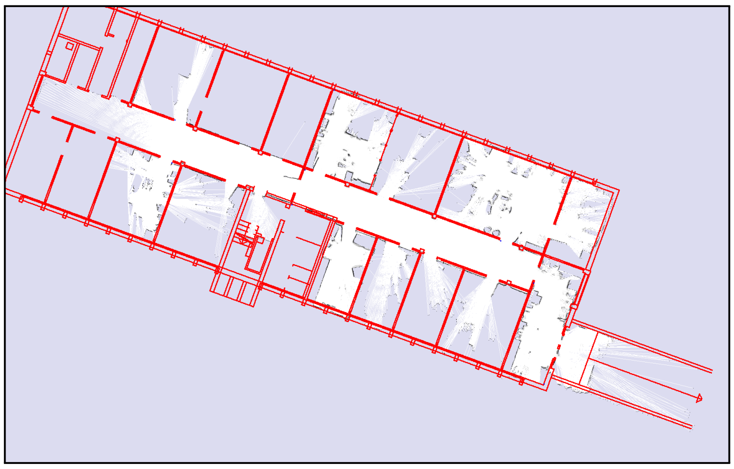
University Freiburg
Technical Faculty
Department of Computer Science
Autonomous Intelligent Systems



Matching Occupancy Gridmaps and CAD-Floorplans for Mobile Robot Navigation

Master Thesis

Submitted by:	Andreas Kuhner
Student Number:	2761155
First Examiner:	Prof. Dr. Wolfram Burgard
Second Examiner:	Dr. Gian Diego Tipaldi
Supervisor:	Christoph Sprunk
Date:	23. January, 2015

Declaration

I hereby declare, that I am the sole author and composer of my thesis

**Matching Occupancy Gridmaps and
CAD-Floorplans for Mobile Robot Navigation**

and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.

I hereby also declare, that my thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Andreas Kuhner

Freiburg im Breisgau, January 23, 2015

Zusammenfassung

Sensorkarten sind für den Laien schlecht zu lesen. CAD-Gebäudepläne hingegen beinhalten viele zusätzliche Details und verbessern die Fähigkeit eines Nutzers einen Roboter zu instruieren und zu überwachen. Diese Arbeit dreht sich um eine Abbildung von einem Gebäudeplan auf die Sensorkarte. Wir haben mehrere Ansätze entwickelt. Der Erste berechnet eine globale Transformation, während der Zweite mehrere Transformationen zu einer Einzigen kombiniert. Zwei weitere Ansätze basieren auf Gauß'schen Prozessen. Beitrag dieser Arbeit sind mehrere Verfahren um die Sensorkarte und den Gebäudeplan aufeinander abzubilden und somit dem Nutzer die Möglichkeit bietet einen intuitiveren Gebäudeplan anstatt einer Sensorkarte zu verwenden. Eine Auswertung der präsentierten Methoden auf verschiedenen Datensätzen zeigt vielversprechende Resultate.

Abstract

Gridmaps are hard to read for amateurs. However, floorplans provide a lot of additional information and improve the capability of an operator to instruct and monitor a robot. This thesis is about a mapping approach which transforms a point from the floorplan to a point on the gridmap. The first approach computes a global transformation, while the second one combines multiple transformations into a single one. The other two approaches base on a Gaussian process. The contributions of this work are multiple approaches to match a gridmap with a floorplan which provides the operator an opportunity to use a more intuitive CAD-map. We evaluate the different approaches on multiple data sets and show promising results.

Contents

1 Introduction

Nowadays, robots can localize themselves in their world very robustly. They use laser-range finders, sonars, cameras, and many more to build huge maps. A popular variant is the usage of laser scanners. Robots equipped with such sensors build a gridmap where every cell is black if and only if the robot has seen some obstacle on this particular spot. The robot uses this map to compute its position by matching the current laser scans into it. But for what could we use it?

A common scenario in industrial environments is that we want to control an autonomous transporter with laser scanners. An operator needs to send him to a given position on the map. Unfortunately, the gridmap is difficult to read for a non-expert and lacks a lot of detailed information. Figure 1.1 (a) shows an example of such a gridmap. Parts of the machines and the pillars of the high racks are available. The map misses a complete room. Figure 1.1 (b) illustrates the floorplan of the same warehouse. The operator knows where the brown point is on the gridmap. However, he does not know the exact position of the blue and red point. If he wants to send a transporter to such a position he has to guess the point on the gridmap. On the other hand, if he wants to guide the transporter to a machine he probably cannot say where machine 1 is on the gridmap. This gridmap shows how the world looks to the robot sensors which is not intuitive for a human. Hence, it is easier for the operator to set the goal points on a floorplan to which he is more familiar and where he can see the whole building structure. However, we have the problem to translate a point on the floorplan to a point

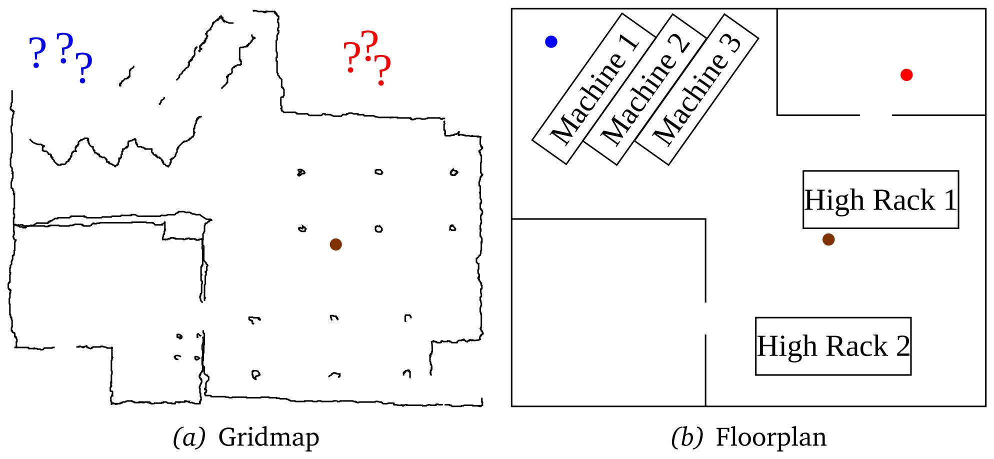
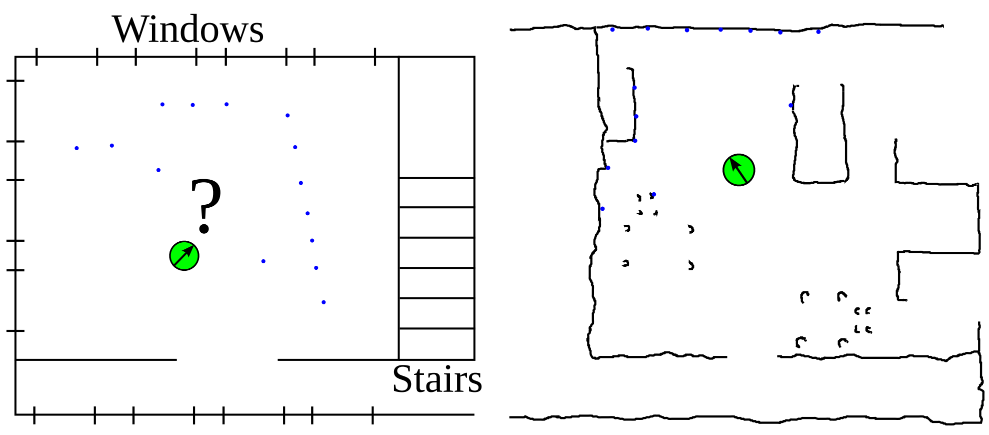


Figure 1.1.: The transporter did not explore the whole area in the warehouse. Hence, the gridmap is not complete. The points represent positions on the map. The brown point on the floorplan has an obvious match on the gridmap. However, we do not know the exact position of the red and blue point.

in the gridmap. Although a human can find itself in the floorplan does not mean that the transporter can localize itself on the floorplan very well. A transporter needs all the present obstacles in a map. Figure 1.2 shows such a scenario. The left room is part of a floorplan while the right side represents a gridmap. The autonomous transporter needs a method to fit its laser scans into the map which is hard if there are no furniture or additional obstacles. The transporter lacks information if it has only a blank room without any details. Hence, we need an approach that is able to find a corresponding point in the gridmap given a point in the floorplan. With such an approach the transporter knows the goal position on the gridmap, can start to plan a path through the environment, and drive to the goal position. On the other hand, the operator can follow the transporter position on the floorplan due to a mapping from the gridmap onto the floorplan.
This thesis presents several methods to map a given point on the floorplan to the corresponding point on the gridmap and vice versa. Therefore, we apply multiple algorithms as computing a rigid transformation or calculating the likeliest point. The main challenges are global and local variations like a



(a) Floorplan *(b)* Gridmap

Figure 1.2.: The green transporter can find its position on the gridmap because a lot of obstacles are drawn into the map. On the other hand, the transporter on the floorplan does not know how to fit his blue laser scan into the room.

bent map or a rotated room. Hence, in Chapter 3 we present all fundamental algorithms, on which we build our matching algorithms in Chapter 4. Section 4.6 is about the automatic generation of reference pairs. Then, we present the basic structure of our approaches in Chapter 5. Finally, in Chapter 6 we evaluate our approaches on different data sets. But first we present related work in the next chapter.

2 Related Work

There are multiple ways to use the matching of two maps. In [15] the authors match different floors to each other to improve the structure of each gridmap and create multi-floor gridmaps. They assume that each floor shares at least some identical structure. To find such areas they apply a particle filter-based localization by using observations and maps from different floors. In contrast to this approach we apply a particle filter to a gridmap and a floorplan to find corresponding points on each map. Another approach [14] solves the multi-floor problem by using a set of 2D maps and aligning them with precise visual odometry. The authors apply state-of-the-art mapping algorithms to compute the gridmap of each floor using a 2D laser scanner. [20] presents an approach which relies on barometric pressure data to separate different floors. The authors generate segments from a data set by looking at the deviation of the barometric pressure and perform SLAM on each of these segments. Afterward, they connect the floors where the robot moved into the next floor. In [17] the authors use publicly available aerial images as prior information to improve the mapping process. They insert correspondences found between the robots sensor data and the aerial images as constraints into a graph-based formulation of the SLAM problem.

Not only is the matching of two maps challenging but also the storage of such maps. In [30] the authors propose a way to store multiple surfaces in a map and how to update them according to matched maps.

Another direction in which matching of maps is useful is for the application of multi-robot mapping. In [13] the authors challenge three problems: In the first one they extend a particle filter to work on multiple robots and deal with the SLAM problem where the initial pose is known. The second challenge is to solve the same problem without knowing the start position of any robot. The last challenge is about the integration of data when two robots meet each other. They use the collected data in inverse order. The final result is a single map. A similar approach is used in [27]. The authors combine a fast maximum likelihood map growing with a Monte Carlo localizer. Hence, they are able to deal with large odometry errors and can create maps with multiple robots. In [16] the authors also have multiple robots and want a single map. Therefore, the key idea of their approach is to localize each robot in the maps of the other robots and verifying matches with a rendezvous strategy. In [3] the authors provide a stochastic approach. Their algorithm transforms and rotates two maps to find the maximum overlap. To identify this position they propose a heuristic which guides the algorithm to the best overlap. A downside of this approach is the reliance that each robot creates a map without making big errors due to bending or skewing. Another approach with a heuristic is introduced in [19]. The authors try to match two maps based on features like doors, corners, T-junctions, and end of corridors. In [28] the authors describe an algorithm which works with landmarks. They align local maps with a tree-based algorithm which searches for similar landmark configurations and combine that with a hill climbing approach that maximizes the overall likelihood in the space of correspondences. A graph based approach is presented in [21]. Multiple robots create a graph where each robot has an own subgraph which is disconnected to all other robots. They use loop closing techniques to detect overlaps between two maps. Hence, they compute a rigid transformation for one map and fit it into the other map.

Another area, which is close to the topic described in the introduction, is the mapping of warehouses. Typically, experts set up markers around the warehouse and create a map from it that contains the exact positions of the marker. Afterward, the robot is able to precisely localize itself in the environment. The downside of this approach is the manual installation of the markers. To overcome this problem [2] presents an approach which creates a gridmap from the warehouse. The authors try to keep the system on

low cost by reducing the installation time and apply a semi-automated exploration and localization approach. Another problem, which they have to tackle, are the large areas of a warehouse. Angular errors are critical in such a scenario. In [25] the authors use a wireless sensor network to keep track of their transport vehicles and in [31] the authors combine a radio-frequency identification to roughly localize a vehicle with a vision based system to perform the fine positioning.
In this thesis we present an approach which overcomes the limitation of an operator to stick to a warehouse gridmap. We show how a point, which we choose on the floorplan, translates to a point on the gridmap. In difference to the related work we do not match sensor maps but CAD-floorplans to sensor maps and vice versa. Hence, we have to deal with different representations of data. But first, we look into the preliminaries in the following Chapter.

3 Preliminaries

In this chapter we discuss the algorithms for the approaches presented in Chapter 4. The first part is about the estimation of rigid transformations which we use to compute global and local transformations. Part two introduces Gaussian processes which we use to calculate the likeliest reference point given our data. The third section is about Delaunay triangulations which we use for the purpose to create a ground truth and the final part is about the calculation of reference points.

3.1. Problem Definition

Before we start with the preliminaries we discuss the problem which we have to solve in Chapter 4. We have two maps M_F and M_G where M_F is a floorplan and M_G a gridmap. F and G represent point sets on these maps where F is the respective set of M_F and G corresponds to M_G with $F, G \in \mathbb{R}^2$. FG is a set of tuples (p, q) with $p \in F$ and $q \in G$. We call this set the set of reference pairs. Thus, each point in F has exactly one point in G. With the set FG we compute a transformation T from M_F to M_G. We use the transformation T to map points from M_F onto M_G which are not in F and vice versa.

3.2. Estimation of Rigid Transformation

We use rigid transformations to apply a translation, rotation, scaling and shear to a point. The function we are looking for is a homogeneous transformation $T\colon F \to G$ with $F, G \subset \mathbb{R}^3$. The definition of F and G is the same as described in the previous section, just with the difference that we compute a homogeneous transformation and extend every point $a = (x, y)^T$ to $a = (x, y, 1)^T$. The basic idea is that we take three random tuples from FG, which we also adapt for the homogeneous case, and calculate the transformation with a Singular Value Decomposition which we explain in the next section. However, we repeat this multiple times using the Random Sample Consensus algorithm 3.2.2 and get an approximated transformation from the set F to the set G. We use this technique in Section 4.2 to compute a transformation to map points from the floorplan onto the gridmap.

3.2.1. Singular Value Decomposition

The Singular Value Decomposition (SVD) [11] decomposes a matrix into several pieces

$$M = U\Sigma V^*, \tag{3.1}$$

where U is an $m \times m$ unitary matrix, Σ is an $m \times n$ diagonal matrix with elements in $\mathbb{R}_+$ and the $n \times n$ unitary matrix V^* denotes the conjugate transpose of the unitary matrix V with $n, m \in \mathbb{N}$. To find these components we have to get the Eigenvalues and Eigenvectors of MM^T and M^TM. The Eigenvectors of M^TM form the columns of V and the Eigenvectors of MM^T make up the columns of U. On the other hand, the singular values of Σ are square roots of Eigenvalues from the matrix MM^T and M^TM, respectively. These values are the diagonal entries of Σ and are typically sorted in descending order. Let

$$W = MM^T \text{ and } Z = M^TM \tag{3.2}$$

be two matrices. We want to compute the Eigenvalues. Thus, we solve the equations:

$$(W - \lambda I)x = 0, \tag{3.3}$$

$$(Z - \mu I)x = 0. \tag{3.4}$$

With the resulting Eigenvectors we get U and V. As mentioned previously, the diagonal elements of Σ are the ordered square roots of the Eigenvalues from W and Z. Therefore we possess the decomposition of M.
We split the computation of the transformation between corresponding points in FG into the following three steps:

1. Calculate the centroids of both point sets

2. Transform both sets to the centroid and compute the rotation R

3. Find the translation t

The first step involves the calculation of the centroids which are typically the average points of the set.

$$c_F = \frac{1}{N}\sum_{i=1}^{N} P_F^i \tag{3.5}$$

$$c_G = \frac{1}{N}\sum_{i=1}^{N} P_G^i, \tag{3.6}$$

where P_F and P_G are the point sets referring to F and G. Now we can compute the rotation with the help of the SVD. First we subtract from every point the centroid such that we only have to deal with the rotation and compute the covariance matrix H like

$$H = \sum_{i=1}^{N} \left(P_F^i - c_F\right)^T \left(P_G^i - c_G\right). \tag{3.7}$$

Splitting this matrix up into its SVD composition yields

$$H = U\Sigma V^*. \tag{3.8}$$

If H is a square matrix than U and V^* are rotation matrices and Σ is a scaling matrix. The composition

$$R = VU^T \tag{3.9}$$

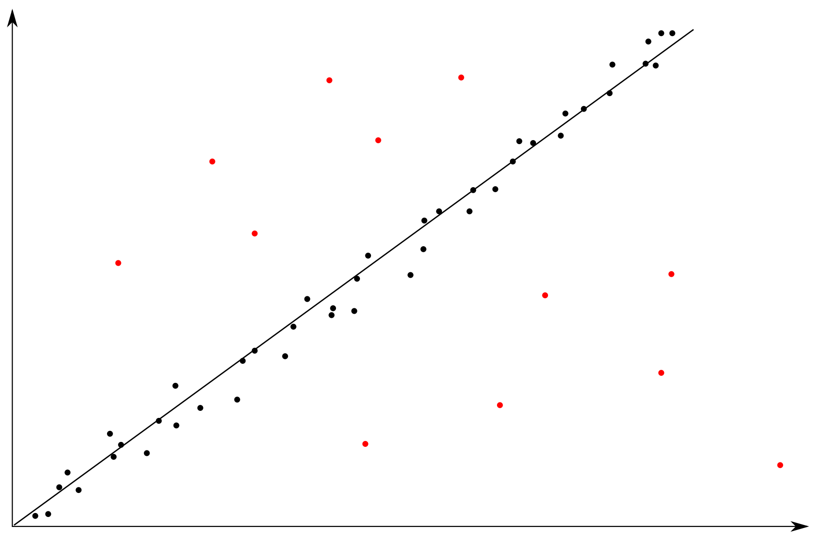


Figure 3.1.: In this example RANSAC takes two points into account and fits a line to it. Here are the red points the outliers which are not part of the biggest consensus set and the line is the RANSAC result.

yields our rotation matrix.
The last step handles the computation of the translation:

$$t = -Rc_F + c_G. \tag{3.10}$$

Finally, we get the resulting transformation:

$$T = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}. \tag{3.11}$$

3.2.2. Random Sample Consensus

Random Sample Consensus, short RANSAC, [8] is a model that describes given data under consideration of outliers. Figure 3.1 shows a scenario where a line should be fit into some 2D data. In our scenario we have two sets of points F and G, as given in Section 3.1, and the set of tuples FG which connects these points. We want the transformation between these sets of points. One iteration of the RANSAC method can be split up into two parts. In the first one we chose randomly three tuples from the set FG and calculate the transformation T from the previous Section 3.2.1:

$$Tx = Rx + t. \tag{3.12}$$

In the second part we compute the error for each point:

$$e_i = \|P_G^i - T \cdot P_F^i - t\|^2. \tag{3.13}$$

If the model estimates the transformed point well, it is added to the consensus set. Therefore, we calculate the error for each point and after this process we compare the set size against the so far biggest consensus set. If the set size is bigger we have a new consensus set.
After a given number of iterations or after the error falls below a threshold we compute the transformation using all point correspondences of the best consensus set one more time and get our final transformation.

