

As already mentioned 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., 2012] 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 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 processes 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 to build the map. 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. [Duvallet and Tews, 2008] use this approach to estimate the position of industrial vehicles. Here they used the Wi-Fi position estimation as a seed for a localization system using laser scanners. So for example in case the system fails and the computed location is inaccurate it uses the Wi-Fi position estimation for a rough estimate and a new starting point for the localization system. [Ito et al., 2014] had a similar approach where they used Wi-Fi data for initialization for a global Monte Carlo localization with a RGB-D sensor. Both [Ito et al., 2014] and [Duvallet and Tews, 2008] use Gaussian processes to estimate a position via Wi-Fi.

Chapter 2

Basics

2.1 Localization

2.1.1 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. Any place where people live will change constantly just by moving from one place to another. And even the robot itself changes its own environment by acting in it. Robots usually have a variety of sensors. They are used to make sense of the robots 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 100% exact. When a laser scanner measures the distance to an obstacle we have to expect that there will be an error margin. In localization 2d maps get used very often. They are a model of the real world. But once again it is only an inexact 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]

2.1.2 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, coming from sensors like cameras, laser range scanners or Wi-Fi receivers. The second kind of data gives us information about the change of state and could for example be the velocity or alternatively data from an odometer. [Thrun et al., 2005, p. 22-23]

The current state is denoted x_t. At this point we will rely on probability for the reasons already explained. So the probability of state x_t is the following: $p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t})$. This takes all previous states and data 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 same assumption is behind the following equation:

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

2.1 is known as the state transition probability and 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 reflect 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.

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

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]

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 simply a constant, 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, it will always stay the same.

$\overline{bel}(x_t)$ 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 the 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**

Algorithm 1 is the foundation of the Monte Carlo localization.

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. Then the belief gets updated. First with the control data. This makes sure that the change of state gets reflected in the distribution before we apply our knowledge from the environment data to our distribution. Then in line 4 the belief gets updated with the measurement data too. This way we applied all the data that we have and thus can infer the state from the belief distribution.

Figure 2.1 shows a simple example how the algorithm could be used for localization. It reflects the nature of the control data and measurement data. Whenever the robot moved and the control data is used to update 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

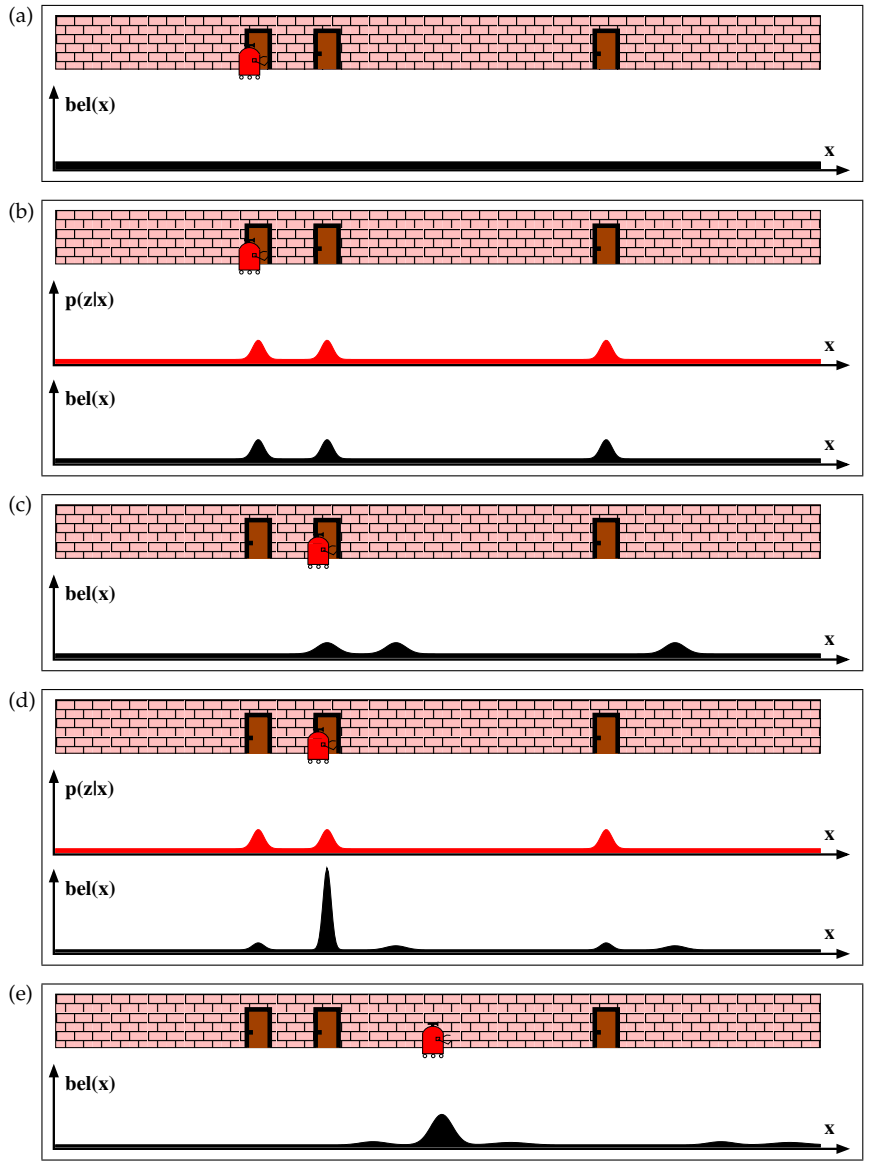

FIGURE 2.1: A simple example to explain how Bayes filter works. In (a) the belief is evenly distributed. In (b) the robot observes the door and the measurement data reflects this and updates $bel(x)$ accordingly. In (c) the robot moves to the right and the beliefs structure shifts to the right as well, 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]

time interval the uncertainty induced by moving around is small enough that we can compensate for it by using the measurement data.