3.3. Gaussian Processes

The Gaussian process [22] is another way to create a transformation between two sets of points. In principle it is a generalized Gaussian distribution with infinitely many random variables. The feature of this approach is its focus on basic stochastic processes. Therefore, it can also predict the variance of each predicted point. In contrast to other approaches, like linear regression, quadratic error minimization, or higher dimensional approaches, Gaussian processes do not relate to any specific model. Moreover, they can represent the underlying function of a data set obliquely but still exact. The basic idea is that the approach works on the data set rather than on a generated, approximated function. Thus, on the one hand, the Gaussian processes depend less on parameters. On the other hand, the processes cannot disclaim the parameters completely. We still have to make certain assumptions to get a reasonable result. However, if we compare it to the previous section we do not compute a transformation but a function which depends on local points. We apply this technique in Section 4.4 to the matching problem of two maps. Let (Y_i) be a multidimensional normal distribution with index set I. The process is a Gaussian process if and only if each finite set of indices in I $(Y_{i_1}, ..., Y_{i_n})$ is a multivariate Gaussian random variable. This random variable can be seen as n observations from a data set. We assume that the mean of all Gaussians is zero. Thus, all observations are related to each other over the covariance function $k(x_1, x_2)$. A common choice is the squared exponential:

$$k(x_1, x_2) = \sigma_f^2 \exp\left(\frac{-(x_1 - x_2)^2}{2l^2}\right), \tag{3.14}$$

where σ_f^2 is the covariance. If our data set is really noisy σ_f^2 should be high. For $x_1 \approx x_2$ our covariance function k approaches the maximum. Therefore, $f(x_1)$ correlates to $f(x_2)$ nearly perfect. On the other hand, if x_1 and x_2 are very distant from each other the resulting covariance $k(x_1, x_2)$ is close to zero. This are the preferred properties because for a smooth function close points should be highly correlated. To summarize this: Points in close proximity of an interpolated point p should have higher impact on p than points which are far away. The impact of this effect is parameterized by the length l.
Another problem is noisy observation data. Typically, we assume a Gaussian noise model:

$$y = f(x) + \mathcal{N}(0, \sigma_n^2), \tag{3.15}$$

with σ_n^2 as the variance of the noise. It is possible to fold this noise into the covariance function k:

$$k(x_1, x_2) = \sigma_f^2 \exp\left(\frac{-(x_1 - x_2)^2}{2l^2}\right) + \sigma_n^2 \delta(x_1, x_2), \tag{3.16}$$

where $\delta(x_1, x_2)$ is the Kronecker delta function. Finally, we do not want to calculate the function f but a predicted value y_* for input x. Therefore, we compute the covariance matrix:

$$K = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \dots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \dots & k(x_n, x_n) \end{pmatrix}. \tag{3.17}$$

We also need two other components for the calculation of y_*:

$$K_* = \begin{pmatrix} k(x_*, x_1) & k(x_*, x_2) & \dots & k(x_*, x_n) \end{pmatrix} \tag{3.18}$$

and

$$K_{**} = (k(x_*, x_*)), \tag{3.19}$$

where x_* is the value for which we search a respective y_*. The key assumption with Gaussian processes is that our data can be presented as a multivariate Gaussian distribution. Thus, we have:

$$\begin{pmatrix} y \\ y_* \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} K & K_*^T \\ K_* & K_{**} \end{pmatrix}\right). \tag{3.20}$$

Hence, we want a y_* which maximizes the probability $p(y_*|y)$. This follows a Gaussian distribution:

$$y_*|y \sim \mathcal{N}(K_* K^{-1} y, K_{**} - K_* K^{-1} K_*^T). \tag{3.21}$$

Therefore, the best guess is the mean of this distribution:

$$\overline{y}_* = K_* K^{-1} y \tag{3.22}$$

and its respective variance

$$\sigma(\overline{y}_*)^2 = K_{**} - K_* K^{-1} K_*^T. \tag{3.23}$$

What we still do not know are the hyperparameters σ_f and l. We either could estimate them based on knowledge over the data set or try to optimize them as described in the next section.

3.3.1. Resilient Propagation

σ_f and l are two important parameters. Thus, we have to learn them and fit them to the data. Therefore, we use resilient propagation, short RPROP, [23]. It is a gradient descent algorithm which softens the effects of pure gradient decent approaches which sometimes miss the minimum. These algorithms rely on the magnitude of the derivative. Therefore, they make huge steps on steep parts of the function and little steps on plateaus. To overcome these effects RPROP adapts its weight update to the local behavior of the error function by looking at sign changes. A detailed presentation of the algorithm can be found in [24].
The basic idea of a backpropagation learning algorithm is to calculate the influence of each weight with respect to an error function. RPROP minimizes the error function by performing a gradient decent step:

$$w_i(t+1) = w_i(t) - \epsilon \frac{\delta E}{\delta w_i}(t), \tag{3.24}$$

where w_i is a parameter like σ_f or l. The learning rate ϵ is a critical part of the performance to reach convergence. If the parameter is set too high the algorithm could end up in an oscillation. On the other hand, a small ϵ could lead to a very slow convergence rate. To adapt this parameter there are two major categories of strategies: Global strategies and local strategies. In the global case the algorithm depends on all parameters, whereas local strategies focus on parameter specific information like the partial derivative. The main problem in both categories are unforeseeable fluctuations in the derivative. Thus, a well-trained learning rate can still be disturbed by the derivative and lead to unwanted behavior. To deal with this issue RPROP changes the size of the parameter update directly without considering the magnitude of the derivative. To achieve this RPROP has for each weight its own update value and each of it evolves over the course of the algorithm independently but dependent on the partial derivative of the error function. Thus, RPROP uses this rule for the updates:

$$\Delta_i(t) = \begin{cases} \eta^+ \Delta_i(t-1) & \text{,if } \frac{\delta E}{\delta w_i}(t-1) \cdot \frac{\delta E}{\delta w_i}(t) > 0 \\ \eta^- \Delta_i(t-1) & \text{,if } \frac{\delta E}{\delta w_i}(t-1) \cdot \frac{\delta E}{\delta w_i}(t) < 0 \\ \Delta_i(t-1) & \text{else} \end{cases}, \tag{3.25}$$

with $0 < \eta^- < 1 < \eta^+$. If the sign of the derivative gets changed then the update was too big and the algorithm missed the local minimum. Thus, the update value gets decreased by η^-. If the sign did not change the update value can be increased by η^+ to speed up the convergence rate. The parameter update itself follows a simple rule: With a positive derivative the weight is decreased by its update value. On the other hand, if RPROP has a negative derivative, it adds the update value to the parameter:

$$\Delta w_i(t) = \begin{cases} -\Delta_i(t) & \text{,if } \frac{\delta E}{\delta w_i}(t) > 0 \\ +\Delta_i(t) & \text{,if } \frac{\delta E}{\delta w_i}(t) < 0 \\ 0 & \text{else} \end{cases} \tag{3.26}$$

and

$$w_i(t+1) = w_i(t) + \Delta w_i(t). \tag{3.27}$$

However, if the sign of the derivative is changed, over the course of this update step, the result is getting worse. Hence, the parameter update was too big. Thus, RPROP reverts the previous weight update:

$$w_i(t) = -w_i(t-1) \quad \text{,if } \frac{\delta E}{\delta w_i}(t-1) \cdot \frac{\delta E}{\delta w_i}(t) < 0. \tag{3.28}$$

Here, the problem is that RPROP punishes the update value twice because the derivative is supposed to change its sign once again in the upcoming step. To solve this RPROP sets $\frac{\delta E}{\delta w_i}(t-1) = 0$ in the adaption rule of Equation 3.25.

3.3.2. Applying Resilient Propagation to a Gaussian Process

In the previous section we discussed how we can learn the hyperparameters of a Gaussian process. To apply RPROP to Gaussian processes we need an error function as well as an initialization. Finding an initialization is problem dependent. As in Section 3.2.2 we can use the error function

$$e_i = \|y_i - \overline{y}_i^*\|^2, \tag{3.29}$$

where G and F are the point sets from Section 3.1 and $x_i \in G$ is the corresponding point to $y_i \in F$. $\overline{y}_i^*$ is based on the predicted point x_i which we get from Equation 3.21. Along this error function we can deploy RPROP to optimize σ_f as well as l.

3.4. Delaunay Triangulation

In the previous sections we explained several algorithms which form the base for describing our approaches in Chapter 4. To compare these algorithms we need a tool to create a ground truth for which we use a triangulation.
A triangulation on a graph is called Delaunay if and only if in the circumcircle of each triangle is no other point. This triangulation [4] is the dual graph of a Voronoi diagram. Hence, if we take the middle point of the circumcircle of each triangle in the Delaunay graph and connect them, we get a Voronoi graph.
To compute a Delaunay graph we apply an incremental approach. As proposed in [1] the key idea is to start with a single triangle and incrementally add points to the graph. Afterward, we connect the point to other ones in such a way that the Delaunay condition is still meet. Figure 3.2 shows one step of this incremental approach. We apply the Delaunay triangulation to a set of corresponding points to compute a ground truth which we later use in Section 4.5 to compare it with our other algorithms.
Let p be a new point. If p is inside of the circumcircle the triangle fulfills the Delaunay property no longer. Therefore, we apply the algorithm proposed

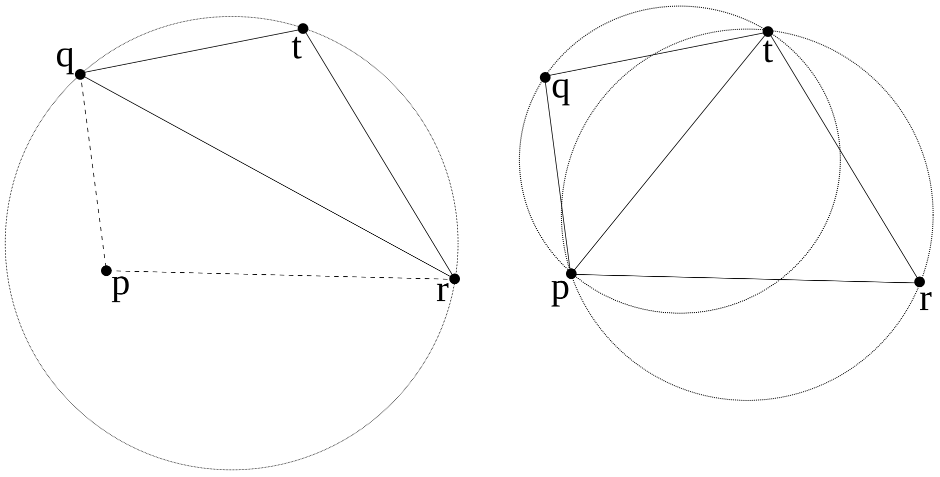


Figure 3.2.: p violates the Delaunay property of the triangle $\overline{qrt}$ and lies inside of its circumcircle. Flipping the line $\overline{qr}$ with $\overline{pt}$ results in two triangles with a valid Delaunay property.

by [18] which solves the issue by flipping edges. If the point p lies within the circumcircle of the triangle $\overline{qrt}$ the edge $\overline{qr}$ is flipped with the line $\overline{pt}$. Afterward, we have two new triangles which both fulfill the Delaunay property. This procedure is repeated as long as p invalidates any edges and stops if no other violation is present. Hence, we just insert points and make edge flips until the full graph fulfills the Delaunay property.

3.5. Monte Carlo Localization

The Monte Carlo Localization [9] is a variant of the Markov localization and solves the hijacked robot problem [6]. The key idea of Markov localization approaches is to represent the robot's belief with a probability distribution over possible states. It transitions from one state to the next by applying Bayes rule from Section 3.5.1 whenever the robot moves or observes its environment. In contrast to the early beginning of robot position tracking with Kalman filters [10] which only can represent one state at once, Markov localization is able to deal with multiple states simultaneously and, therefore, can tackle the robot localization problem. Unfortunately, Markov localization ap-

proaches only approximate the belief and the quality depends on the sample size. Hence, if we want a more accurate model than we have to invest more computation power and memory to the problem. The Monte Carlo localization is part of the family of Monte Carlo methods [12]. It does not represent a robot's belief with a continuous function but with samples. Therefore, in each update step we apply an importance re-sampling [26] to estimate the new distribution. We use this localization and tracking technique to calculate in Section 4.6 two sets of points and the correspondences between them.
In the next section we discuss the fundamental Bayes filter.

3.5.1. Bayes Filter

A Bayesian filter is a general probabilistic approach to approximate a probability distribution. Given a belief Bel(x_t) of the robot with state x_t at time t:

$$\text{Bel}(x_t) = P(x_y|u_1, z_1, \cdots, u_t, z_t), \tag{3.30}$$

with u_i being the controls of the robot and z_i the observations. To break this term down we have to look at several rules.
The first one is the Bayes rule. Let $P(x|y)$ the conditional probability of x given y. It holds that

$$P(x|y) = \frac{P(x,y)}{P(y)}, \tag{3.31}$$

where $P(x, y)$ is the joint probability of x and y. If they are independent of each other

$$P(x, y) = P(x)P(y) \tag{3.32}$$

and

$$P(x|y) = P(x). \tag{3.33}$$

The second part is about the Markov assumptions. A model has the Markov property if and only if the next state only depends on the current state and not on the sequence which led to it. Figure 3.3 shows the dependency graph on which the Bayes filter operates. In the middle section we see the robot states x_i. These states depend on the previous state x_{i-1} as well as on the

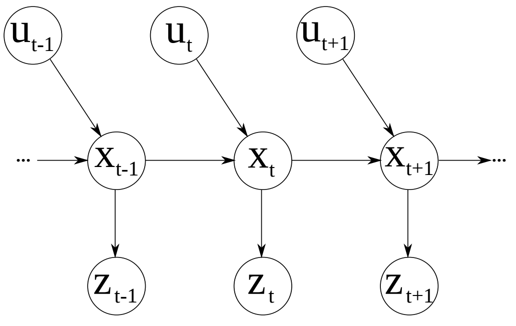


Figure 3.3.: A dependency graph: x_i is the robot state which depends on the previous state and the control command u_i. The observation z_i depends only on the state x_i.

control u_i. The observation z_i depends only on the robot state x_i. The first underlying assumption, which leads to the Markov assumption, is a static world. Therefore, the world does not change over time and the observations of the robot remain unchanged as long as it is located at a point p and oriented in direction α.
The next assumption is independent noise. Hence, it does not evolve from any robot state, control command or observation. We can sample it from some distribution, i.e., a Gaussian normal distribution.
The third and last assumption is a perfect model. Therefore, the robot executes every control command without errors and retrieves a perfect observation of the environment. These three assumptions form the Markov assumption and lead to the following equations:

$$P(z_t|x_{0:t}, z_{1:t-1}, u_{1:t}) = P(z_t|x_t) \tag{3.34}$$

$$P(x_t|x_{1:t-1}, z_{1:t-1}, u_{1:t}) = P(x_t|x_{t-1}, u_t) \tag{3.35}$$

The first equation tells us that the probability $P(z_t|x_{0:t}, z_{1:t-1}, u_{1:t})$ of the observation only depends on the current state. The second one is about the current robot state. The probability $P(x_t|x_{1:t-1}, z_{1:t-1}, u_{1:t})$ only depends on the previous state and the current control command. All states of the robot are independent from the observations and all control commands up to time $t-1$ are incorporated into x_{t-1}.

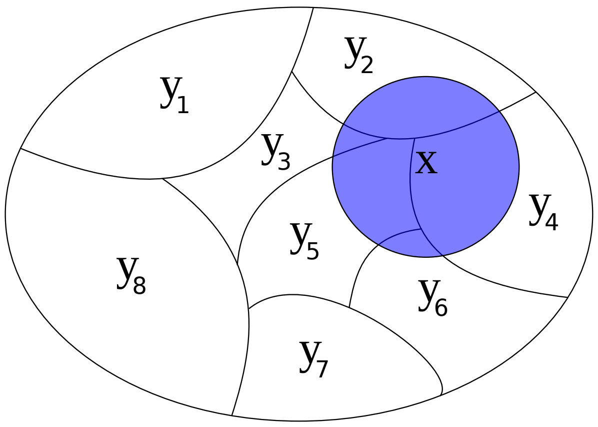


Figure 3.4.: The space is subdivided into disjunctive pieces $y_1, \cdots, y_7$. We can compute $P(x)$ by summing up all conditional probabilities $P(x|y)$ multiplied with $P(y)$. Hence, only such conditional probabilities have an influence where x and y_i intersect. All other conditional probabilities are zero.

To calculate the Bel(x_t) of a robot we need the law of total probability. It states that

$$P(x) = \sum_y P(x|y)P(y). \tag{3.36}$$

The basic idea is that we can compute $P(x)$ if we know all conditional probabilities $P(x|y)$ and the probabilities $P(y)$. Figure 3.4 shows an example: The space is divided into several parts $y_1, \cdots, y_7$ and only such conditional probabilities are not zero where x and y_i intersect. Let us break down the term

$$\text{Bel}(x_t) = P(x_y|u_1, z_1, \cdots, u_t, z_t).$$

We want to form it into a recursive version where we know all parts. Hence, we apply the Bayes rule in the first step:

$$P(x_y|u_1, z_1, \cdots, u_t, z_t) = \eta P(z_t|x_t, u_1, z_1, \cdots, u_t)P(x_t|u_1, z_1, \cdots, u_t).$$

η is the normalization term. In the next step we use the Markov assumption from Equation 3.34 to reduce the second part:

$$\eta P(z_t|x_t, u_1, z_1, \cdots, u_t) = \eta P(z_t|x_t).$$

The third part can be computed by using the law of total probability from Equation 3.36:

$$P(x_t|u_1, z_1, \cdots, u_t) = \\ \int P(x_t|u_1, z_1, \cdots, u_t, x_{t-1}) P(x_{t-1}|u_1, z_1, \cdots, u_t) \, \mathbf{d}x_{t-1}.$$

By applying the Markov assumption twice we can shrink the integral term:

$$\int P(x_t|u_1, z_1, \cdots, u_t, x_{t-1}) P(x_{t-1}|u_1, z_1, \cdots, u_t) \, \mathbf{d}x_{t-1} = \\ \int P(x_t|u_t, x_{t-1}) P(x_{t-1}|u_1, z_1, \cdots, u_t) \, \mathbf{d}x_{t-1} = \\ \int P(x_t|u_t, x_{t-1}) P(x_{t-1}|u_1, z_1, \cdots, u_{t-1}, z_{t-1}) \, \mathbf{d}x_{t-1}.$$

The last term corresponds to Bel$(x_t - 1)$. Thus, we can write the probability in a recursive way:

$$\eta P(z_t|x_t) \int P(x_t|u_t, x_{t-1}) P(x_{t-1}|u_1, z_1, \cdots, u_{t-1}, z_{t-1}) \, \mathbf{d}x_{t-1} = \\ \eta P(z_t|x_t) \int P(x_t|u_t, x_{t-1}) \text{Bel}(x_{t-1}) \, \mathbf{d}x_{t-1}.$$

This leads to the Bayes filter equation:

$$\text{Bel}(x_t) = \eta P(z_t|x_t) \int P(x_t|u_t, x_{t-1}) \text{Bel}(x_{t-1}) \, \mathbf{d}x_{t-1}. \quad (3.37)$$

We can think of this formula as a combination of two parts: In $P(x_t|u_t, x_{t-1})$ we compute the motion model. Accordingly, we want the probability of x_t given that the robot was at position x_{t-1} in the previous time step and got the control command u_t. The other part is the observation model. The term

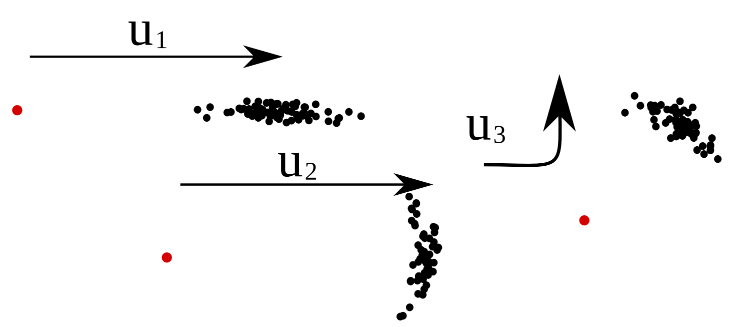


Figure 3.5.: The first motion command u_1 is moving on a straight line with a robot which introduces a lot of translational but less rotational error. Here, the red dot is the start point of the robot and the black dots correspond to the resulting shape according to the error distribution. Another robot with less translational but much rotational error is performing the motion command u_2. The consequence is a banana shape. A third robot gets the command u_3. It introduces equally errors in rotation and translation.

$P(z_t|x_t)$ implies that we have an observation z_t given the robot is on position x_t. Many approaches like the Kalman filter [10] and the Particle filter [5] base on this formula and implement the motion and observation model in different ways. In the Section 3.5.2 and 3.5.3 we discuss how the Monte Carlo localization is dealing with these two models.

3.5.2. Motion Model

Robot motion is typically not perfect. Every move initiates an error. Therefore, short motions are not problematic because the error is not relevant but for a whole trajectory this error accumulates and can lead to a completely different position in comparison to the commands that we sent. Figure 3.5 represents three different robots with motion commands. The first robot has much translational but less rotational error. Therefore, the motion command u_1 creates a set of points which is stretched according to the movement direction. The second movement command u_2 is executed by another robot with high rotational but small translational error. The resulting shape looks like a banana. The third robot with motion command u_3 has an equally sized

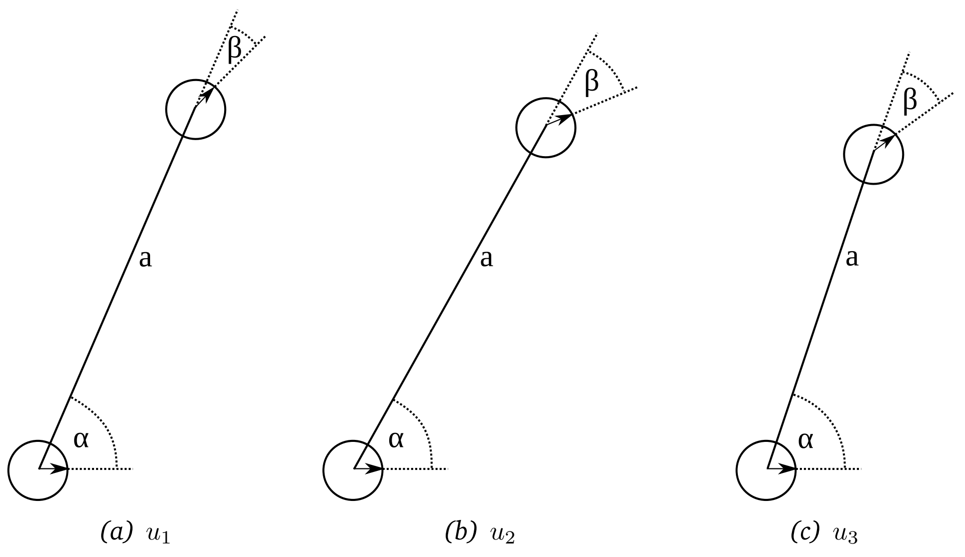


Figure 3.6.: u_1 represents the command sent to the robot. u_2 and u_3 are two ways a robot could execute this command. In u_2 the robot did not turn enough into the α direction but too much into the β one. The translational part is perfect. On the other hand, in u_3 the execution of the rotation in α and β shows an error. Also, the robot did not drive the whole path a. Hence, all three trials end up in a different configuration.

error in translation and rotation. Therefore, the shape is a banana which is stretched in the direction of movement.
The key idea is that we take the current robot position and sample the next position from some distribution. Here, we assume that the robot moves with an odometry model. Hence, a motion command is split into three parts: The first one is a rotation into the direction of the movement, the second one a translation, and the third one the rotation to the final orientation. In every step we presume that the robot is making some error. Therefore, we add noise to all rotations and translations. We sample this noise from a Gaussian distribution centered on zero:

$$\epsilon_{\sigma^2} = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{x^2}{\sigma^2}\right), \tag{3.38}$$

where σ^2 is the variance. In Figure 3.6 we see three examples. Here, u_1 is the motion command. Both u_2 and u_3 show examples how the goal position differs if the robot is making rotation and translation errors. To summarize the odometry model we compute the new position in this way:

$$\theta_{cur} = \theta_{old} - \alpha + \xi \tag{3.39}$$

$$x = \cos(\theta_{cur})t + \zeta \tag{3.40}$$

$$y = \sin(\theta_{cur})t + \zeta \tag{3.41}$$

$$\theta = \theta_{cur} - \beta + \xi, \tag{3.42}$$

where x, y is the robot position, α is the first rotation to the movement direction, β the second rotation to the final orientation, t the distance of the translation, ξ is the rotational noise, and ζ the translational noise.
In the next section we discuss how we can calculate the probability that a robot sees some observation z.

3.5.3. Observation Model

The following section is about the probability model of the sensor function of a laser range-finder. In contrast to other proximity sensors like sonar sensors, laser range-finder are more exact. Hence, we do not have to deal with big measurement errors. However, we still suffer from various problems. Crosstalk is one of it. A scanner could read a neighboring laser beam. Another problem are reflective objects. If they reflect one of the beams in another direction the resulting transition time is too high and the scanner measures a too long distance. Two more problems are random measurements and max range readings. The first can occur if the scanner senses an erroneous distance. The second one happens to all beams which either got lost or are outside of the laser's range. To deal with these problems [29] provides a sensor function consisting of several parts. The first part of this function models all beams which hit the expected obstacle or are at least close to it. It is a normal distribution centered on z_{exp}:

$$P_{hit}(z|x,m) = \mu \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(z - z_{exp})^2}{\sigma^2}\right), \tag{3.43}$$

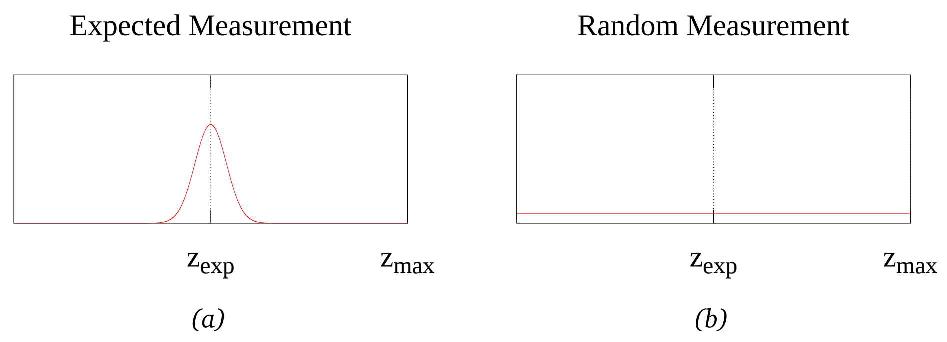


Figure 3.7.: The left graph shows a normal distribution centered on z_{exp}. It deals with all beams which hit an expected obstacle. The right function models random objects.

where μ is a normalization constant, σ^2 the variance, z the measured distance to the obstacle and z_{exp} the expected distance. x is the robot position and m is the map. The variance is a factor to model the uncertainty of the device. However, we use a laser scanner which is precise. Therefore, we can set the variance to a small value and the resulting function gets more peaked. In Figure 3.7 (a) we illustrate the normal distribution which represents the first part of the function.
The next part is a constant function:

$$P_{rand}(z|x,m) = \mu \frac{1}{z_{max}}, \tag{3.44}$$

with z_{max} as the max range. In Figure 3.7 (b) is an example. This function covers all laser scan readings which we do not model and are random. It also prevents the robot to get lost if it is delocalized. With this positive, constant function and the iterative process in Equation 3.37 we still have a probability unequal to zero on all possible positions. Therefore, the robot can localize itself again. However, Monte Carlo localization is working on samples. Hence, all particles in regions with a low probability are likely to die out.

Part three of the sensor function is about max range readings. We see the

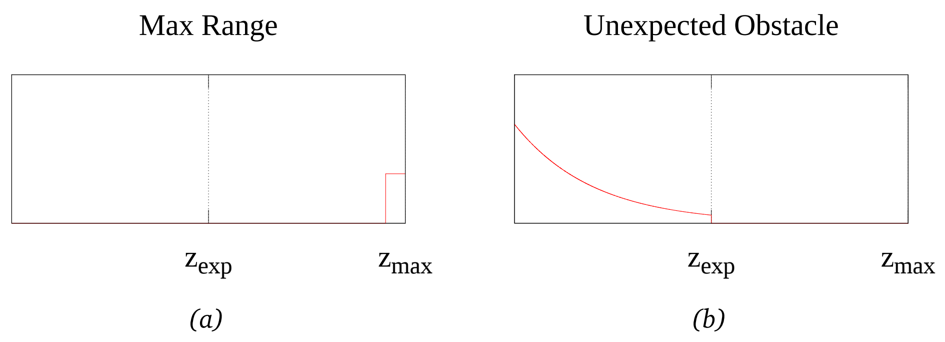


Figure 3.8.: The first graph is about max range readings. We map all measurements which are longer than z_{max} to this value. The other graph shows the function which deals with dynamical objects in front of the robot.

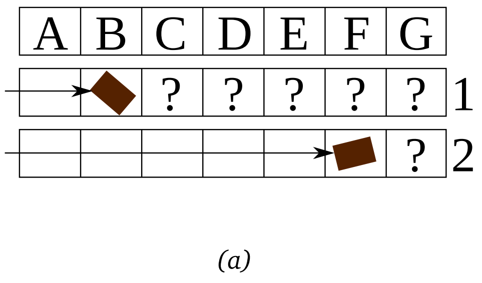
function in Figure 3.8 (a). The formula is:

$$P_{max}(z|x,m) = \begin{cases} \mu\frac{1}{c} & \text{,if } |z - z_{max}| < 0.1 \\ 0 & \text{else} \end{cases} . \tag{3.45}$$

Here, c is a constant value. This max range reading function models all beams which are further away than z_{max}. They either got lost or hit an obstacle which was too far away. The last piece is an adaption to dynamic objects. They are not part of the static world and they do not match to anything on the map. Therefore, we see in Figure 3.8 (b) the function which we use to deal with these dynamic objects. It is given by:

$$P_{unexp}(z|x,m) = \begin{cases} \mu\lambda\exp\left(-\lambda z\right) & \text{,if } z < z_{exp} \\ 0 & \text{else} \end{cases} , \tag{3.46}$$

where λ is a scaling factor. The higher it is the more likely are objects close to the robot. On the other hand, it is less likely when the beam gets closer to the expected distance. The basic idea about this function is the fact that objects which are close to the robot are more likely than objects far away. Figure 3.9 shows two examples. Let the robot be on the left side and A to G



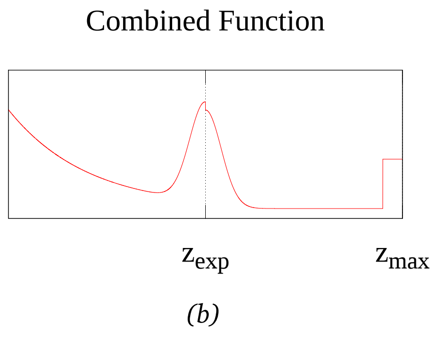


Figure 3.9.: The picture (a) is about the function which deals with dynamic objects. Objects which are close to the robot are likelier than far away objects. The Graph (b) shows the combined function.

the cell names. In row number one we have an obstacle in cell B. Therefore, the robot cannot see anything behind it. What we do is to count the number of possibilities which have this configuration. If the wall is behind cell G we have 2^5 different configurations where $?$ is either blocked or free. In contrast to this is in row two the obstacle in cell F. Hence, we have 2^1 different possibilities. To summarize this small example: In the first case there are more possible configurations compared to the second case. Thus, the first case is more likely.
All parts together form the final formula for the sensor model:

$$P(z|x,m) = \alpha P_{hit}(z|x,m) + \beta P_{unexp}(z|x,m) + \gamma P_{rand}(z|x,m) + \delta P_{max}(z|x,m), \tag{3.47}$$

where α, β, γ and δ are coefficients to regulate the weight of each part. In most cases we neglect the map and only write x to describe both the robot state and the map.
The last section is about the combination of motion model and observation model.

3.5.4. The Structure of Monte Carlo Localization

We discussed in the previous sections how the Monte Carlo localizations observation and motion model looks like. In this section we look at the update

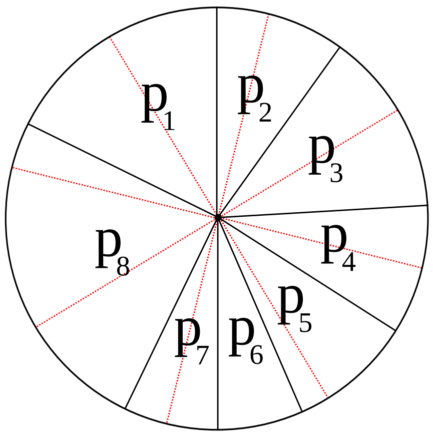


Figure 3.10.: The weight of each particle p_i is the size of its slice. The red lines symbolize which particles we choose. Hence, we take particle p_8 two times and particle p_6 not at all. All other particles are used only once in the final particle set.

rule. The Monte Carlo localization is a particle filter. Therefore, it represents the underlying probability distribution with particles. Each of these particles is a sample drawn from some distribution. After the initialization phase, where we uniformly distribute particles over a map, we move each particle by a motion and, afterward, draw new samples according to the motion and the observation model. Let $p_i \in P$ be a particle in the particle set P. In each update step we perform Equation 3.39 on each particle p_i where p_i represents the robot state. Afterward, we compute the weight of each particle with Equation 3.47. In the case of the Monte Carlo localization x is the particle p_i. We use the weights of each particle to apply an importance resampling. Figure 3.10 shows the importance sampling. The chosen method is stochastic universal sampling. Hence, we have a roulette wheel and each particle has a slice on it proportional to its weight. If we want to sample n particles from this wheel, we draw a random number r_1 between zero and $\frac{1}{n}$. r_1 is the starting point. Afterward, we add $\frac{1}{n}$ to r_1 and we get r_2. We repeat this step n times. The points r_1 to r_n are the positions where we look for particles on the wheel. If r_i is within the bounds of the area of p_j we add the particle p_j to the new particle set P and continue once more with the motion update.

4 Matching of Floorplans and Gridmaps

The problem we face is how to calculate or learn a function which projects a point p from one map to the other one. In our scenario we have a floorplan and a gridmap where we map a point p on the floorplan to a point q on the gridmap. We also show in the experiments how well this projection performs in the other direction where we map a point from the gridmap onto the floorplan. In a perfect world the transformation would be a combination of translation, rotation and scaling. Unfortunately, gridmaps are not exact. Figure 4.1 depicts such a scenario. The left side shows a well aligned floorplan while the other side shows a gridmap. They are not smooth or have a perfect alignment. Also some parts could be stretched while other parts have the correct scale. Therefore, a single homogeneous transformation is an approximation and we are not able to use a single transformation for the whole map. It is even worse: At some points the gridmap could be folded into itself which makes it hard to express this area with just one transformation.
In this chapter we propose several approaches to deal with that problem and to calculate a mapping from one map to the other.

4.1. Linear Combination of Reference Pairs

The first technique we propose is a linear combination of one support vector and two direction vectors. Let F be a set of points which are located on the floorplan and a set of points G which are part of the gridmap. For each point $p \in F$ we have a corresponding point $q \in G$ which is defined in the set FG

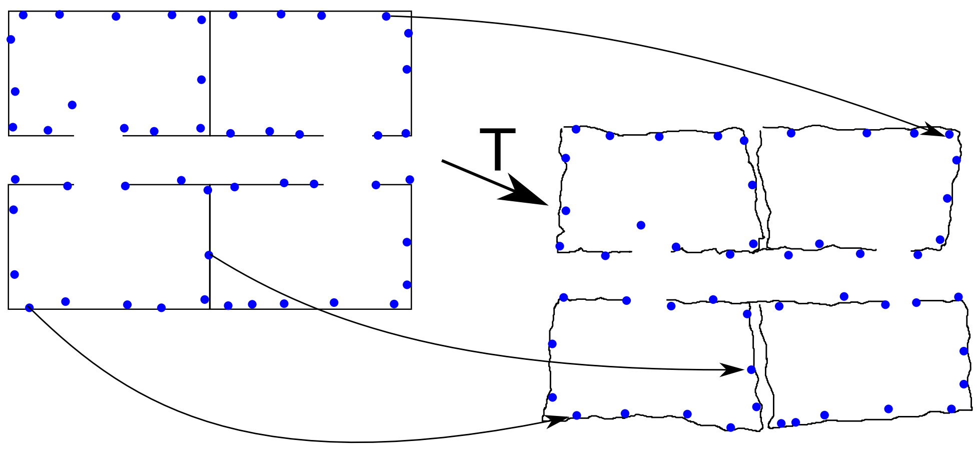


Figure 4.1.: On the left side is the floorplan with blue points which all have a reference point on the right gridmap. These corresponding pairs of points are used to calculate a transformation T.

from Section 3.1. For a point r, which lies on the floorplan, we search for the nearest neighbors in our set F. Let them be r_1, r_2 and r_3. The metric we use is the Euclidean norm:

$$\|a - b\| = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}. \tag{4.1}$$

We express r as a linear combination of the points r_1, r_2, and r_3:

$$r = r_1 + l r_{12} + k r_{13}, \tag{4.2}$$

where we start with the normalized difference vector between r_1 to r_2 and r_1 to r_3:

$$r_{12} = \frac{r_2 - r_1}{\|r_2 - r_1\|}, \tag{4.3}$$

$$r_{13} = \frac{r_3 - r_1}{\|r_3 - r_1\|}. \tag{4.4}$$

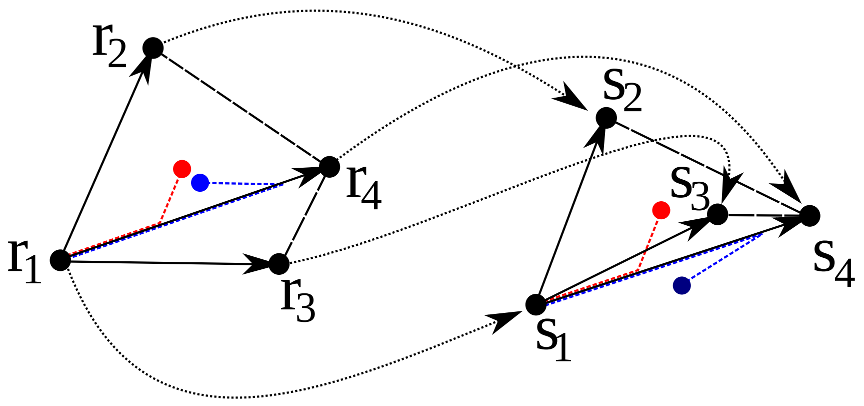


Figure 4.2.: On the left side we have the linear combinations for the red and blue point. The nearest neighbors for the red point are r_1, r_2 and r_3. However the nearest neighbors for the blue point are r_1, r_3 and r_4. This results into two very different linear combinations and a gap between the red and blue point.