This is also a good example for why we use probabilities. In figure 2.1 in (b) the robot senses a door, but 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 time steps before get taken into account. The spikes move according to the control data. Now the 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.

2.1.3 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]

The posterior distribution, so the particles, are 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]

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 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.

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

1: **procedure** PARTICLE_FILTER(X_{t-1}, u_t, z_t)
2: $\bar{X}_t = X_t = \emptyset$
3: **for** $m = 1$ to M **do**
4: sample $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} = p(z_t|x_t^{[m]})$
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**

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.

2.1.4 Monte Carlo Localization

The particle filter can be used for localization.[Thrun et al., 2005, p. 252] Figure 2.2 shows a simple example. This figure is similar to figure 2.1. 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 ground truth. 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 gets explained in the following sections.

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

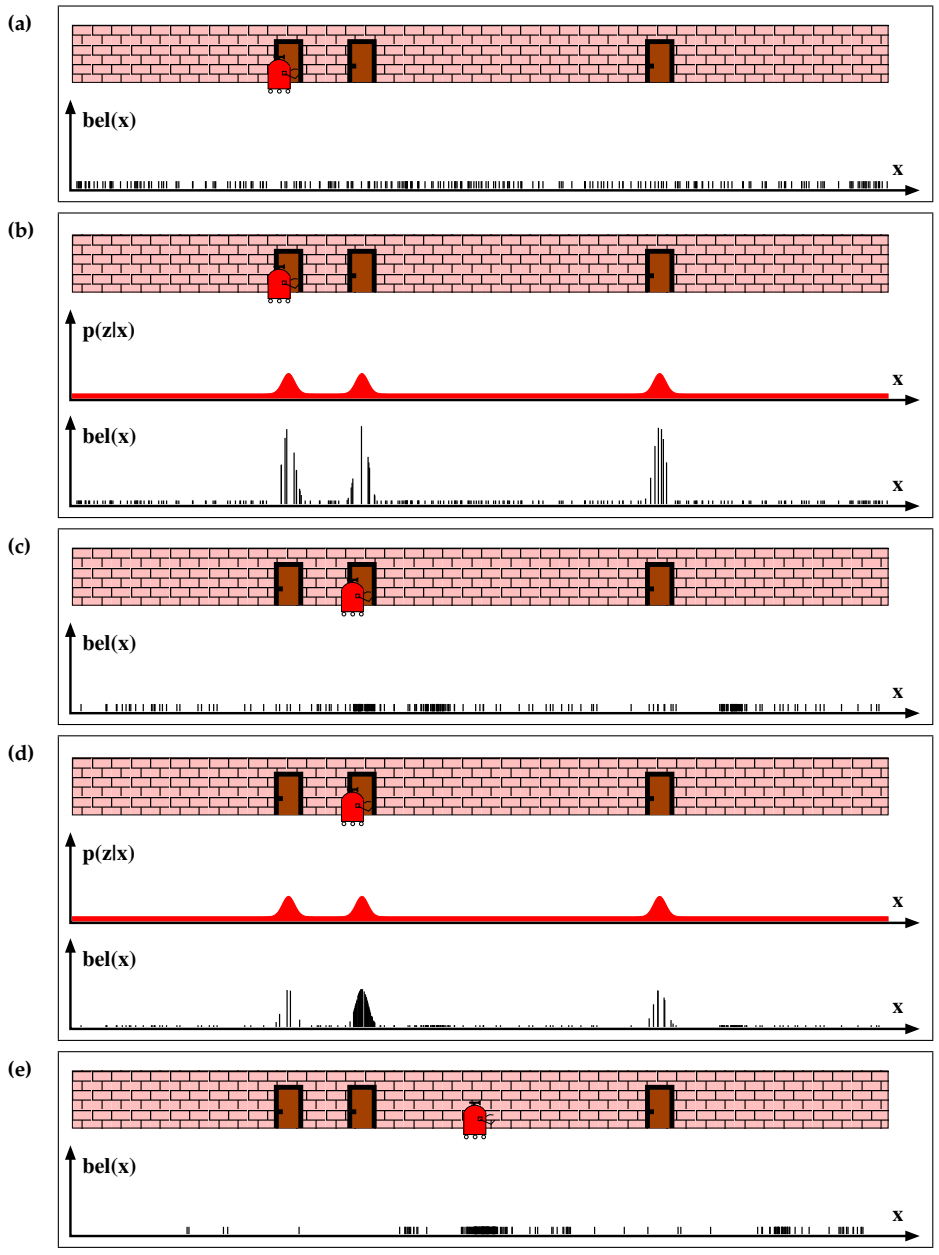

FIGURE 2.2: This is a simple example of the localization realized with a particle filter. It is similar to figure 2.1, but this time the belief is not a curve but instead represented by the distribution of particles. [Thrun et al., 2005, p. 251]

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 doesn't take in account the measurement data 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 state is representative of the real state.

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 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**

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 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.

2.1.5 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 task then is just to track the position from there on out. In the global localization problem we still have to determine the position.