The scaling factors l and k can be calculated by

$$l = \frac{r^x - r^x_1 - \frac{r^y - r^y_1}{r^y_{12} r^x_{12}}}{r^x_{13} - \frac{r^y_{13}}{r^y_{12} r^x_{12}}} \tag{4.5}$$

$$k = \frac{r^y - r^y_1 - l r^y_{13}}{r^y_{12}}. \tag{4.6}$$

Afterward, we look up the corresponding points s_1, s_2, and $s_3 \in G$ and compute the difference vectors:

$$s_{12} = s_2 - s_1 \tag{4.7}$$

$$s_{13} = s_3 - s_1. \tag{4.8}$$

The corresponding point s of r can finally be expressed by a linear combination of s_1, s_2 and s_3:

$$s = s_1 + l s_{12} + k s_{13}. \tag{4.9}$$

A downside of this approach is the fact that it cannot handle overlapping triangles. That means if we have well aligned triangles on the floorplan, like in Figure 4.2, and corresponding overlapping triangles on the gridmap there

will be a gap between two adjacent triangles.
The next section proposes an approach which can deal with this problem.

4.2. Global Transformation

The previous approach had problems to create a continuous function which maps points from one map onto another. The problem comes from the fact that we did not care for neighboring point pairs and ignore the dependency of adjacent points. This time we take this problem into account and calculate a transformation on the whole point set. Therefore, we apply the algorithm explained in Section 3.2.2 and combine it with Section 3.2.1. The global transformation approach has three steps: In the first one we randomly choose three points from F. In step two we calculate with these three points and their reference points a transformation T by applying SVD from Section 3.2.1 to it. The third step deals with the consensus set. Every point from F, which agrees with the transformation T, is added to the consensus set. At the end the transformation with the biggest consensus set is the actual transformation which is used to calculate the reference point from a point on the floorplan.
This strategy leads to a very smooth function without any gaps. It also can interpolate points which are far away from any seen point. A downside still exists: For the sake of a smooth function we ignored that the gridmap could be stretched or bent on some parts and a global transformation cannot handle such local exceptions. The next section conquers the last issue.

4.3. Combination of Local Transformations

In the last section we had problems to deal with local variations which cannot be explained by a global transformation. In this approach we separate the points from the floorplan, cluster them and compute a local transformation for each cluster. Therefore, the approach consists of two parts. In the first step we cluster our point set F and in the second part we calculate

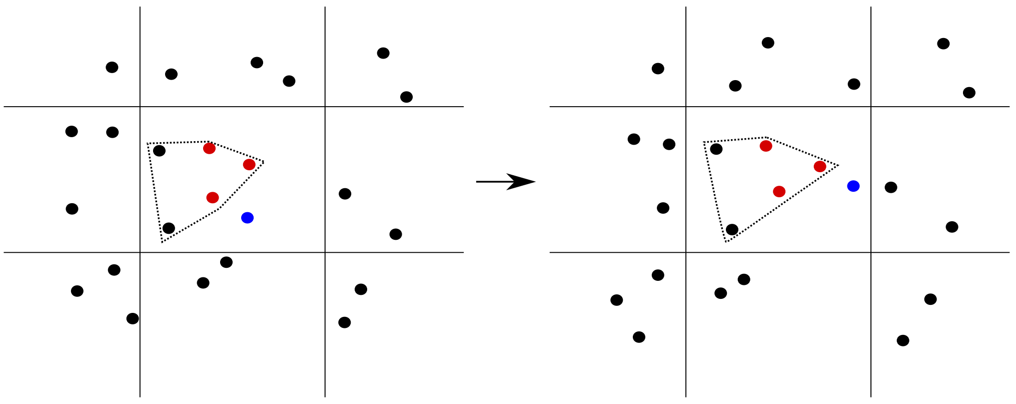


Figure 4.3.: Based on the red points we compute a transformation T. The two black points which are located in the cluster (dotted polygon) are in accordance with the transformation. The blue point does not.

transformations based on the clusters.

4.3.1. Clustering

For the clustering we perform a bottom up approach. On the lowest level we split the floorplan into equally sized cells Z_i. In each cell Z_i we randomly choose three points and compute a transformation T_k based on 3.2.1. Afterward, we start with a new cluster C_k, add these three points to it and start a region growing process. Every other point p of the cell Z_i is added to the cluster if the error, introduced by the transformation, with respect to its reference point q, is lower than a given threshold and does not overlap with another cluster. Figure 4.3 shows this case: The red points are the initialization of the cluster. All points, which are inside of this cluster, agree with their transformation. On the other hand, the blue point does not. This is repeated as long as there are no other valid points left in cell Z_i. Subsequently, the first combination step is performed. We check if two clusters of neighboring cells have a similar transformation. If that is the case they are merged. Thereafter, we jump to the next layer with a coarser grid and redo the same step over and over again.

4.3.2. Combined Transformations

We use the set of clusters C previously calculated to map a point p from the floorplan to the gridmap. Each cluster has its own transformation and we combine these transformations with a weighted sum. Given a cluster C_k, its geometric mean c_k^m and a point p, which we want to transform, we can compute the weight as:

$$w_k = \frac{1}{\|p - c_k^m\|}. \tag{4.10}$$

Hence, it is inversely proportional to the distance to the geometric mean of cluster C_k. Finally, the transformation of a point q can be described by:

$$T = \sum_{i=0}^{N} w_k T_k, \tag{4.11}$$

$$q = Tp. \tag{4.12}$$

where q is the transformed point p on the gridmap.
So far, we used a single or multiple transformations to describe the mapping between the gridmap and the floorplan. In the next section we do not transform the point p anymore but calculate the most likely point q on the gridmap using a Gaussian process.

4.4. Matching with Gaussian Processes

In the last sections we had various problems: In Section 4.1 we had problems with a transformation which was not smooth on the whole map. Using a global transformation (Section 4.2) yields a mapping which is not capable of handling local variations. However, we described in Section 4.3 an algorithm which could deal with local variations. In this section we see another approach which can tackle this issue. The resulting mapping of the points from the floorplan to the gridmap is both smooth and adapts to local variations. In Section 3.3 we discussed the fundamental idea and structure of Gaussian processes. In Figure 4.4 we see a point p and its neighbors. The

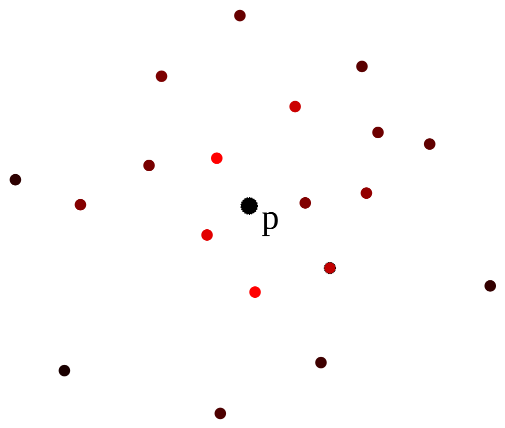


Figure 4.4.: The different shades of red correspond to the correlation to the point p. Red is a strong correlation while black is a weak one.

different shades of red depicts the correlation to p where red is a strong correlation and black a weak one. Thus, it has a bigger influence on the calculation of the corresponding point q on the gridmap if it is red.

4.4.1. Gaussian Process

The first approach with Gaussian processes (GP) computes a system which operates on unedited data. Let F be a set of points on the floorplan and G a set of points on the gridmap. Each point in F has a corresponding point in G which is given by the set FG. In general, a Gaussian process computes a function $\mathbb{R}^n \to \mathbb{R}$. Therefore, we calculate the x and y coordinate of a corresponding point q on the gridmap independently and combine them afterward. To optimize the hyperparameters of Gaussian processes we apply RPROP as described in Section 3.3.1.
To compute q on the gridmap we first have to get K as it is given in Formula

3.17:

$$K = \begin{pmatrix} k(p_1,p_1) & k(p_1,p_2) & \dots & k(p_1,p_k) \\ k(p_2,p_1) & k(p_2,p_2) & \dots & k(p_2,p_k) \\ \vdots & \vdots & \ddots & \vdots \\ k(p_k,p_1) & k(p_k,p_2) & \dots & k(p_k,p_k) \end{pmatrix}, \tag{4.13}$$

where $p_1, \cdots, p_k \in F$ and $k = |F|$. We also need K_* and K_{**} from Formula 3.18 and 3.19:

$$K_* = \begin{pmatrix} k(p,p_1) & k(p,p_2) & \dots & k(p,p_k) \end{pmatrix} \tag{4.14}$$

$$K_{**} = (k(p,p))\,. \tag{4.15}$$

The final step is the calculation of q that maximizes $p(q^x|q^x_1, \cdots, q^x_n)$. To achieve this, we compute the mean:

$$\overline{q}^x = K_* K^{-1}(q^x_1, \cdots, q^x_n)^T. \tag{4.16}$$

We apply the same procedure to get the y coordinate of q.
One downside of this approach is a poor interpolation of points which are far away from any point in F. The Gaussian process tends to converge to a zero mean which does not represent the correct transformation from the floorplan to the gridmap. Hence, we apply an adaption to this algorithm in the next section.

4.4.2. Gaussian Process with Global Transformation

We previously discussed the problem with points which do not have any close neighbors. One solution is to increase the length parameter l. It regulates the amount how much neighbors impact on a point p. A bigger value increases the range where neighbors are stronger correlated to a point p whereas a smaller value leads to less neighbors. Another downside of an increased length parameter is the loss of local variations. The higher the value is the

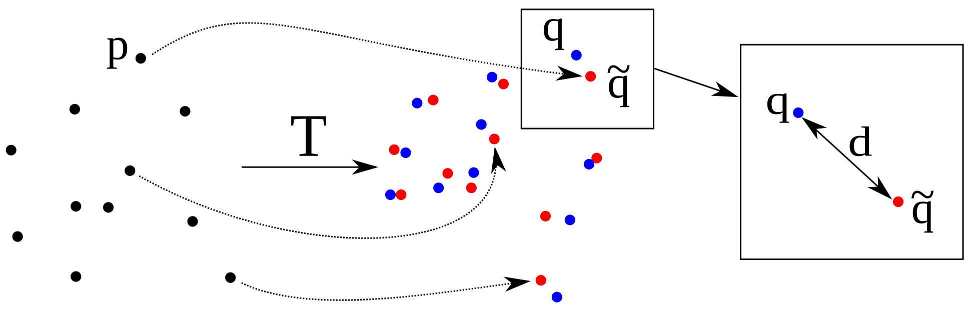


Figure 4.5.: First we transform each point p with the global transformation to the red point $\tilde{q}$: $\tilde{q} = Tp$. Afterward, we compute the difference vector d between the blue correspondence q and the transformed $\tilde{q}$: $d = q - \tilde{q}$. Therefore the Gaussian process learns with the tuple (p, d).

more we get a global transformation. On the other hand, if a point has no correlated neighbors its transformed point on the gridmap tends to go to the mean. To avoid this issue we preprocess the points from the set F and G where the points from F have a reference point in G due to the set FG. Figure 4.5 shows the basic idea. First of all we compute a global transformation T as described in Section 4.2. With T we transform every point p_i from the floorplan to the gridmap:

$$\tilde{q}_i = Tp_i. \tag{4.17}$$

Afterward, we calculate the difference between q_i and $\tilde{q}_i$ where q_i is the corresponding point from p_i on the gridmap:

$$d_i = q_i - \tilde{q}_i. \tag{4.18}$$

Thus, we learn a Gaussian process with input (p_i, d_i) and because of that we no longer work on an absolute value but on the difference vector between q_i and $\tilde{q}_i$. Finally we can calculate a corresponding point q_i of p_i by transforming the point p with the global transformation and adding the difference vector of the Gaussian process in Formula 4.16:

$$q_i = Tp_i + \overline{d_i}. \tag{4.19}$$

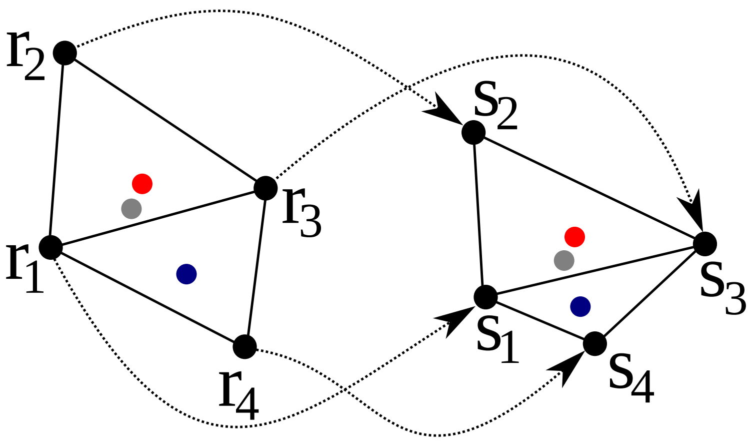


Figure 4.6.: Two triangles with the Delaunay property. The red, blue and gray point are transformed with respect to their Barycentric coordinates. The points r_i on the left side have a corresponding point s_i on the right side.

This adaption solves the problem with a wrong interpolation of points which are not close to any neighbors. These points depend mostly on the global transformation which is the best guess for them if we do not have any further information about the mapping.
The next section explains how we create our ground truth.

4.5. Mapping with a Triangulation

In this section we discuss how we compute a ground truth to compare it to our other approaches. We have a point p which is on the floorplan and we want to get the transformed point q on the gridmap which corresponds to p. Hence, the key idea is that we first search for the triangle M in which our point p lies. Figure 4.6 shows an example where the triangle is given by $\overline{r_1r_2r_3}$ and is enclosing the red point. Afterward, we calculate the barycentric coordinates l, k, and m of the point within the triangle M, look for the corresponding triangle N on the gridmap (e.g. $\overline{s_1s_2s_3}$) and use the same barycentric coordinates l, k, and m to get the point within the triangle N on the gridmap. The first step involves the creation of a Delaunay triangulation DT on the set of points F from the floorplan described in Section 3.4. The second step involves to find the triangle of DT in which the point p lies where p is a point from the floorplan. If we get a match we calculate the barycentric coordinates [7] l, k and m of p within the triangle $\overline{r_1 r_2 r_3}$:

$$l = \frac{(r_2^y - r_3^y)(r^x - r_3^x) + (r_3^x - r_2^x)(r^y - r_3^y)}{(r_2^y - r_3^y)(r_1^x - r_3^x) + (r_3^x - r_2^x)(r_1^y - r_3^y)}, \tag{4.20}$$

$$k = \frac{(r_3^y - r_1^y)(r^x - r_3^x) + (r_1^x - r_3^x)(r^y - r_3^y)}{(r_2^y - r_3^y)(r_1^x - r_3^x) + (r_3^x - r_2^x)(r_1^y - r_3^y)}, \tag{4.21}$$

$$m = 1 - l - k. \tag{4.22}$$

Similar to Section 4.1 we obtain q with the correspondences of p_1, p_2 and p_3:

$$q = lq_1 + kq_2 + mq_3. \tag{4.23}$$

However, we need the triangulation generated by the correspondences on the gridmap to be non-overlapping. Otherwise, we end up in the same configuration as in Section 4.1 and get a discontinuous function. To avoid this issue we generate the point pairs by hand because this approach can not handle overlapping triangles. This is a difference from the other approaches which use automatic generated point pairs. Therefore, we discuss this automatic generation of point pairs in the following section.

4.6. Approximated Reference Pairs

All algorithms in Chapter 4 depend on two sets of points F and G where every point in F has a corresponding point in G. Therefore, we discuss in the following section how we compute our two sets of points and how we create the connection between them. In Figure 4.7 we see two maps. The left one is a floorplan and the right one a gridmap. The red dot is the robot position on both maps and the blue dots represent the endpoints of the laser scan. Each blue point on the left side has a corresponding point on the right side. The basic idea is that we drive a robot through a building and track the position of the robot on both the floorplan and gridmap with a particle

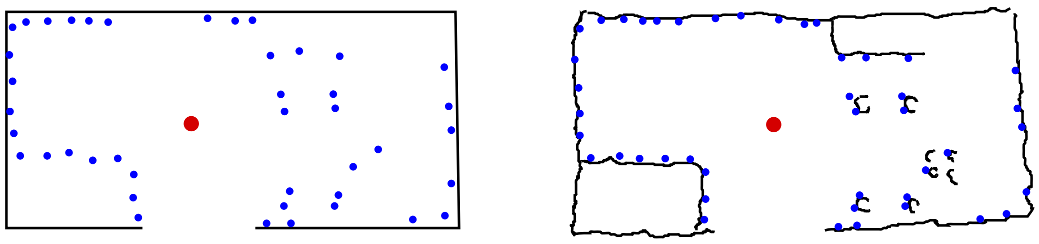


Figure 4.7.: On the left we have the floorplan and on the right side a corresponding gridmap. The red dot represents the position of the robot and the blue points are the endpoints of the robots' laser scan. Each blue point on the left side has a corresponding point on the right side. These are our point pairs.

filter. We take each laser scan, produced by the robot, and compute the position of every endpoint of the laser scan with respect to the current robot position on the maps. Hence, we have our corresponding points. However, we see that both maps look very different. Typically, a floorplan consists of the buildings' structure whereas a gridmap has much more details like chairs, tables, cupboards, machines, and other stuff in its map. We discuss how that impacts the results in Chapter 6.

4.6.1. Calculation of Reference Pairs

We discussed in Chapter 4 how we compute a function which maps points from the floorplan onto the gridmap. To calculate this function we implemented several approaches. However, to create them we need a set of points on the floorplan and we need to know where the corresponding points on the gridmap are. We track a robot on both the floorplan and gridmap with the Monte Carlo localization from Section 3.5.4. For this task we have two possibilities: Either we could first localize our robot and then track it or we start with a known pose and track it immediately. In both ways we start to generate reference pairs when the robot is in the tracking stage. Hence, we have a particle set on both maps. To compute the actual robot position from these sets we apply a weighted mean approach. Therefore, we have:

$$p^x = \sum_i^N w_i p_i^x \tag{4.24}$$

$$p^y = \sum_i^N w_i p_i^y \tag{4.25}$$

$$p^\theta = \arctan\left(\sum_i^N w_i \sin p_i^\theta, \sum_i^N w_i \cos p_i^\theta\right), \tag{4.26}$$

where p_i is a particle, w_i the respective weight computed in the observation model and N the number of particles. In every update step of the Monte Carlo localization we apply this mean calculation and get the robot position on both the floorplan and gridmap separately. These two positions serve as the starting points to get the laser endpoints of the current laser scan. For every measured distance we compute the position on the floorplan and on the gridmap and add them to the reference pair set. Therefore, during the tour of the robot through a building we generate a lot of reference pairs which we use in Chapter 4 to compute the mapping function.
In the next Chapter we discuss some implementation details and the structure of the applied algorithms.

5 Implementation Details

The next two sections are about the structure and details about the implementation of the algorithms described in the previous chapters.

5.1. Combined Transformations with Clusters

The combination of transformations in Section 4.3 consists of two parts: First we have to calculate a clustering and second we need to compute on each cluster the transformation. Hence, we look at the basic structure of the clustering. There, we divide the map into equally sized cells and calculate clusters in each cell. These have to be non-overlapping. Then, we combine, with respect to the transformation, similar clusters by merging two cells. Here, P is the point set which lies within the cell Z and C is the cluster. Using the Algorithm 5.1 we can compute the clusters and the corresponding transformations. We want the corresponding point q on the gridmap. Let p be a point on the floorplan. Therefore, we apply the Algorithm 5.1. w_i is the weight in Equation 4.10, C_m is the geometric mean of cluster C_i and w the total weight. Hence, we have computed our reference point q of p with the weighted sum using several transformations.
The next section describes how we implemented the transformations with the Gaussian processes.