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. And in many situations

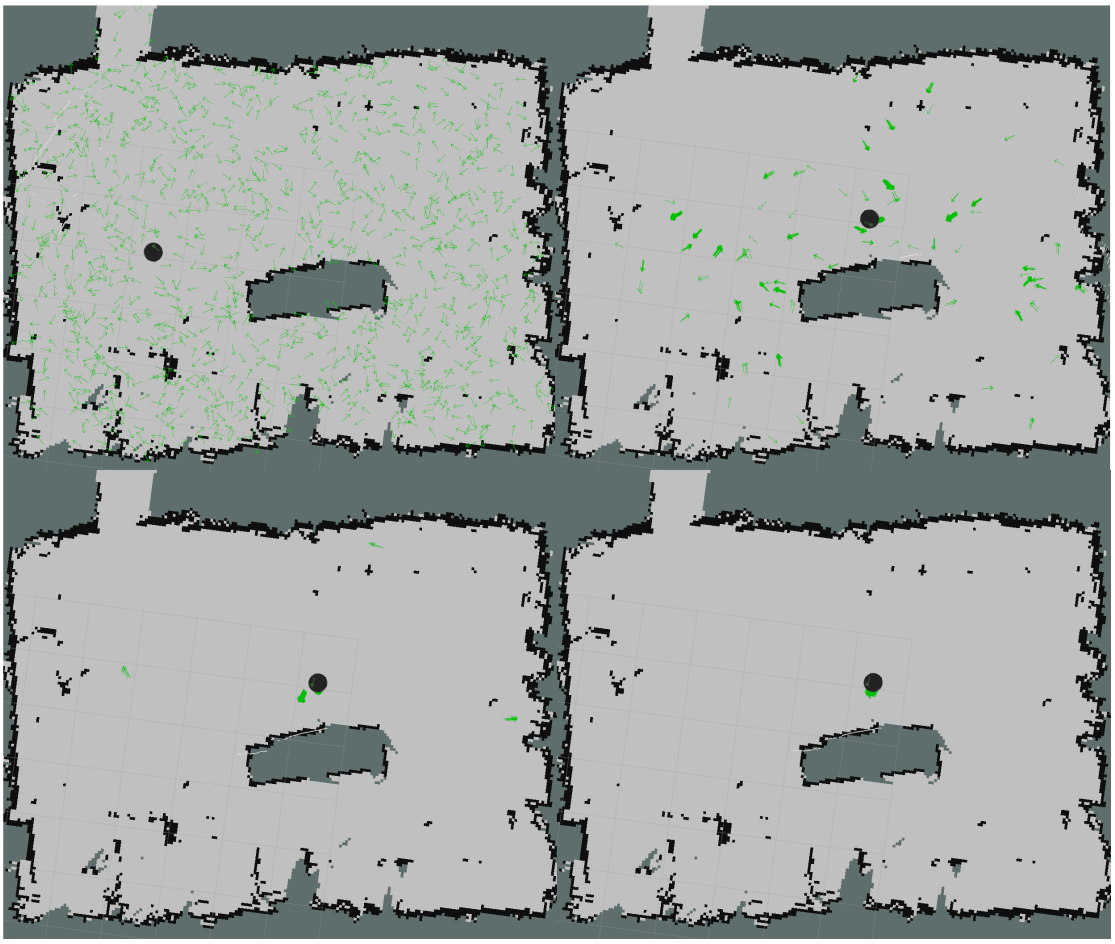

FIGURE 2.3: Example of global localization. At first the particles are spread on the entire map. The robot rotates to scan the environment. The particles converge to the correct position.

this works just fine. There are some situations where this can pose a problem though.

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.

2.1.6 Kidnapped Robot Problem

The kidnapped robot problem occurs when 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.

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}.

Then 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. So 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 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.

The Wi-Fi signal strength is measured in dBm. The theoretical minimum is -100 dBm and the maximum is -10 dbm. But these values are practically never achieved in real life. But generally anything over -50 dBm is a great signal, until -70 it is still a fair signal and anything under -70 can be considered weak to unusable.

So the goal is to fetch the data from every available Wi-Fi signal 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

In order to create a measurement model for the Wi-Fi receiver we choose to create a map of the Wi-Fi signal strengths. For Wi-Fi signals this is different from creating an obstacle map 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 will be walls and other obstacles between the access point and the receiver.

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. 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 it 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? One needs to make previous assumptions about the function because otherwise all functions that crosses 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. 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 we can just ignore the infinite other points. This makes it computationally feasible to use Gaussian processes for regression. [Rasmussen and Williams, 2005, p. 2]

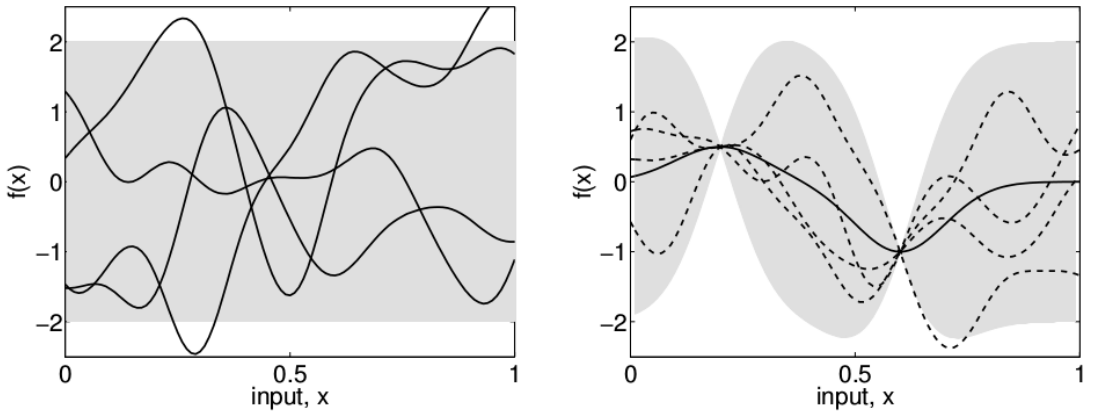

FIGURE 2.4: An example of how the prior distribution of functions works. In (a) functions were drawn from the prior. As one can see they have similar characteristics. In (b) the posterior is shown. Two points were observed. The solid line is the mean prediction. The shaded region is twice the standard deviation at each input value x. [Rasmussen and Williams, 2005, p. 3]

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 2.4. 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. Also the variance is lower the closer the input is too the observations as indicated by the gray shadows.

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 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 in a later section. In the following section we will define what a Gaussian process is and how it can be used for regression.

2.3.2.1 Regression

Regression is used to infer values from existing data. We are going to use it 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:

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

Here $f(\mathbf{x})$ is the function value. $\mathbf{w}$ is a vector of weights, that 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 belief 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 with.

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

$p(\mathbf{y}|\mathbf{X})$ 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, by 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 $K = K(\mathbf{X}, \mathbf{X})$.