Algorithm 1 The clustering of the combined transformations approach.

```
1: **for all** cell $Z$ **do**
2:   **while** $P$ not empty **do**
3:     Add 3 points from $P$ to $C$
4:     Remove these points from $P$
5:     Calculate Transformation $T$ of $C$
6:     **for all** $p$ in $P$ **do**
7:       **if** $p$ agrees with $T$ **then**
8:         Add $p$ to $C$
9:         Remove $p$ from $P$
10:      **end if**
11:      **if** $C$ overlaps with any $C_i$ **then**
12:        Remove $p$ from $C$
13:      **end if**
14:    **end for**
15:  **end while**
16: **end for**
17: **for all** Neighboring cells $C_i$ $C_j$ **do**
18:   **if** $T_i$ of $C_i$ agrees with $T_j$ of $C_j$ **then**
19:     Merge $C_i$ and $C_j$
20:   **end if**
21: **end for**
```

5.2. Transformation with Gaussian Processes

In the last section we looked closer on the structure of combined transformations. In this section we do not compute an exact transformation but the most probable reference point q on the gridmap of a point p on the floorplan. The first part is the learning and optimization process described in Section 5.2. Here, FG is the set of reference points from the floorplan and gridmap and GP the Gaussian process. Hence, we first add all pairs to the Gaussian process and then optimize the length parameter and the variance. To finally calculate a reference point q we apply the Gaussian process from Algorithm 5.2. Hence, we compute the most likely reference for the point p.
We also implemented the variant from Section 4.4.2: Instead of the zero mean strategy we have an initial guess with the global transformation from Section 4.2. Therefore, if all neighbors are not in correlation with the point

Algorithm 2 First we calculate the weight for each cluster and, afterward, we combine the transformations with respect to the weight.

```
1: $q = 0$
2: $w = 0$
3: for all Cluster $C_i$ do
4:     $w_i = \frac{1}{dist(p,C_m)}$
5:     $w = w + w_i$
6: end for
7: for all Cluster $C_i$ do
8:     $w_i = \frac{w_i}{w}$
9:     $q = q + w_i T_i p$
10: end for
```

Algorithm 3 We add all reference pairs from the set FG to the Gaussian process and optimize the parameters with RPROP.

```
1: for all $pq$ in $FG$ do
2:     Add $pq$ to GP
3:     Calculate covariances of $pq$
4: end for
5: Optimize GP with RPROP
```

p we transform it with the global transformation as a fallback strategy. Thus, the structure changes to the Algorithm 5.2. Hence, we first transform the point p onto the gridmap using the global transformation and then initialize the Gaussian process with the difference vector d between the transformed point $\tilde{q}$ and q. For the calculation of a reference point we first have to transform the point and then add the output of the Gaussian process to it.

In the next chapter we discuss how well the different approaches perform in comparison with the ground truth.

Algorithm 4 The calculation of the reference point q.

```
1: Calculate $K_*$ and $K_{**}$ of $pq$
2: $q = K^* K^{-1} (q_1, \cdots, q_n)^T$
```

Algorithm 5 The adapted variant of the Gaussian process with a global transformation as fallback strategy.

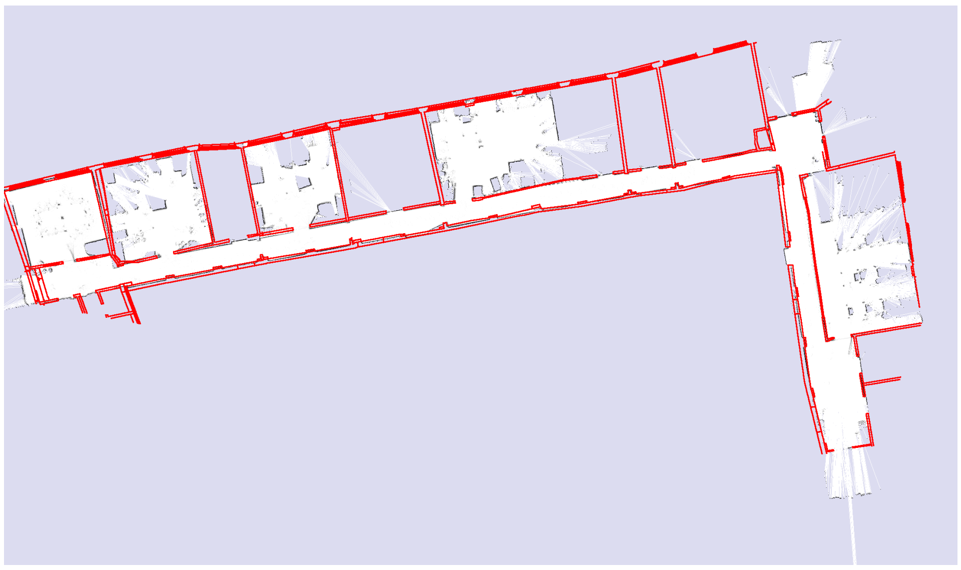
```
 1: Calculate global transformation T
 2: for all $pq$ in $PQ$ do
 3:     $d = q - Tp$
 4:     Add $pd$ to GP
 5:     Calculate covariances of $pd$
 6: end for
 7: Optimize GP with RPROP
 8: $\tilde{q} = Tp$
 9: Calculate $K_*$ and $K_{**}$ of $pd$
10: $q = \tilde{q} + K^* K^{-1} (d_1, \cdots, d_n)^T$
```

6 Experiments

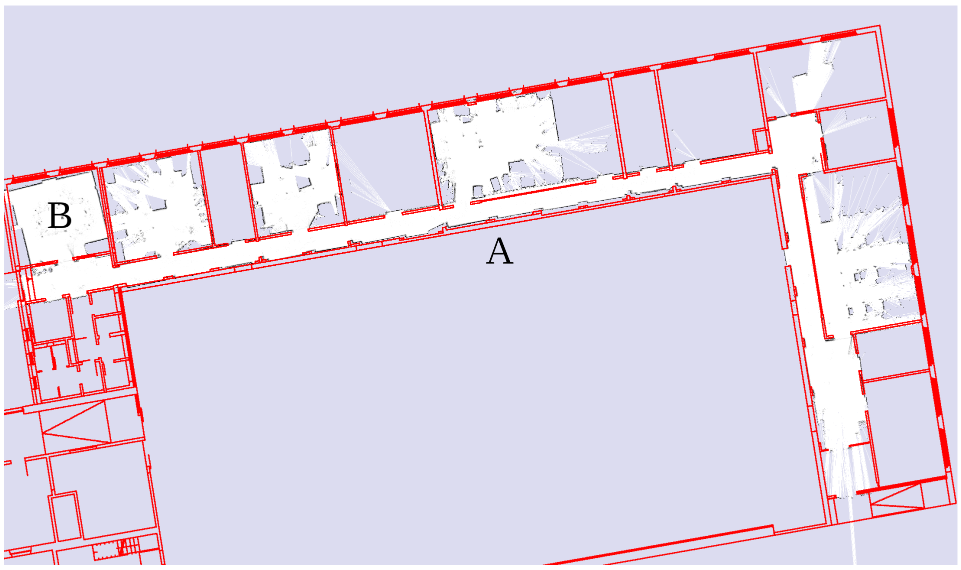
We proposed several approaches to calculate a function or transformation which projects a point p on a floorplan to a point q on the gridmap and vice versa. To learn these projections we described a set of algorithms in Chapter 4. They either compute a transformation, use an interpolation or calculate the most likely point q, given some data, on the gridmap. In the following chapter we evaluate the output of these algorithms and compare it with a ground truth. Therefore, we have multiple floorplans and the respective gridmaps. We recorded the output from a laser scanner attached to a Pioneer robot and the odometry. Afterward, we built multiple gridmaps from these data sets and generated the set of reference pairs by applying the proposed algorithm from Section 4.6. We used these reference pairs to apply our approaches and compute a transformation.
The first experiment is about the performance of each algorithm in comparison to the ground truth.

6.1. Comparison to the Ground Truth

The first experiment is about the application of the approaches to the different data sets. We compare the results to the ground truth which we generate by manually setting points on the floorplan and its corresponding points on the gridmap. Through these point pairs we compute a Delaunay triangulation as described in Section 3.4. However, we set only points on positions where the gridmap is available. We can see the results in Table 6.1 and the



(a) Delaunay Triangulation



(b) Global Transformation (GT)

Figure 6.1.: The top picture shows the ground truth of the floorplan matched into the gridmap. The bottom picture is an approach with a global transformation. The meaning of the two positions A and B will be discussed in the text.

Approach	Building 79	Building 74	Building 106	Factory Floor
GT	0.265 m	0.226 m	0.157 m	0.151 m
CT	1.705 m	0.458 m	-	-
GP	0.692 m	0.455 m	0.172 m	0.170 m
GPGT	0.419 m	0.268 m	0.173 m	0.202 m

Table 6.1.: The mean error of all approaches.

following sections present the individual results on the different data sets.

6.1.1. Building 74

The data set of building 74 consists of a trajectory through a floor and office environment where we visited a few rooms. In the top picture of Figure 6.1 we see the ground truth. We take the floorplan and map every wall of it into the gridmap. The result is a gridmap with a red overlay which represents parts of the floorplan of building 74. As we mentioned before: We just take walls of the floorplan which are inside of a triangle from the Delaunay triangulation and map them onto the gridmap. The lower picture shows the approach with the global transformation (GT) from Section 4.2. It takes all points and the corresponding ones into account and computes a transformation. We can see that the global shape fits into the gridmap. However, it does not consider local details. For example the transformed plan at A does not fit onto the gridmap. At position B we have a slightly rotated room. The GT misses this detail as well. The mean error of this approach is 0.226 m with a map resolution of 0.025 m. Here, the mean error is the summed up error of every sample divided by the total amount of samples. The samples are the wall-points on the floorplan. The next Figure 6.2 is about the approach where we combine several local transformations (CT) and weight each of them with respect to the distance to the cluster center. It is able to adapt to the rotated room at position A and the hallway at B as well. However, the global performance is not as good as in the previous approach. Hence, it

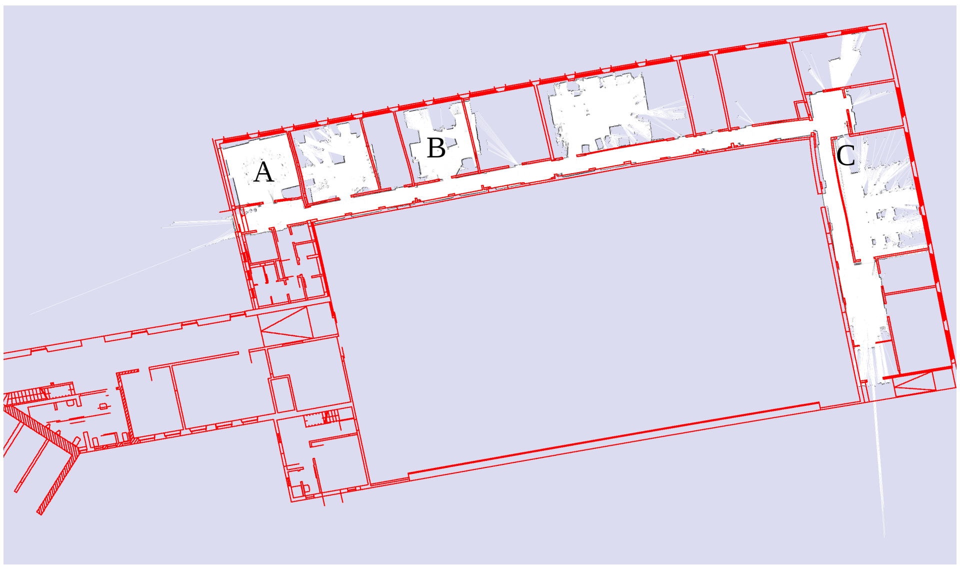
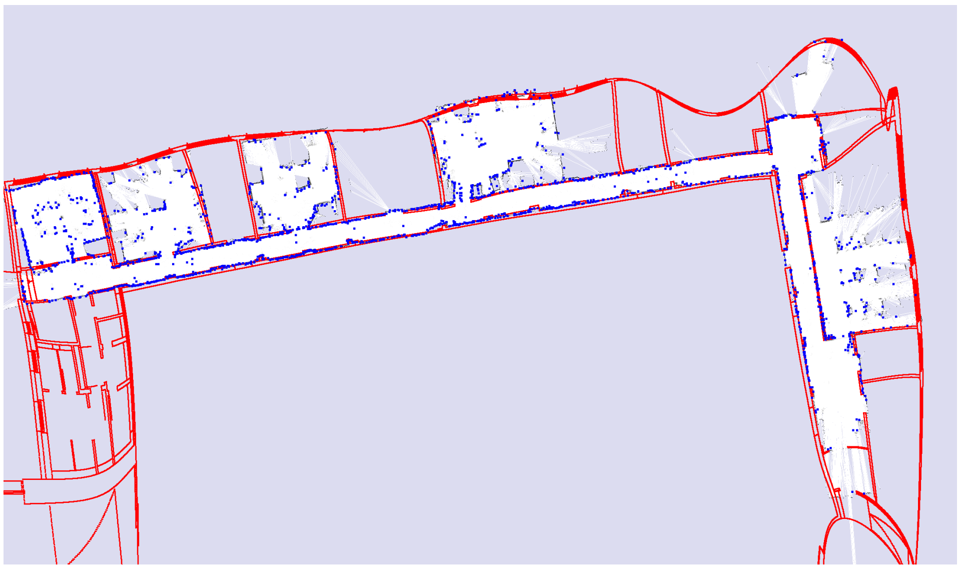
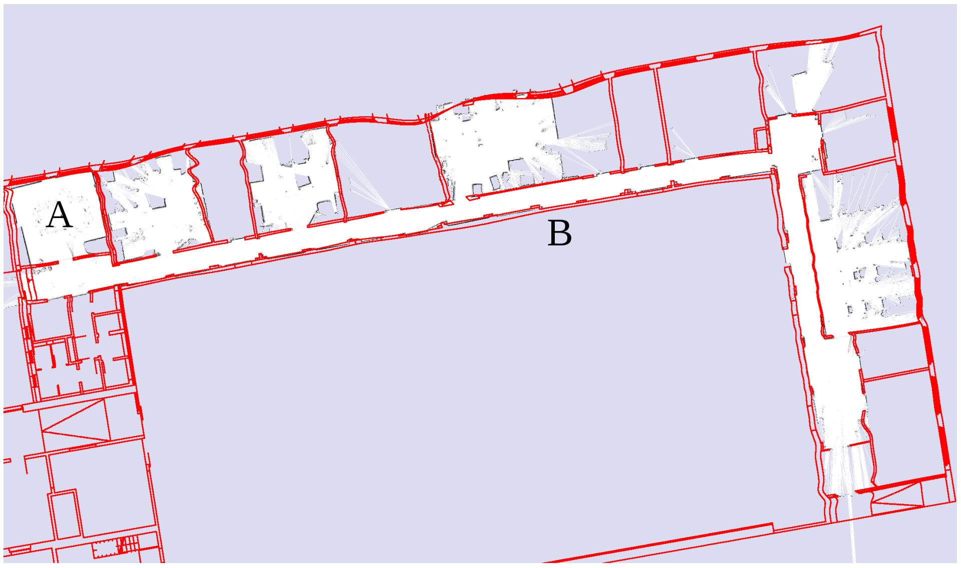


Figure 6.2.: The floorplan is mapped onto the gridmap through a combination of transformations.

shifts the floorplan at position B and C. The mean error is 0.458 m. Figure 6.3 shows two approaches based on Gaussian processes. Here, the top picture is the Gaussian process (GP) described in Section 4.4 without any preprocessing. The blue points represent the reference points. At the first sight it looks like a bad transformation. However, a Gaussian process heavily depends on samples in a close region to interpolate a point. Hence, if we have no points in a small region around our point p from the floorplan we cannot find a good counterpart on the gridmap. We see this issue at the curved parts of the floorplan which we mapped onto the gridmap. But as long as we stay close to samples the performance is close to the correct position. Due to the big errors in areas without any samples the mean error is 0.455 m. To deal with this issue we proposed in Section 4.4.2 an adaption. We first transform all points from the floorplan onto the gridmap with a global transformation and, afterward, learn a Gaussian process on the difference between the transformed point p and the reference point q on the gridmap. We call it GPGT. The lower picture shows the result. Areas without any samples are closer to the global transformation which is, without any further knowledge, a good approximation. Hence, the global shape depends on the global transforma-



(a) Gaussian Process (GP)



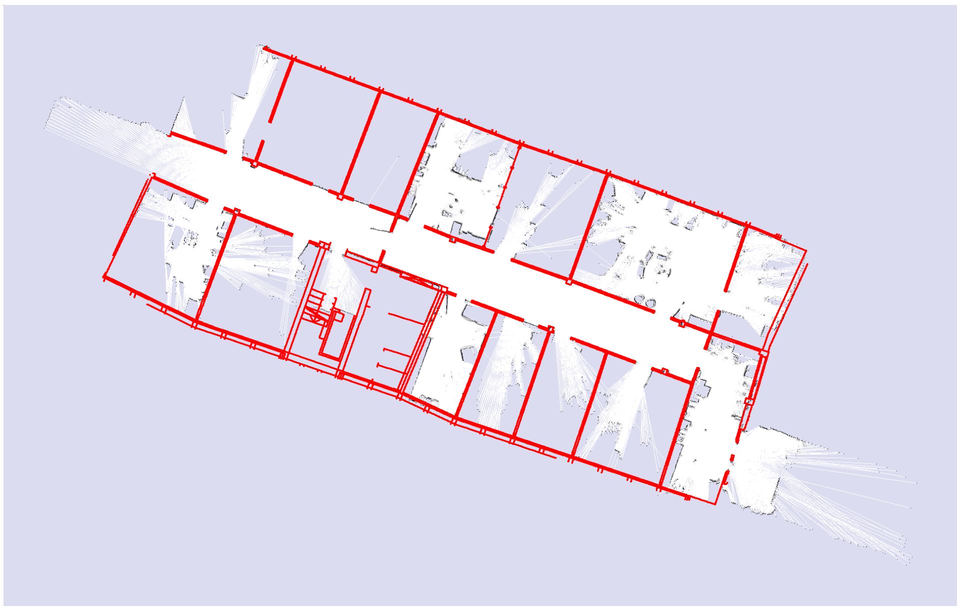
(b) Gaussian Process with Global Transformation (GPGT)

Figure 6.3.: The top picture shows the performance of a Gaussian process. The blue points represent the reference points. The lower picture is the Gaussian process where we first transform the points with a global transformation and learn on top of it a Gaussian process.

tion with local variations due to the Gaussian Process. It is able to adapt the rotated room at position A and also aligns the hallway at B to the appropriate one on the gridmap. The mean error is 0.268 m which is slightly higher than the error from the global transformation.
We see that the GT has the best result on this data set. The reason is that, beside one slightly rotated room, the whole map has no bad alignments. Therefore, a GT is a good approximation. On the other hand, the GPGT approach can deal with the rotated room. It adapts a global transformation and, hence, is the second best approach on this data set.

6.1.2. Building 79

Building 79 is an office building that has a lot of rooms and a connecting hallway. Figure 6.4 shows the ground truth and the first approach with a global transformation. In comparison to the ground truth the GT approach did not adapt the scaling factor correctly. On the left side of the picture we have a higher offset compared to the other side. Therefore, the walls of the floorplan do not match with the walls of the gridmap. Despite this issue, the gridmap is well aligned. The mean error is 0.265 m. The next approach is the CT one. In Figure 6.5 we see the result. It is globally shifted downwards and does not align very well with the structure. Therefore, we have a bigger mean error of 1.075 m. Both Gaussian process variations depend heavily on sample density. Therefore, we have one more time a lot of curvy lines which are too far away from samples and are not well interpolated. However, in regions with samples the GP in the top picture (Figure 6.6 (a)) fits into the gridmap. Here, the blue points are our reference points. The two learned length scale parameters are 6.26 and 4.96 for the x direction and 5.41 and 5.61 for the y direction. Because of the bad interpolation performance the mean error is 0.692 m. In the lower picture we see the GPGT approach. It fits a lot better to the global shape but is still very curvy between reference pairs. The learned length scale parameters are 4.6 and 3.08 for the x direction and 4.02 and 3.01 for the y-direction. Hence, this approach sticks more to local samples than the previous approach. The mean error is 0.419 m.


(a) Delaunay Triangulation

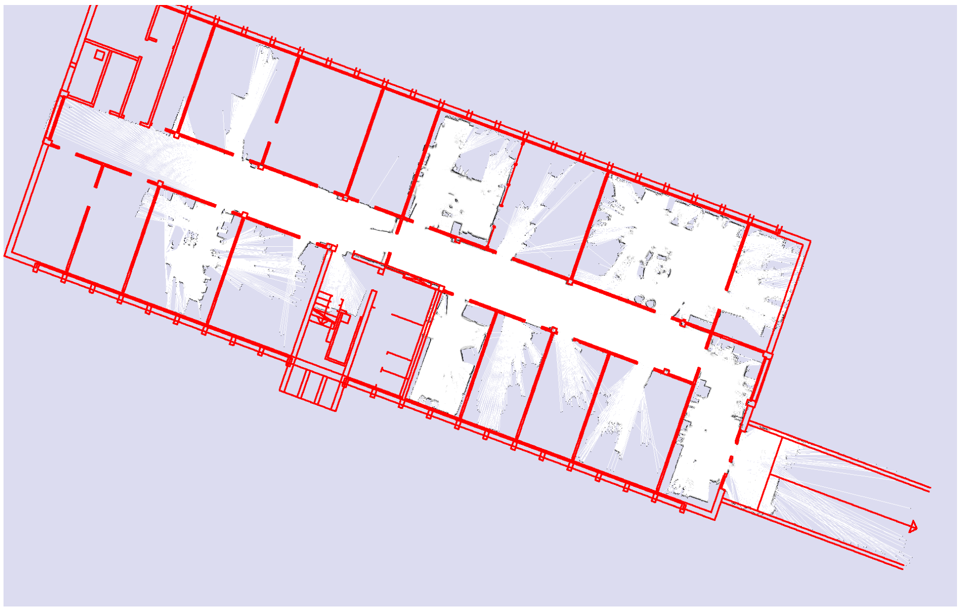

(b) Global Transformation

Figure 6.4.: The first picture shows the ground truth whereas the second one the global transformation.

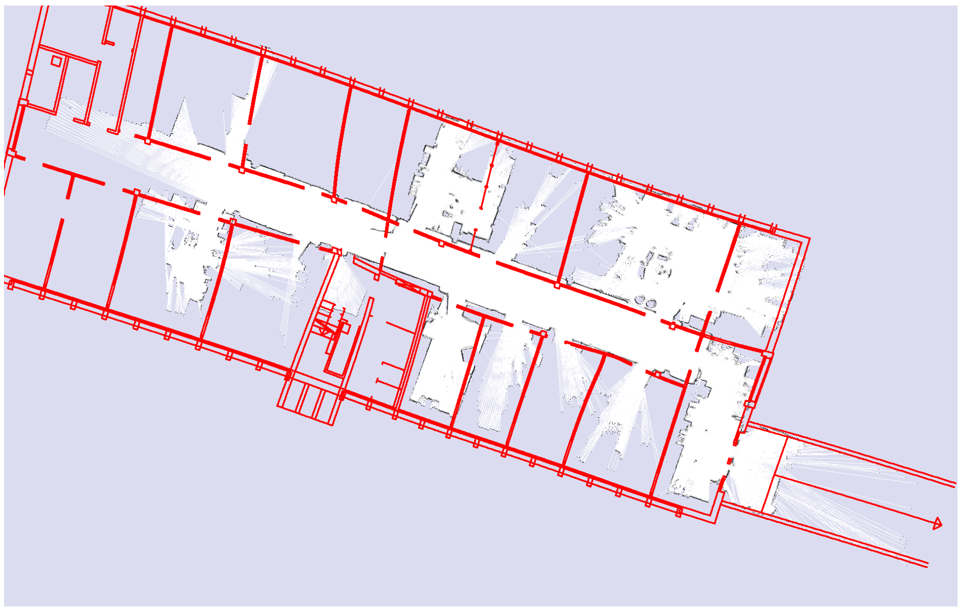
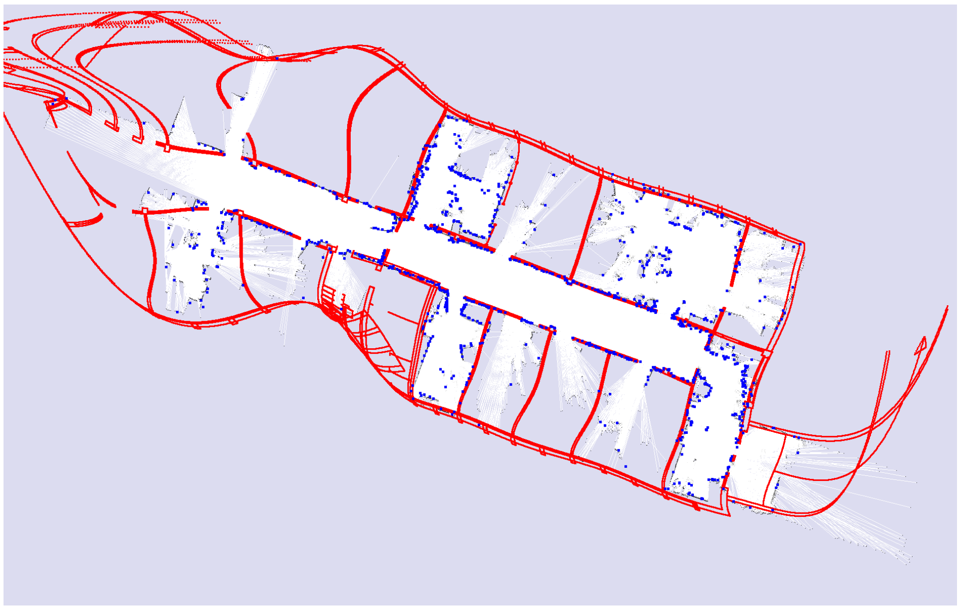


Figure 6.5.: This picture shows the result of the combined transformations approach.

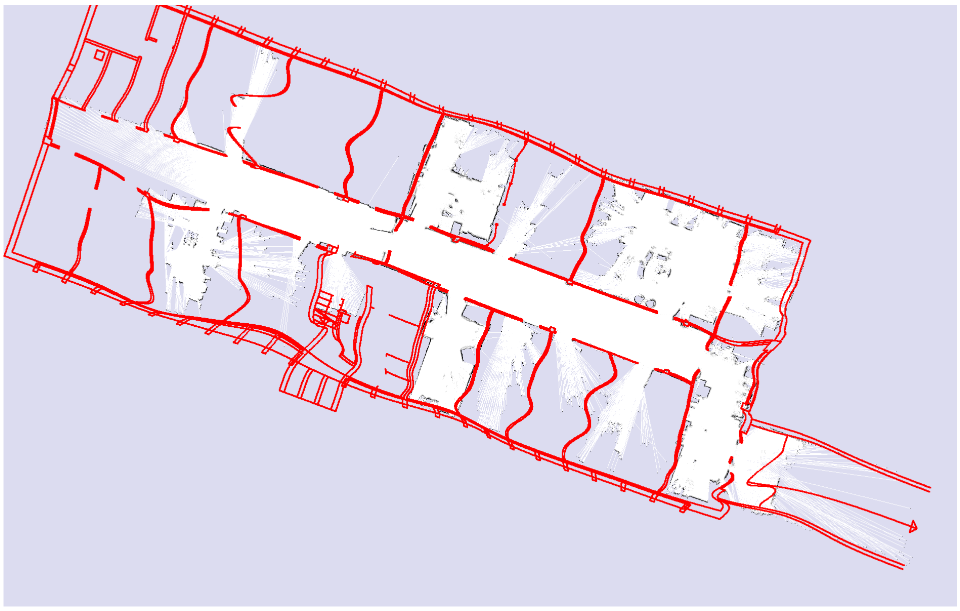
However, if we make the same experiments on the same data set which we use for the ground truth all mean errors decrease. The CT approach is still the worst with a mean error of 0.161 m followed by the GT with a mean error of 0.129 m. We achieve the best results with the GP and the adaption. The GP has a mean error of 0.094 m and the GPGT version a mean error of 0.092 m. Thus, we see that the performance also depends on the given references. However, we have to generate them manually which is not feasible with larger data sets. Therefore, we stick to the automatically generated references from Section 4.6.

6.1.3. Building 106

The data set from building 106 consists of a trajectory through hallways. Therefore, we do not have to deal with many obstacles and furniture. The resulting gridmap and floorplan look very similar. Hence, we have good reference points. The GT in Figure 6.7 shows a good matching of gridmap and floorplan. The result is slightly rotated but does not end up in big errors



(a) Gaussian Process



(b) Gaussian Process with Global Transformation

Figure 6.6.: The first picture is about the pure GP with our reference points in blue. In the other one we see the GPGT.

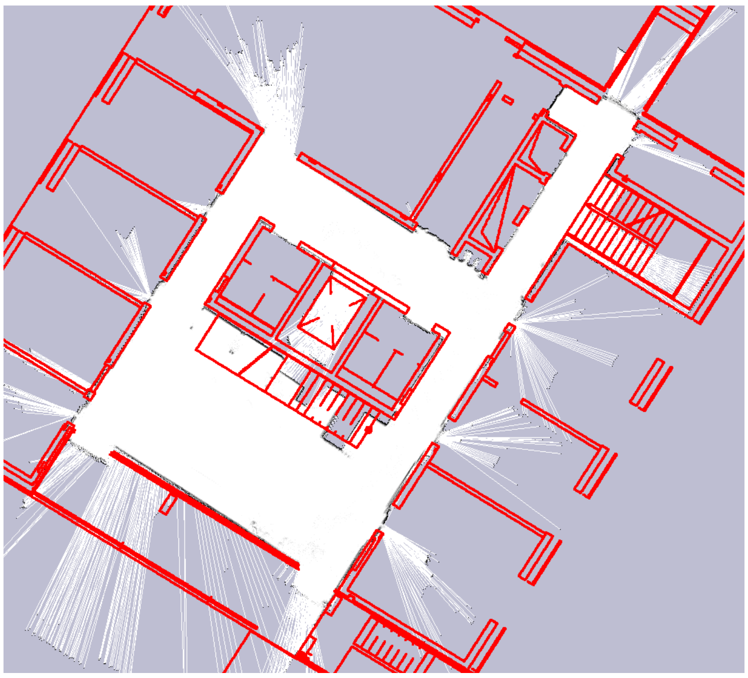


Figure 6.7.: Due to good reference points the global approach creates a good matching of floorplan and gridmap.

due to the small map. The mean error is 0.157 m. However, the GP performs very well, too. It has a mean error of 0.172 m which is very close to the GT approach. We can see the result in Figure A.2 in the appendix. Once more, the only downside are the poor interpolation capabilities in border regions where it shifts away from the optimal position. The next Figure 6.8 illustrates the GPGT. As in former experiments the result looks wavy. However, the mean error is 0.173 m which is close to the other two approaches. The CT approach has difficulties to create a proper clustering. Hence, we cannot evaluate this approach for this data set.

6.1.4. Factory Floor

The factory floor data set consists of four trajectories which all start at the same spot. The hallways are several hundred meters long. The CAD-floorplan contains different machine setups. Here, the problem is that all structures are in the map. Hence, we also have to deal with obstacles which the robot does

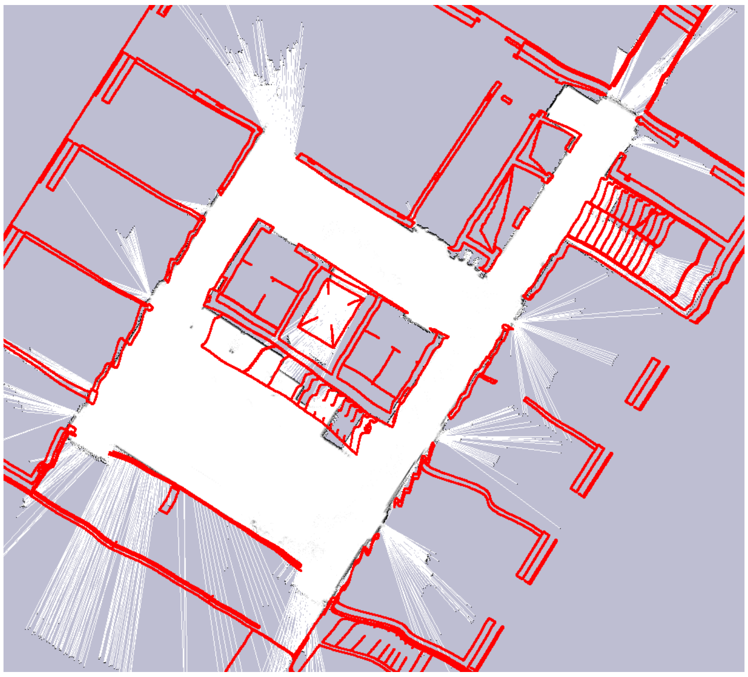


Figure 6.8.: The GPGT performs similarly as the GT.

not see. We also edited the floorplan, such that the robot has a free and empty path and no structure spans over a hallway. Nevertheless, the algorithm for the automatic reference pair generation finished all four trajectories without delocalizing itself. Figure 6.9 shows the result of the GT approach. The mean error is 0.151 m. Therefore, we have the best results on this huge data set. On the other data sets the mean error was between 0.157 m and 0.265 m. One reason for such a good performance are the reference points. In Figure 6.10 we see the reference points which we generated with a particle filter on the floorplan and one on the gridmap. The blue points match the structure of the floorplan. Only at some positions the points are scattered where the particle filter changed over into a multi-modal mode. Because of the good reference points the GP performed well, too. In Figure 6.11 we see the results of it. Around the hallways are no obvious mismatches. The mean error is 0.170 m. The GPGT is a bit less accurate. It has a mean error of 0.202 m. We see the result in Figure A.3 in the appendix. The approach aligns the floorplan as good as the GP to the gridmap. Only one spot (Posi-

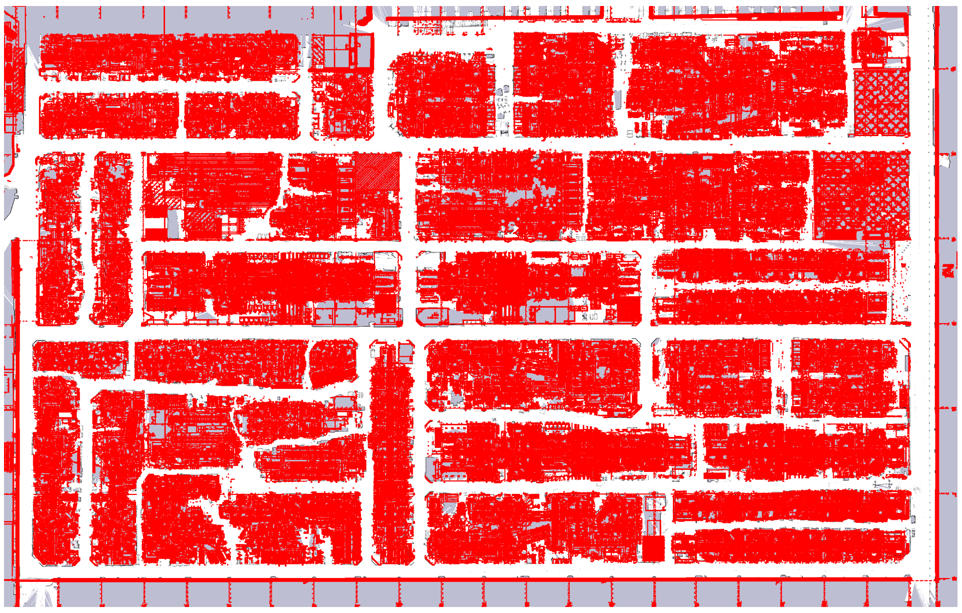


Figure 6.9.: The GT on the factory floor data set.

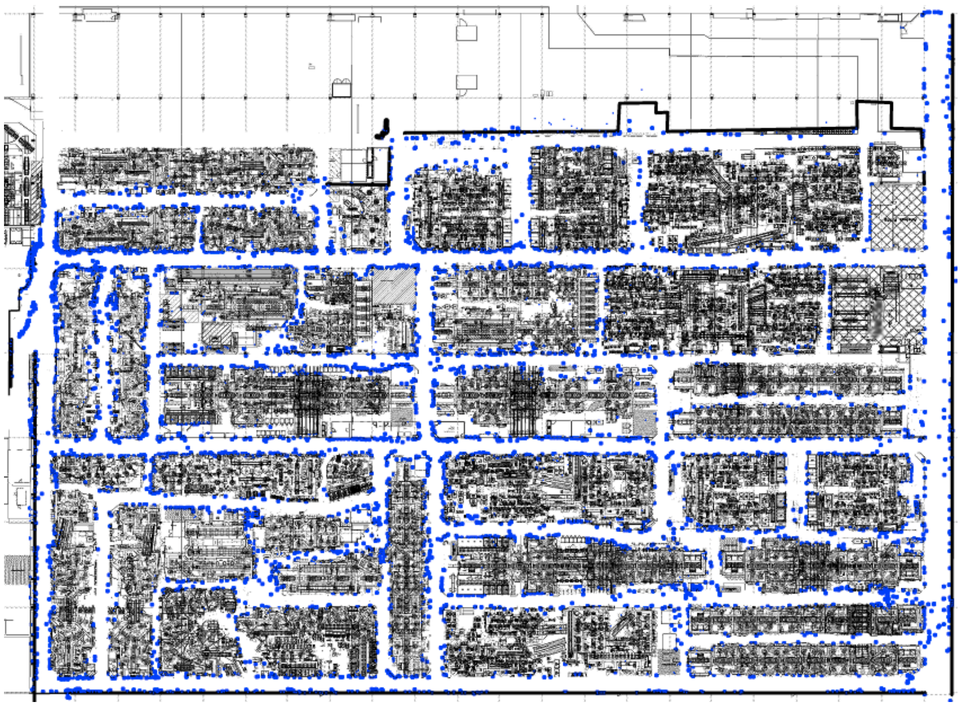


Figure 6.10.: The reference points on the floorplan.