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).

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

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.

2.3.2.2 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 compute values that are working well. In order to find those values we take the log-likelihood and maximize it.

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]

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

Some examples of algorithms that are suitable to optimize the hyperparameters are gradient descent, BFGS or L-BFGS. [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.

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 $\eta^+ > 0$ 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.

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.

Chapter 3

Implementation

3.1 ROS

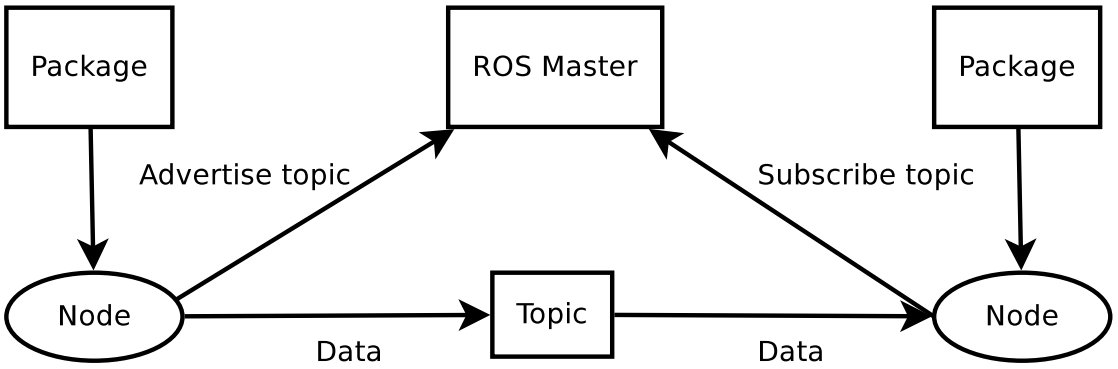

FIGURE 3.1: Relationship of the basic parts of the ROS environment.

The Wi-Fi position estimation was implemented with the ROS-Framework and run on a TurtleBot 2. ROS 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.

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 gets usually published constantly, no matter what.

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.

3.3 amcl

The amcl package implements various algorithms from [Thrun et al., 2005] and provides the possibility to localize the robot on a given map.(http://wiki.ros.org/amcl) It uses the following algorithms from [Thrun et al., 2005]: sample_motion_model_odometry, beam_range_finder_model, likelihood_field_range_finder_model, Augmented_MCL, and KLD_Sampling_MCL.

When started amcl publishes the pose of the robot determined by the localization. This pose contains the x and y coordinate on the given map and the orientation as a quaternion. The poses also contain a covariance matrix, that represents the spread of the particles around the determined position.

The package has multiple use cases in this project. For one it 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 2.1.6 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 only compute this value, but does not publish it, so that other nodes can use it. So amcl was slightly modified, so that it publishes that value under the identifier "/max_weight".

3.4 Data Collection

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,

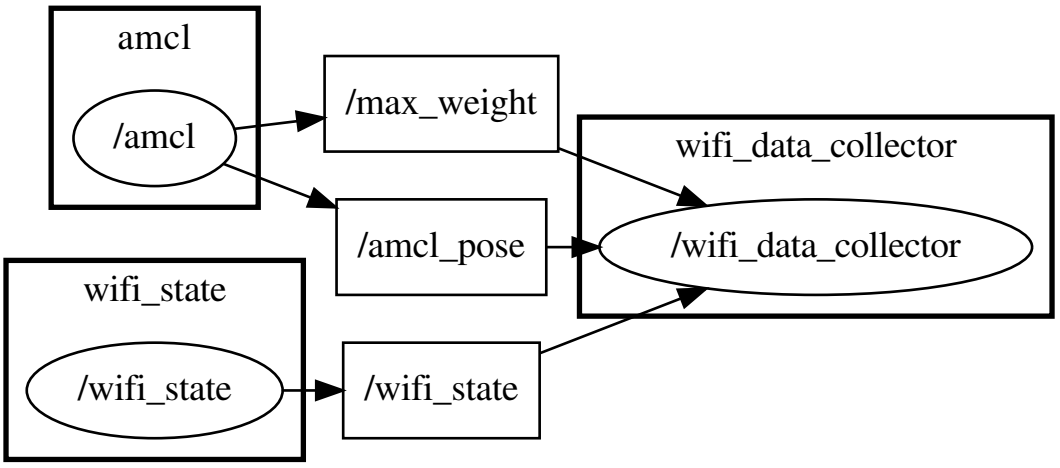

FIGURE 3.2: This shows the nodes and their topics the Wi-Fi_data_collector subscribed to and how it stores the data.

if we want to use the data later on to build a map of the signal strengths. Amcl publishes the needed pose of the robot and the Wi-Fi data publisher the needed 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 csv files. For each MAC-address there is a separate file.

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:

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

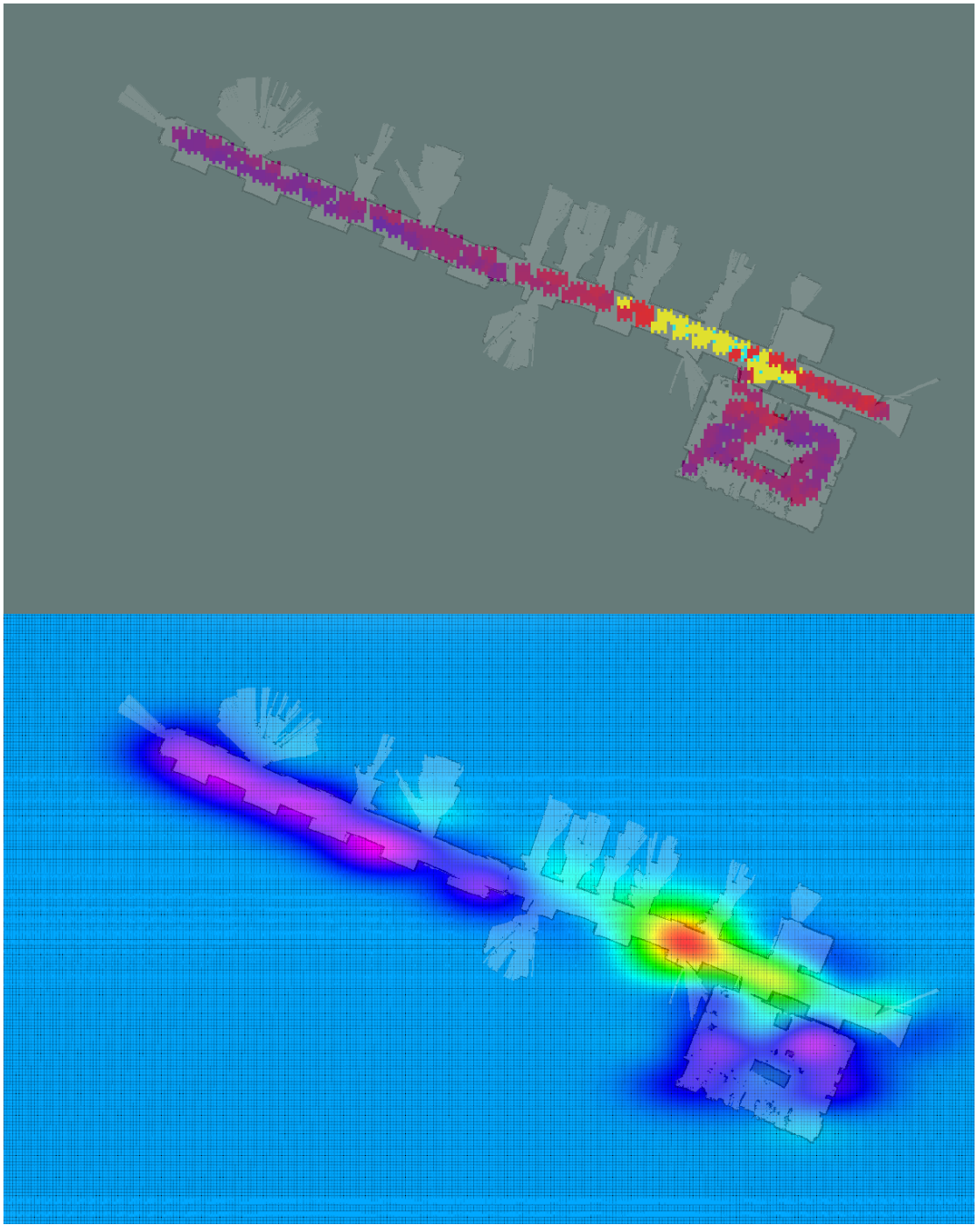

FIGURE 3.3: Above the recorded Wi-Fi signal strengths are shown. The lower image shows the mean of the corresponding Gaussian process.

This leads to changes 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, so 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.

3.6 Wi-Fi Position Estimation

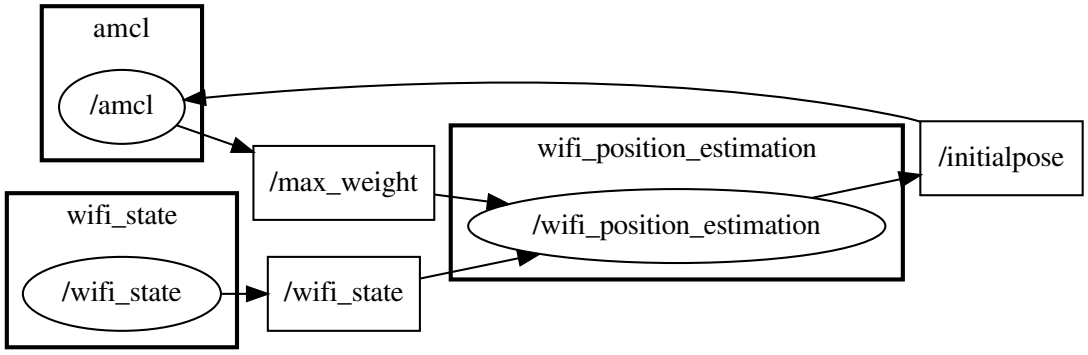

FIGURE 3.4: A basic diagram of the Wi-Fi_localization and the subscribed and published topics.

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

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.