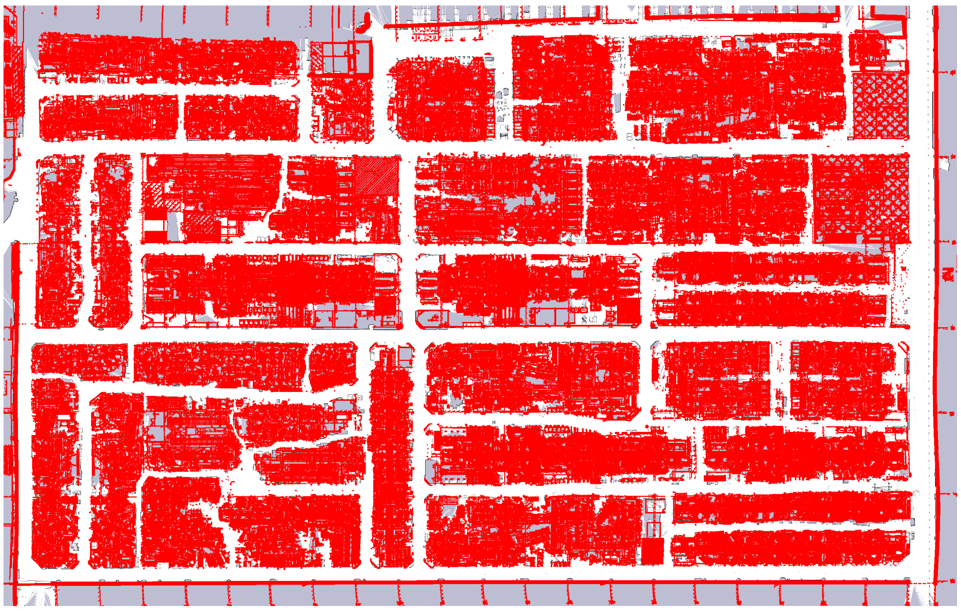


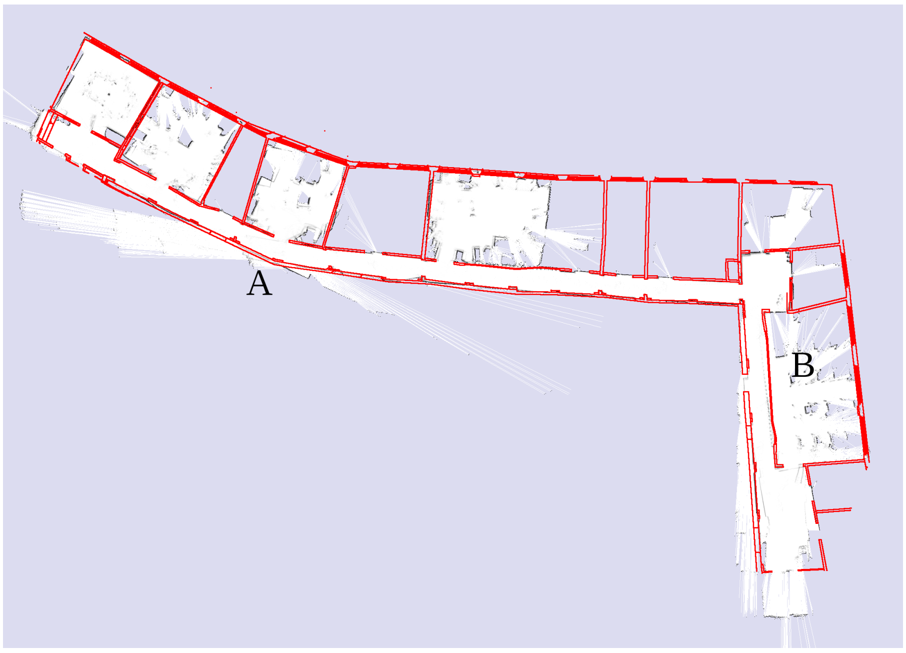
Figure 6.11.: The GP on the factory floor data set.

tion A) is wrong aligned which leads to a higher mean error.

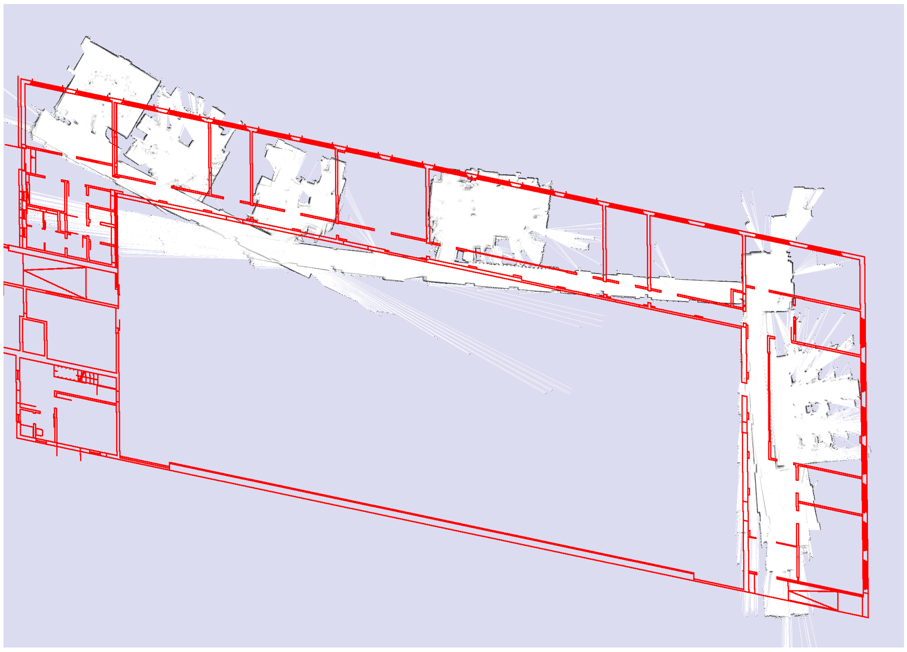
We have seen that the performance of the different approaches is close related to the quality of reference pairs. Especially the GP and GPGT depend on good reference pairs. On the other hand, the global transformation can deal with scattered reference pairs and small errors.
In the next section we see an experiment with an erroneous map of building 74.

6.2. Performance on Erroneous Map of Building 74

We have seen that the performance of a GT is very good in situations where the gridmap is globally consistent. In this section we look at a map for which this no longer holds. Figure 6.12 shows the erroneous map of building 74. The bent, which can be seen in the middle of the map, can happen if the scan matcher creates a wrong connection between two nodes in a graph



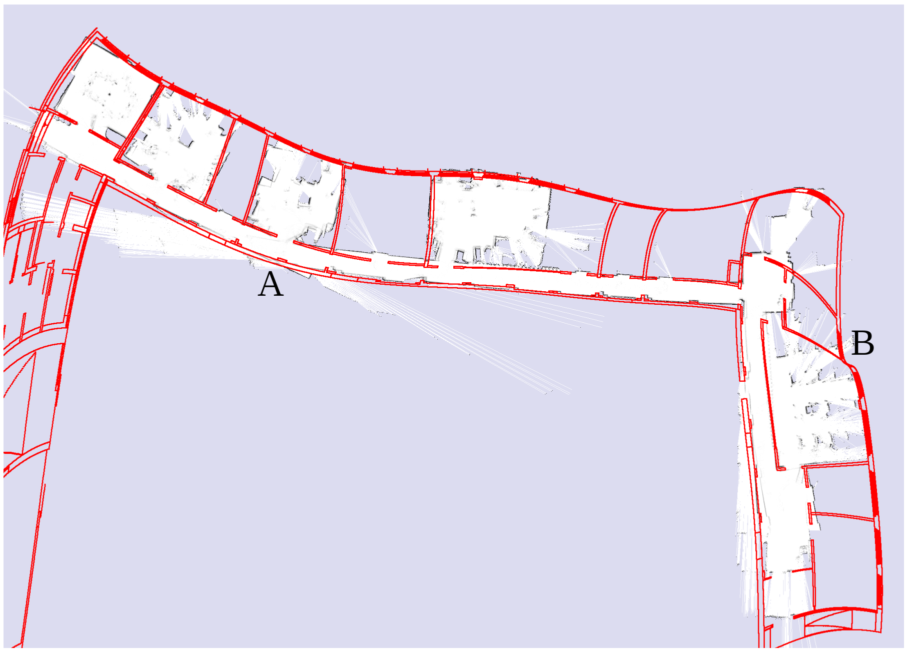
(a) Delaunay Triangulation



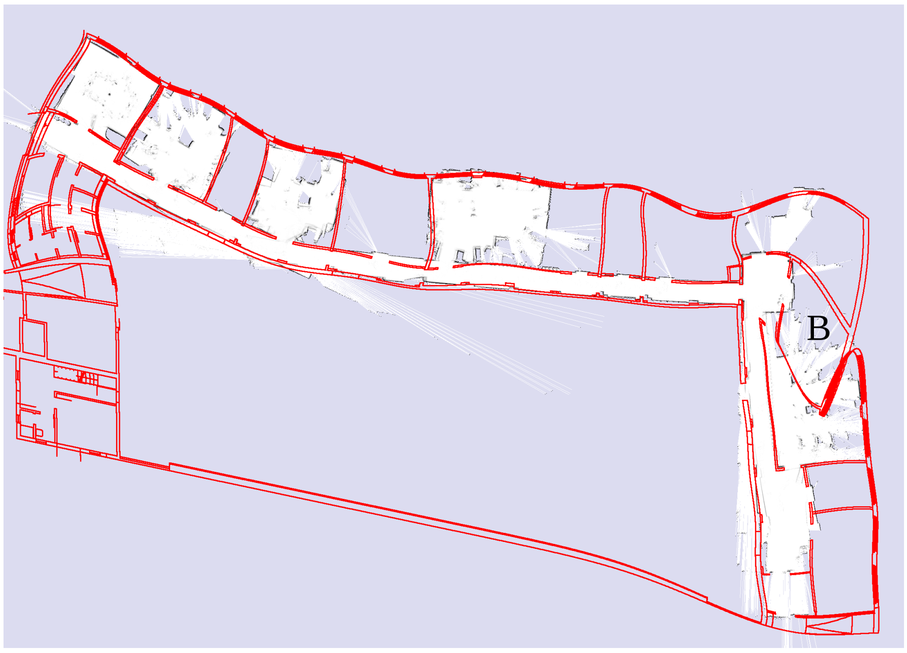
(b) Global Transformation

Figure 6.12.: The erroneous gridmap with the triangulation in the first picture and the GT on the second picture.

based mapping algorithm or a wrong aligned scan in a particle filter based algorithm. If this happens, for example, in an online approach there is no time to restart the mapping but our algorithm has to deal with it. In the top picture we show the ground truth. Hence, we transformed the points from the floorplan onto the gridmap with the algorithm from Section 4.5. The problem with this map is the kink in the area of A and the bend in the area of B. Therefore, we do not have a globally aligned map. As a result the performance of the GT approach is quite bad: It has a mean error of 1.75 m. The lower picture in 6.12 shows the resulting transformed map. It takes all points into account and computes a global transformation which does not follow the alignment of the gridmap. If we use the CT approach we get a problem due to outliers. We illustrate the result in Figure A.1 in the appendix. At A and B we have several outliers from the automatically generated references. Hence, the algorithm incorporates them and creates on top of it a set of transformations. The result is a shift towards this cluster in the area around A. The mean error is 1.81 m. The next two figures in Figure 6.13 show the results of the two approaches based on Gaussian processes. The first two approaches could not align to the local variations which are present in the current experiment. However, the GP as well as the GPGT version can deal with it. The top picture shows the GP. It can deal with the bent at position A and aligns itself with the gridmap. Also, on the right side of the map it fits onto the gridmap except for the area around B where it misses references. The mean error is 0.71 m which is smaller as the error from the previous approaches. The GPGT approach is less accurate having a mean error of 0.798 m which we can see in the lower picture. It can deal with local variations but also suffers from missing references in the area around B. Another downside is the fact that the global constraint is not fulfilled. Therefore, the initial global transformation for every point does not improve the interpolation capabilities of the algorithm. Another comparison can be done if we take the triangulation algorithm into account and apply it on the automatically generated references. The first thing to say is that it cannot deal with any outliers and takes them all into account. Hence, if we have a triangulation with the Delaunay property on the floorplan and look at



(a) Gaussian Process



(b) Gaussian Process with Global Transformation

Figure 6.13.: The first picture shows the result of the Gaussian process. It is able to align to the local variations as well as the second approach with an adapted Gaussian process in the second picture.

APPROACH	Building 79	Building 74	Building 106	Factory Floor
GT	0.20 m	0.20 m	0.10 m	0.15 m
CT	1.00 m+	0.35 m, 0.7 m	0.15 m, 0.85 m	-
GP	1.00 m+	0.15 m	0.20 m	0.15 m
GPGT	0.30 m	0.30 m	0.10 m	0.15 m

Table 6.2.: The main peaks of the approaches.

the matching ones on the gridmap we have overlapping triangles. As a result we get no smooth function and a lot of mismatched points. The mean error is 10.27 m.

The first results illustrate that the GT performs very well on gridmaps which do not contain any global error such as bent or skewed parts. The GPGT beats the Gaussian process due to its better interpolation capabilities in border regions. However, if the map has local variations or is not globally aligned then the GP and GPGT outperforms the GT approach. The linear combination and CT perform both not as good as the other approaches.

6.3. Distribution of Mapped Points

In the last section we looked into the performance with respect to the ground truth. This section is about the distance of each point on the map to the ground truth position. Hence, we map every point of the floorplan onto the gridmap and compare it to the ground truth. Afterward, we count the number how often points fall into a given interval. Table 6.2 illustrates the resulting peaks of the distributions.

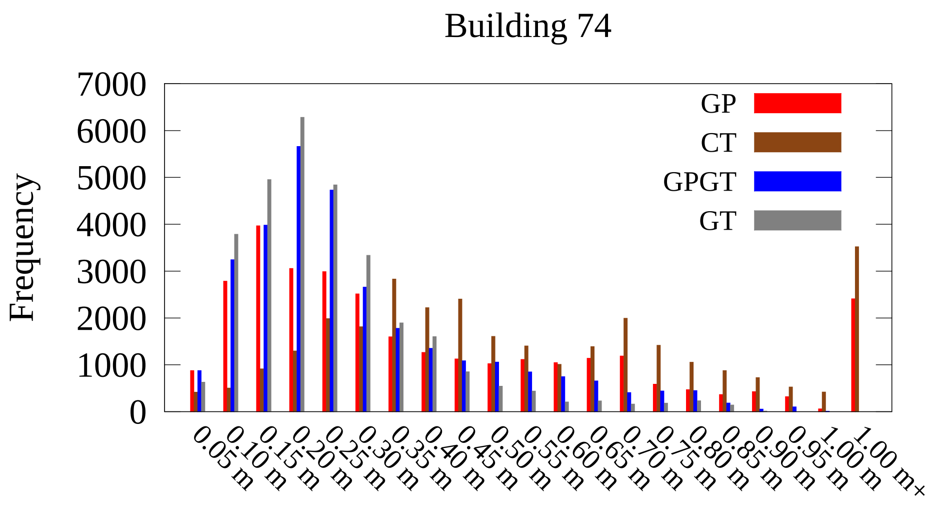


Figure 6.14.: A histogram for building 74. We map every point from the floorplan onto the gridmap and compare it to the ground truth. Afterward, we count the number of points which fall into given intervals.

6.3.1. Building 74

In the previous section we have seen that the GT and the GPGT performs better than the other approaches. We can see this behavior in Figure 6.14 with the gray and blue bars. The main peak of both approaches is around 0.2 m. This corresponds to the mean error which was slightly above 0.24 m. If we look at larger intervals the number of points decreases and no point is further away than 1.0 m. On the other hand, we have the approach where we combine multiple transformations to map a point from one map to the other. Figure 6.14 represents it with brown bars. The CT approach has multiple peaks at 0.35 m and 0.7 m. It also has points which are further away than 1.0 m. The same holds for the GP in red. Due to the bad performance of the interpolation of points which are further away, it has a lot of points in the interval 1.0 m and more. On the other hand, it has a peak at 0.15 m which represents the points in close range around our reference pairs.

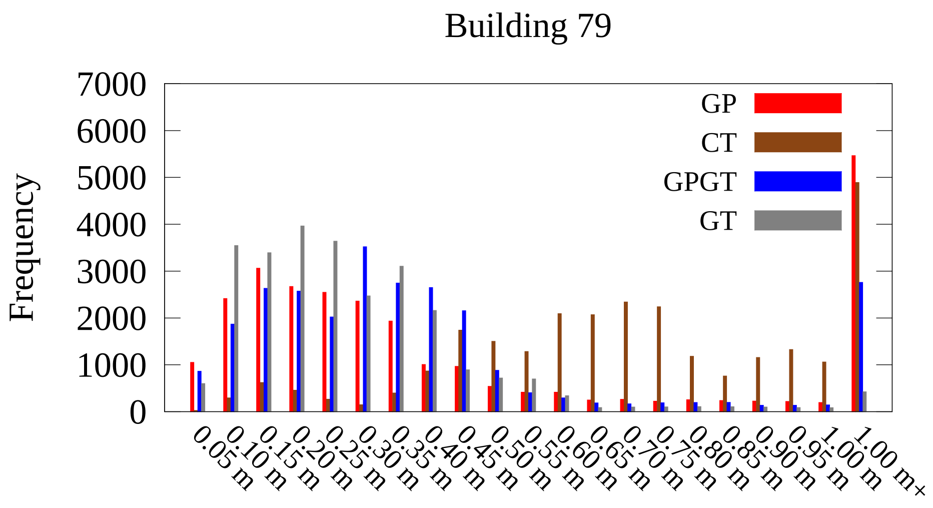


Figure 6.15.: A histogram for building 79. We compare every point transformed to the gridmap with the ground truth and look how much points lie in which distance to the ground truth.

6.3.2. Building 79

One more time the GT approach performed better than the other approaches. We can see this in Figure 6.15. Most points fall into the intervals between 0.1 m and 0.4 m. On the other hand, the GP and the GPGT have the most points in the same region. The GPGT has the peak at 0.3 m and the most points lie between 0.1 m and 0.45 m. The CT approach has the most points in the region from 0.45 m to over 1.0 m. However, all approaches have points which are further away than 1.0 m. Here, the GP has the highest peak followed by the CT and the GPGT. The GT has a rather small peak.

6.3.3. Building 106

On this data set all approaches, except the CT, deliver similar results. Figure 6.16 illustrates the results. The leading approach is the GT. It has its peak at 0.1 m. The second best approach is the GP with the highest peak at 0.2 m. Close to the GP is the GPGT. It has its main peak at 0.1 m, too. Most points

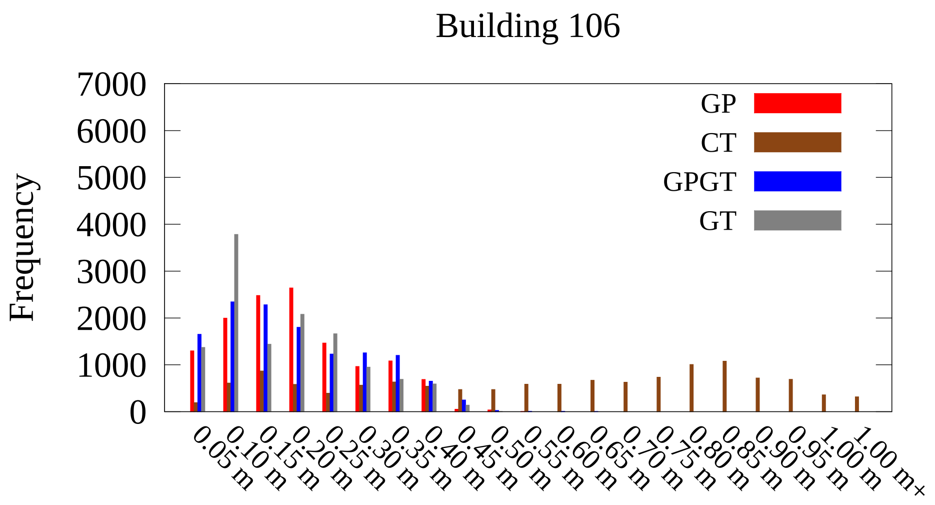


Figure 6.16.: All approaches, except the combined transformations, perform similarly where the main peak is between 0.1 m and 0.2 m.

of all three approaches lie between 0.05 m and 0.45 m. The CT method has points which are far away from the ground truth. The error range is from 0.05 m up to over 1.0 m where all points are equally distributed.

6.3.4. Factory Floor

We were able to generate good reference pairs on the factory floor data set. Hence, the main peaks of all approaches are at 0.15 m. Figure 6.17 shows the results. Most points end up with an error between 0.05 m and 0.40 m. However, the GPGT has another peak at 1.0 m+. This is due to one position where it does not align the floorplan very well to the gridmap. The GT has the highest peak while the GP has a smaller peak, in comparison to the GT, but is more spread in the region around 0.15 m.
In the next section we see how big the offset of a transformed point is from the ground truth.