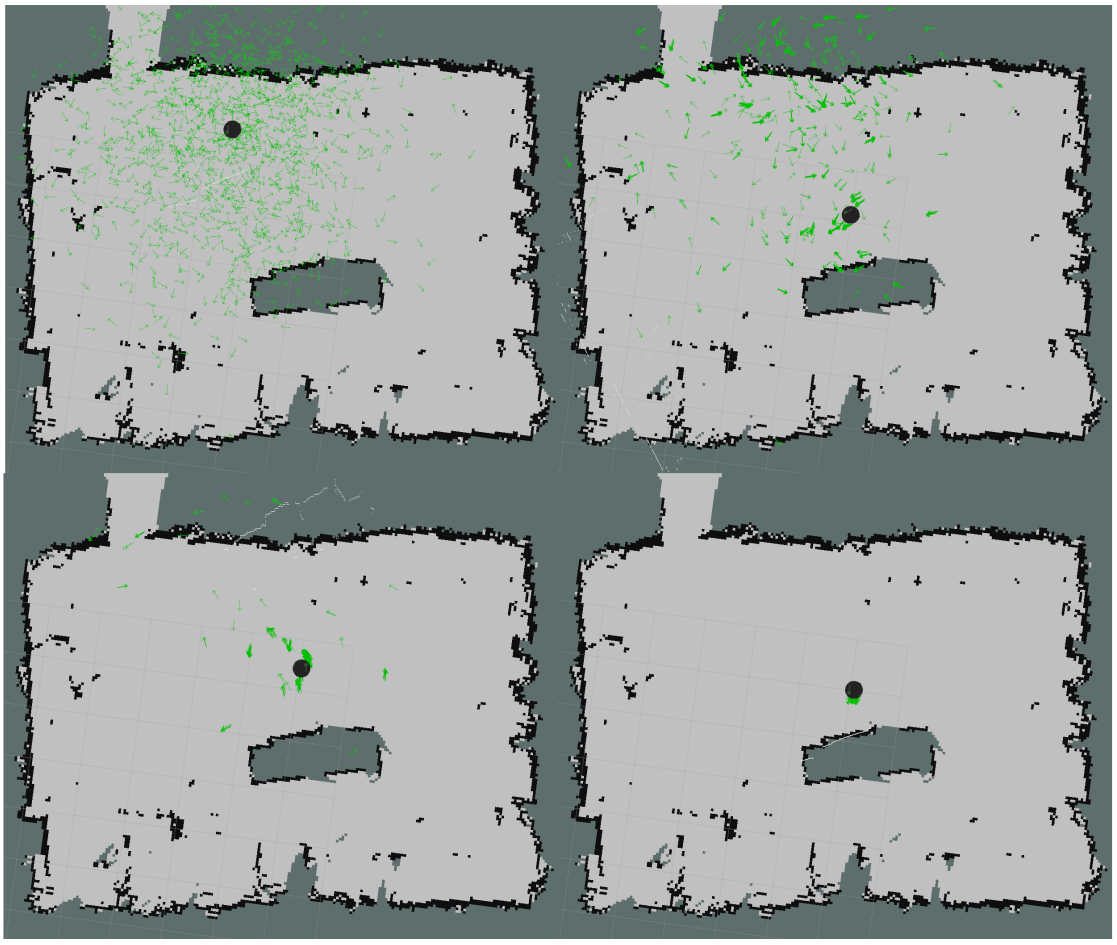

FIGURE 3.5: Example of using the Wi-Fi position estimation for global localization.

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:

$$w(x_*) = \prod_{i=1}^{n} (p(z_t^{[i]}|x_*)) \tag{3.4}$$

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 can use equation 2.11. Usually this is used to add new random particles to the set with the probability determined by equation 2.11.

Instead we will 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. Figure 3.4 shows that the Wi-Fi position estimation node is subscribed to "/max_weight". This value is the value used to determine if there is a localization failure.

Chapter 4

Experiments and Results

4.1 Data Analysis and Experiments

4.1.1 Wi-Fi data

First let's 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 pretty much no obstacles between the access point and the robot at any position. Turning to a different example from figure 4.2. As one can see 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 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.

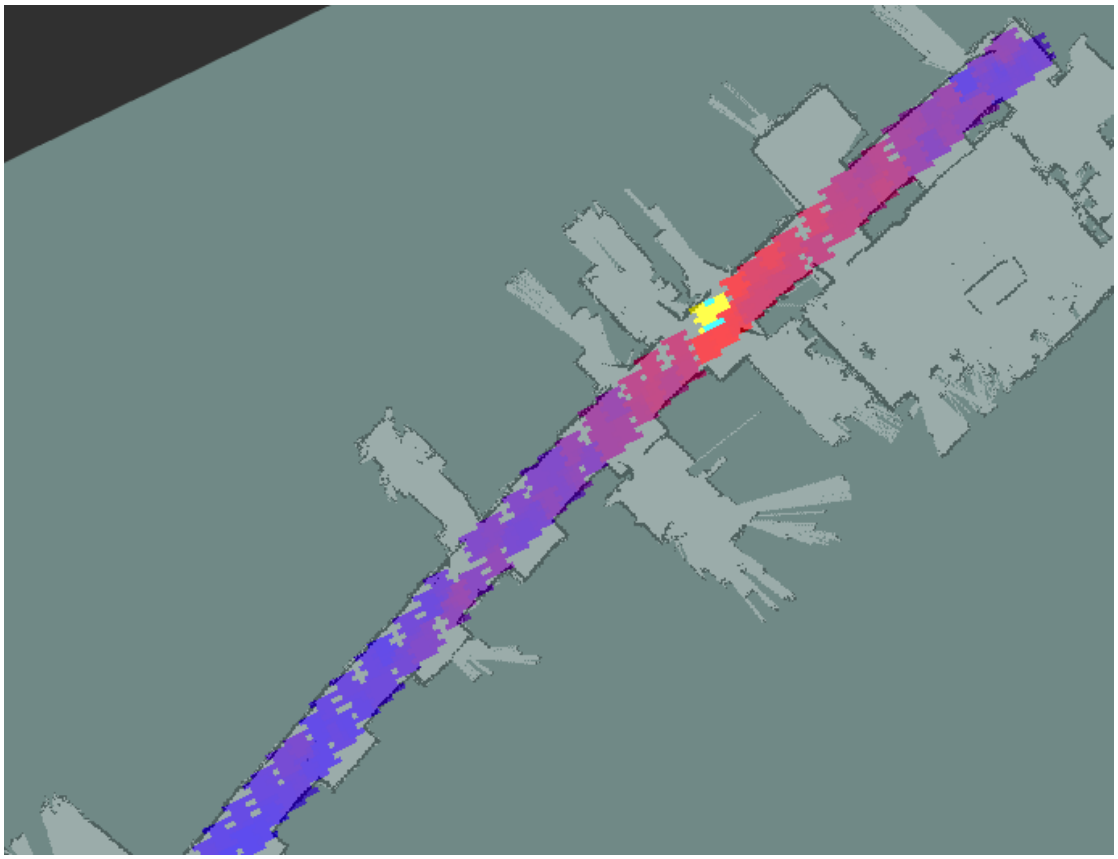

FIGURE 4.1: Visualization of Wi-Fi data collected on a hallway. The access point the signal originated from is on the same floor.

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 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.

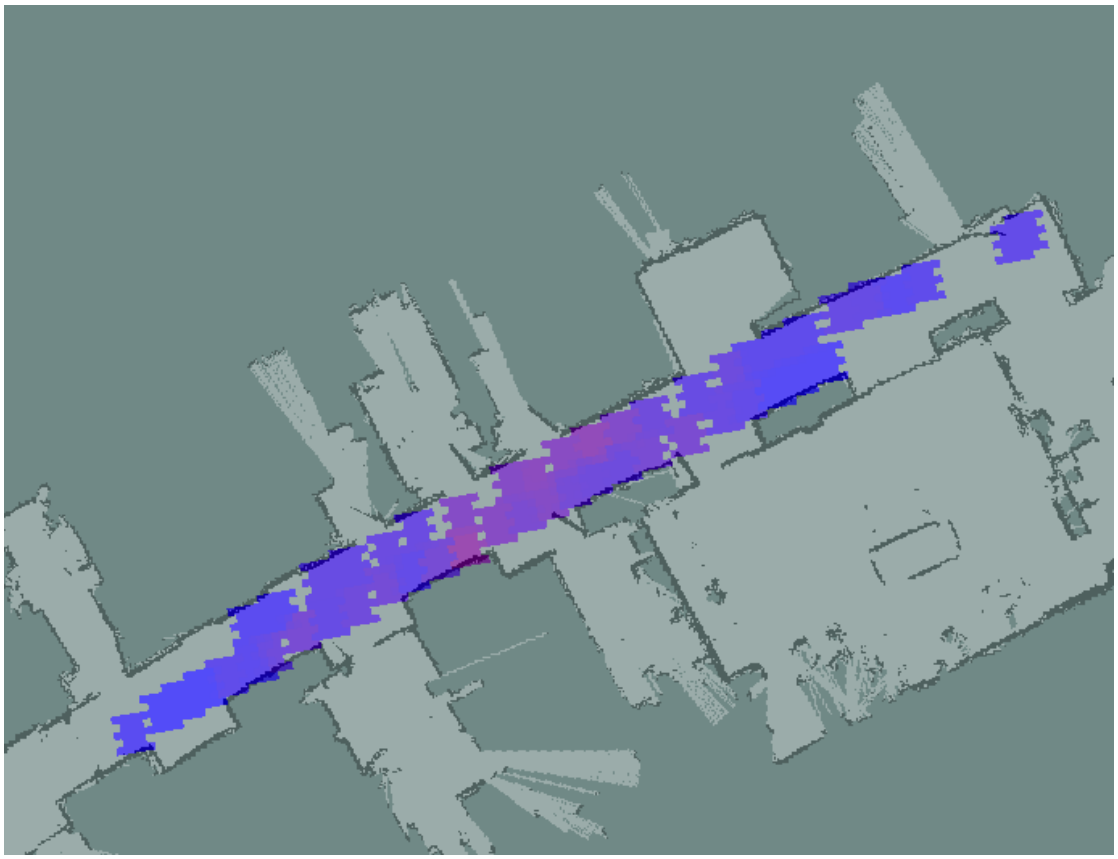

FIGURE 4.2: Visualization of Wi-Fi data collected on a hallway. In this example the access point is on a different floor.

The result of the experiment was, that on average the difference to the ground truth was 1.86 meters.

4.1.3 Amcl with Wi-Fi Position Estimation

4.1.3.1 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.

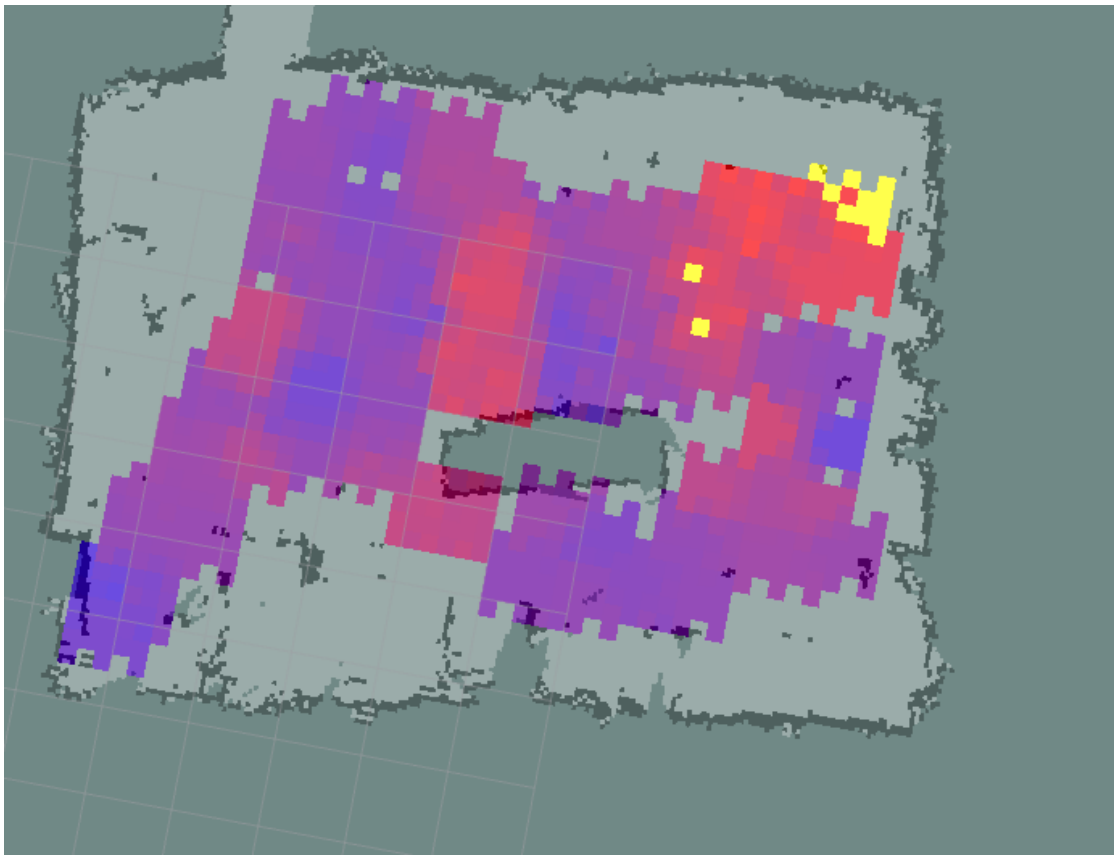

FIGURE 4.3: Visualization of Wi-Fi data collected in a room.

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.

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 2.40 meters. With the global localization the mean error was 15.39 meters.

So as one can see approximating the position beforehand via the Wi-Fi position estimation was a big advantage in an environment with no remarkable landmarks. The global localization was very far off from the true pose in most cases. But the Wi-Fi position estimation also had a notable difference to the ground truth.

4.1.3.2 Kidnapped Robot Problem

Another case where the position estimation can be used in amcl, is when the localization fails. 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 just drive up and down the hallway, with amcl localizing the robot. Then the position of the robot in amcl, will be set to a random new position. Now the robot will be localized on the wrong position.

Now the usual approach from amcl, explained in section 2.1.6, will be compared with the approach from the Wi-Fi position estimation, where the position estimation will be started when the value from equation 2.11 exceeds a certain value.

Chapter 5

Conclusion

5.1 Conclusion

The Wi-Fi position estimation can be realized with Gaussian processes. The results of the first experiment had a mean error of around 1.86 meters to the ground truth. As explained the position is computed by spreading particles over the map and then choosing the one with the highest weight. The Wi-Fi position estimation can be helpful for the Monte Carlo localization with a laser scanner. The experiment showed that in the case of global localization the Wi-Fi position estimation can lead to a more accurate position estimation. With the Wi-Fi position estimation the error to the ground truth after driving for a few seconds was 2.40 meters, while it was 15.39 meters for the global localization of amcl.

But of course there are drawbacks to the Wi-Fi position estimation. For one scanning for new data takes a few seconds. That means when the robot drives around the newest data for an access point can already be seconds old and thus be already meters away from the current position. Another problem is the computation time for the Gaussian processes. When using the Wi-Fi position estimation for global localization it can take a few seconds as well to get a result. The factors here are the number of access points available and the numbers of particles used. To use it for the global localization a large number of particles are needed. Of course this also depends on the size of the map. But once again when the robot drives around it can happen that the result of the Wi-Fi position estimation is already based on data that is tens of seconds old and thus based on a different position on the map.

These are limitations that will pose problems when the Wi-Fi position estimation is actually to be used in real world applications. As for the global localization it can still be usable. The robot needs to know its own position before driving around and the Wi-Fi position estimation can help with that. So it will not be a problem when the robot has to stand while computing it. In the case of a localization failure, so the kidnapped robot problem, the robot is usually already doing a task, so it is probable that the robot will drive around and as discussed this can pose a problem for the position estimation. So when the a localization failure is detected

the robot would need to stop its task and stand still until a position was computed and continue afterwards.

Another problem can arise when the Wi-Fi position estimation computes a position that is too far off from the true position. Since the detection for the localization failure is based on the difference between the short- and long-term average weights it likely that this failure will not be recognized.

5.2 Outlook

Thus far only a position estimation was realized. While this can be helpful for the Monte Carlo localization with a laser scanner, it was shown that there are still a lot of problematic limitations. A sensor measurement model for the Wi-Fi receiver was implemented. This could be used directly in amcl. For the Monte Carlo localization it is no problem to incorporate different sensors. The weights can simply be computed by computing the product of all sensor measurement models. But once again the biggest obstacle here is the time requirement for both the Wi-Fi scan and the computation time of the Wi-Fi measurement model. As for the Wi-Fi measurement model this problem could be eased by the fact that amcl uses a lot less particles when it converged to a position. Another trick that could be used is to omit some access points per iteration. But even then the time a Wi-Fi scan takes would pose a problem. Also the question if incorporating the Wi-Fi data into the localization would lead to a better localization still stands. With the measurement model used here for the Wi-Fi receiver it is not possible to determine the direction of the robot, but only its position. The localization with a laser scanner is already reliable once the true pose was found, so the contribution of Wi-Fi data to this problem remains questionable.

Similar concerns stand for a localization realized purely with the Wi-Fi measurement model, without a laser scanner. Once again the time a Wi-Fi scan takes and the computation time of the Wi-Fi sensor model are too high.

As for the use of the Wi-Fi position estimation for the global localization, we still see possibilities for optimizations. Right now particles are spread over the entire map. So with a bigger map one either heightens the particle count, or one accepts the worse position estimation. But with the recorded Wi-Fi signals beforehand and the current recorded Wi-Fi signals one can approximate the area, where these recordings take place. So by looking at the MAC-addresses observed and then looking at the area where these particular MAC-addresses were observed one can at least do an approximation of the area where the robot is and then only spread the particles there. Of course this only works if one assumes that the training data covers the area where the robot is positioned, but in most cases that should be given.

Bibliography

[Biswas and Veloso, 2010] Biswas, J. and Veloso, M. M. (2010). Wifi localization and navigation for autonomous indoor mobile robots.

[Blum and Riedmiller, 2013] Blum, M. and Riedmiller, M. A. (2013). Optimization of gaussian process hyperparameters using rprop. In *ESANN*.

[Dellaert et al., 1999] Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999). Monte carlo localization for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA99)*.

[Duvallet and Tews, 2008] 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.

[Ferris et al., 2006] Ferris, B., Haehnel, D., and Fox, D. (2006). Gaussian processes for signal strength-based location estimation. In *In proc. of robotics science and systems*. Citeseer.

[Ito et al., 2014] 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 et al., 2007] 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*.

[Pinheiro et al., 2015] 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.

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

[Serrano et al., 2012] Serrano, O., Rodero Merino, L., Matellán Olivera, V., Cañas, J. M., et al. (2012). Robot localization using wifi signal without intensity map.

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

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. Oktober 2016

_____

Benjamin Scholz