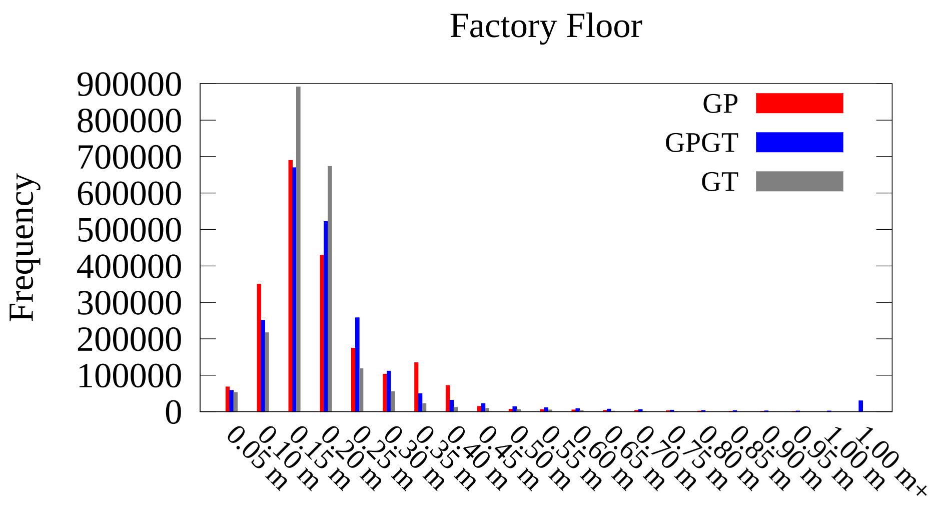


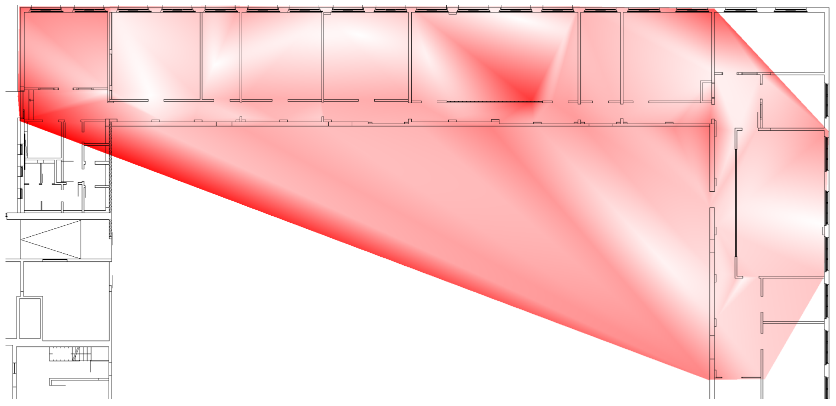
Figure 6.17.: The error distribution of the factory floor data set. All approaches perform similar. However, the GPGT has another small peak at 1.0 m+. The main peak of all approaches is at 0.15 m.

6.4. Error Map

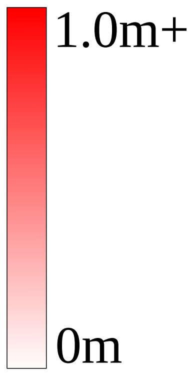
So far we have seen the comparison to the ground truth by mapping the structure from the floorplan onto the gridmap and the error distribution with histograms. This time we map the whole area which is enclosed by the ground truth to the gridmap. The result is a red shaded image where red represents a bad transformation and white a transformation which is rather close to the ground truth.

6.4.1. Building 74

The main challenge in the gridmap of building 74 is the room in the top left corner. We can see this in Figure 6.18. It is slightly rotated. Therefore, the GT approach cannot deal with this local variation and the area around this room is closer to red. Figure A.4 in the appendix shows the transformed map of the CT approach. This approach has still problems with the room in the



(a) Error Map of Building 74



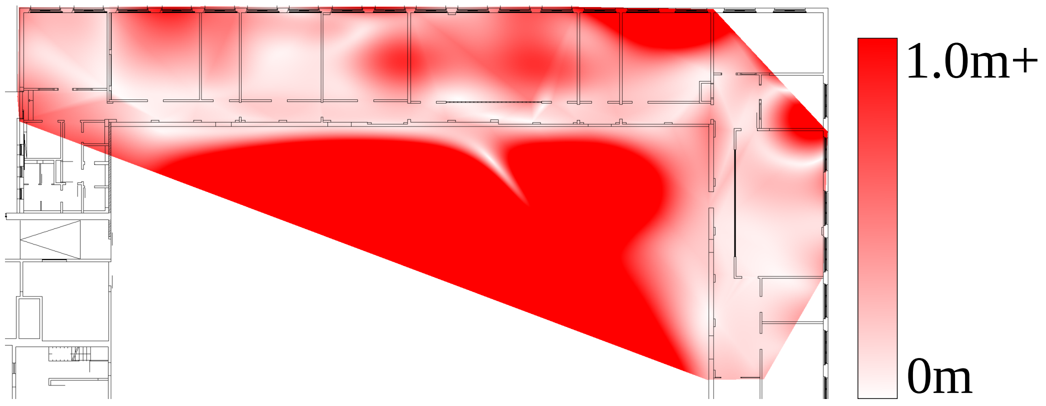
(b) Scale

Figure 6.18.: The GT approach has minor issues with local variations like in the upper left corner. The scale describes the distance between a point and its ground truth correspondence.

top left corner and behaves different at the outer border of the ground truth. Overall the shade of red is darker than with the GT. The next approach is the GP in Figure 6.19. As long as reference points are in close proximity we have a good match to the ground truth. However, there are also big areas of red where the GP cannot fit to the ground truth. This issue occurs because of the poor interpolation capabilities of Gaussian processes. On the other hand, such areas are not of high interest because we do not have any information about it on the gridmap and, therefore, do not want to drive there. The last Figure A.5 in the appendix is about the GPGT. In areas, where the Gaussian process would have to interpolate the points, because no reference pairs are in close proximity, the algorithm falls back to the global transformation. Hence, the result looks similar to the GT. However, it is able to deal with local variations.

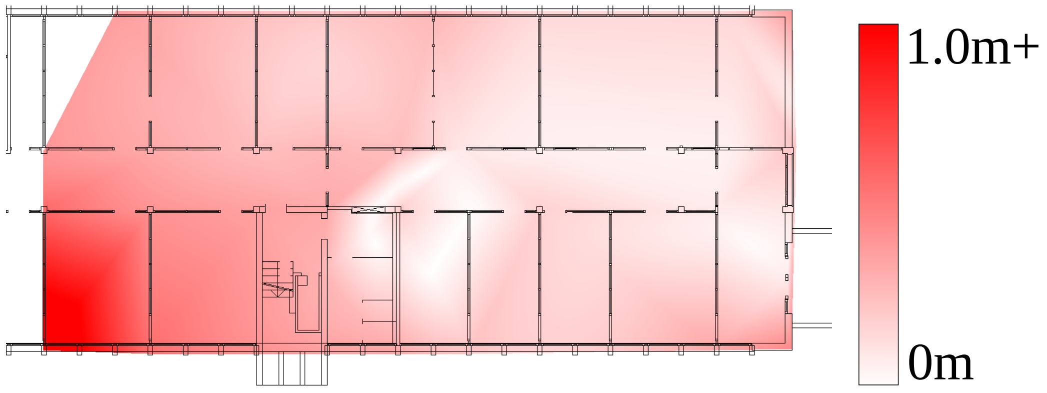
6.4.2. Building 79

The data set of building 79 consists of a robot tour where the robot drives mainly to the right. Hence, there are not enough reference pairs on the left side to represent it and the algorithm cannot take it into account. The



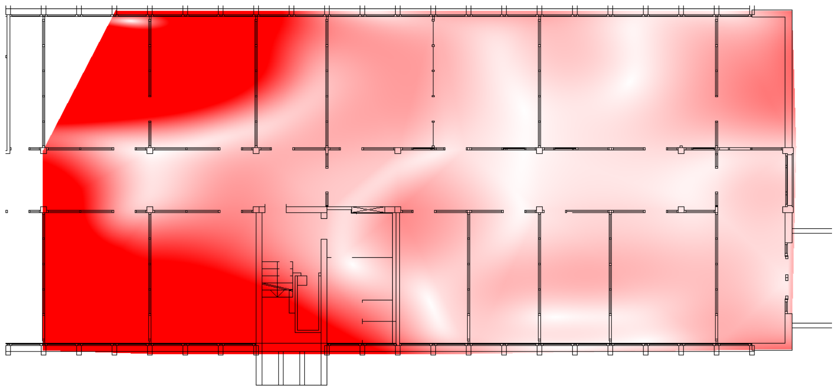
(a) Error Map of Building 74 *(b)* Scale

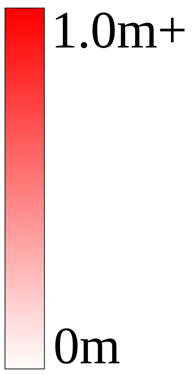
Figure 6.19.: The GP has problems with the interpolation between reference pairs. However, it can deal with local variations.



(a) Error Map of Building 79 *(b)* Scale

Figure 6.20.: The GT approach performs well in on this data set. Just on the left where no reference points are available is the shade red.


(a) Error Map of Building 79



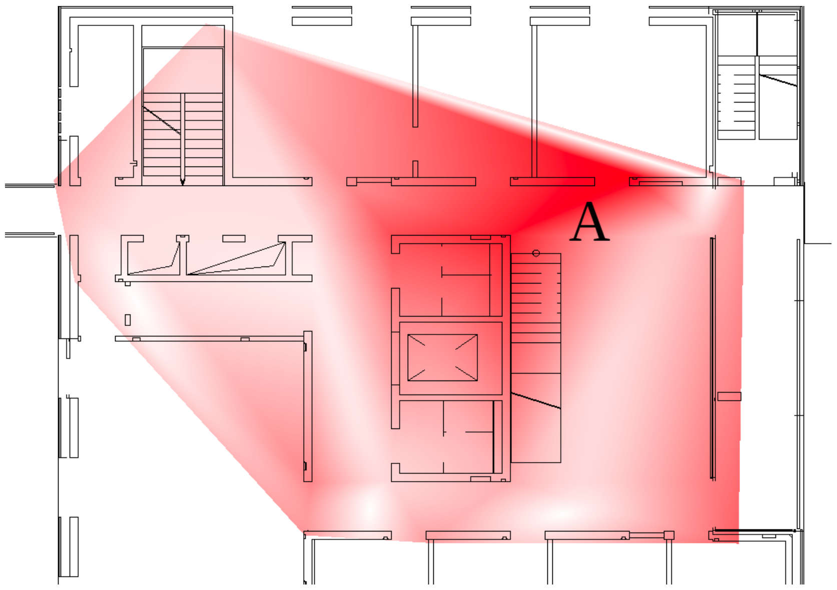
(b) Scale

Figure 6.21.: The GP has problems with the interpolation between reference pairs. However, it can deal with local variations.

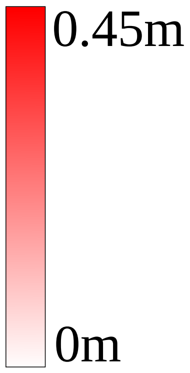
result is that the GT fits on the right side but is less accurate on the left side. However, it still performs better than any other presented approach. On the other hand, the approach where we combine multiple transformations into a single one (CT) performs in the opposite way: It fits on the left side to the ground truth but not on the right side. The next approach is the GP. Due to the lack of interpolation capabilities the performance on the left side is very bad and yields the highest approximation error. However, between reference pairs, like on the floor and some rooms, it performs close to the ground truth. On the other hand, we have the GPGT approach. Similar to the GP it has in some areas problems to interpolate the points well, like in the lower left corner. This is due to the bad performance of the GT in this corner which is the fall back strategy for the GPGT.

6.4.3. Building 106

The first Figure 6.22 illustrates a smooth result of the GT approach. However, the mapping is not well aligned at the starting area (position A) of the robot trajectory. This is due to the fact that the Monte Carlo localization needs a few iterations before the mean position among the particles is at the correct position. We see the same behavior of the GP in Figure 6.23. At the start location we have minor issues regarding a good alignment. However, the

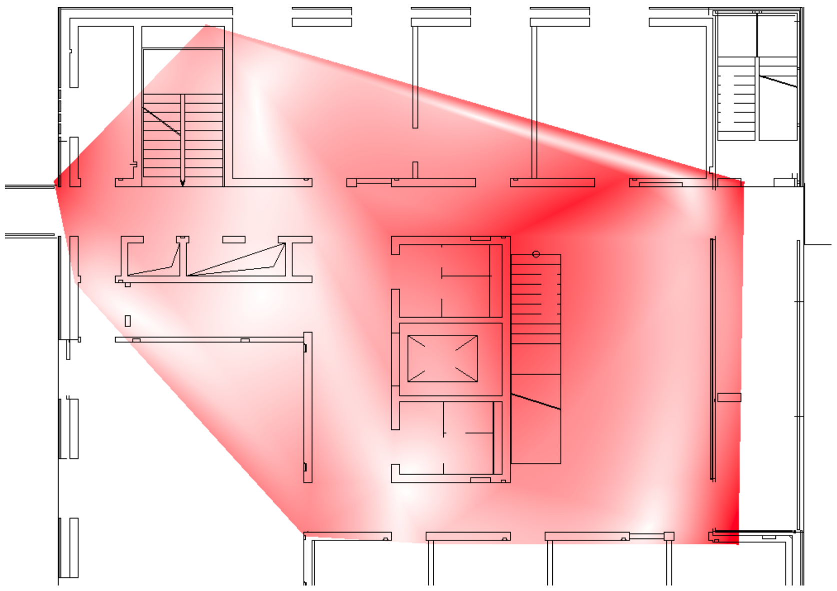


(a) Error Map of Building 106

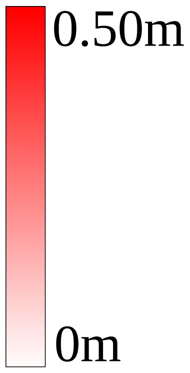


(b) Scale

Figure 6.22.: The red areas are around the start point of the robot. At this point the robot was not perfectly localized.

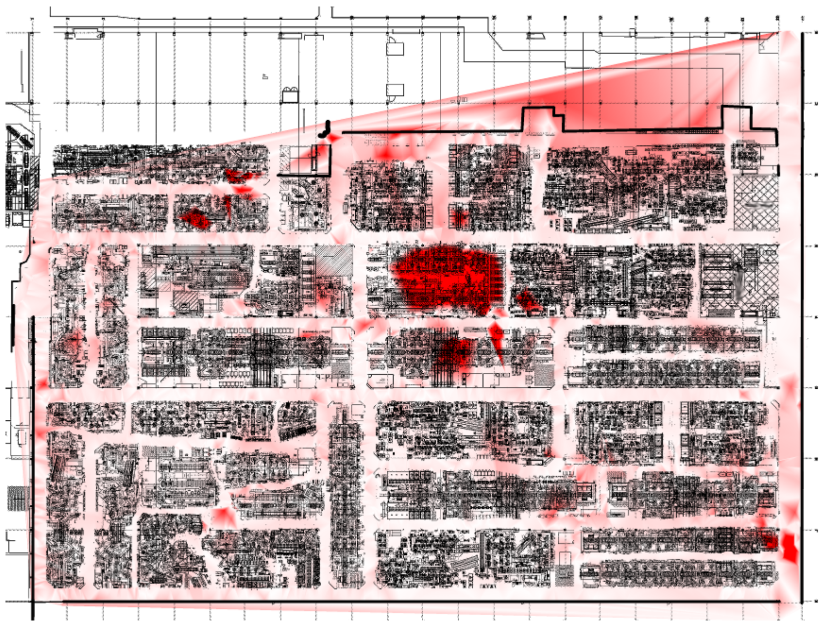


(a) Error Map of Building 106

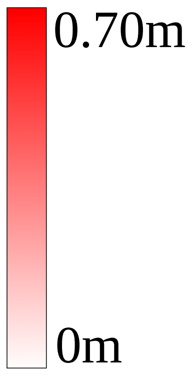


(b) Scale

Figure 6.23.: The GP has minor issues at the robot start position where the biggest red shaded area is located.



(a) Error Map of the Factory Floor



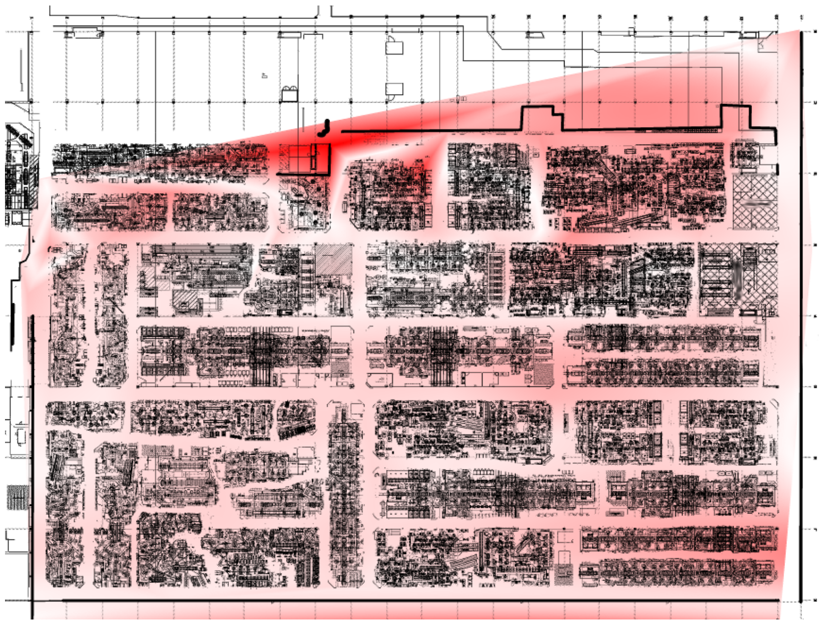
(b) Scale

Figure 6.24.: The GT error ranges from 0.0 m up to 0.7 m. Nevertheless, only small areas have a measurable error.

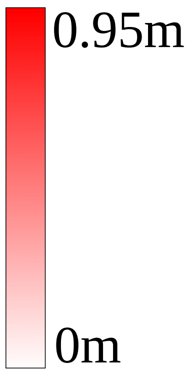
other areas fit to the ground truth. Figure A.8 in the appendix illustrates a noisier example of the GPGT. In the first section of the experiment chapter we have seen the wavy structure of the GPGT. This resolves into a grid pattern. Nonetheless, the overall alignment is close to the ground truth.

6.4.4. Factory Floor

Figure 6.24 shows the first result with the GT on the factory floor data set. A few small areas have an error of 0.7 m. Most other points behave similar to the ground truth. The GP produces a smoother result as can be seen in Figure 6.25. It contains more red shaded areas than the GT approach. On the other hand, the GPGT has a similar structure than the GT. It has red shaded areas on similar spots. However, the global performance is worse because of points which are further away than 1.0 m with respect to the ground truth. In Figure A.9 in the appendix we see the result of the GPGT.
The final section will be about the stability of a back and forth mapped point.



(a) Error Map of the Factory Floor



(b) Scale

Figure 6.25.: The GP has a larger error range. It starts at 0.0 m and ends at 0.95 m.

6.5. Stability of Back and Forth Mapping

In the last sections we have seen several experiments where we compare one transformation from one map to the other. Finally, we want to examine the performance of our presented approaches if we map a point from the floorplan onto the gridmap, and the inverse direction, namely taking the mapped point and transform it back to the floorplan. Hence, we map a point back and forth 100 times and evaluate the distance between the final and start point. Table 6.3 illustrates the closest and farthest distance of all approach.

6.5.1. Building 74

The GT approach performs one more time as the best approach. In Figure 6.26 we see how much the blue points spread with the back and forth mapping. Each point sticks close to the start point. The offsets range from

Approach	Building 79	Building 74	Building 106	Factory Floor
GT	0.22 m to 1.02 m	0.02 m to 1.10 m	0.71 m to 1.98 m	0.12 m to 0.92 m
CT	4.00 m to 50.00 m	1.32 m to 13.32 m	-	-
GP	0.77 m to 4.15 m	0.99 m to 3.16 m	1.23 m to 2.36 m	0.17 m to 18.93 m
GPGT	0.15 m to 4.71 m	0.07 m to 2.71 m	0.86 m to 7.59 m	0.90 m to 8.33 m

Table 6.3.: The closest and farthest point of the back and forth mapping experiment.

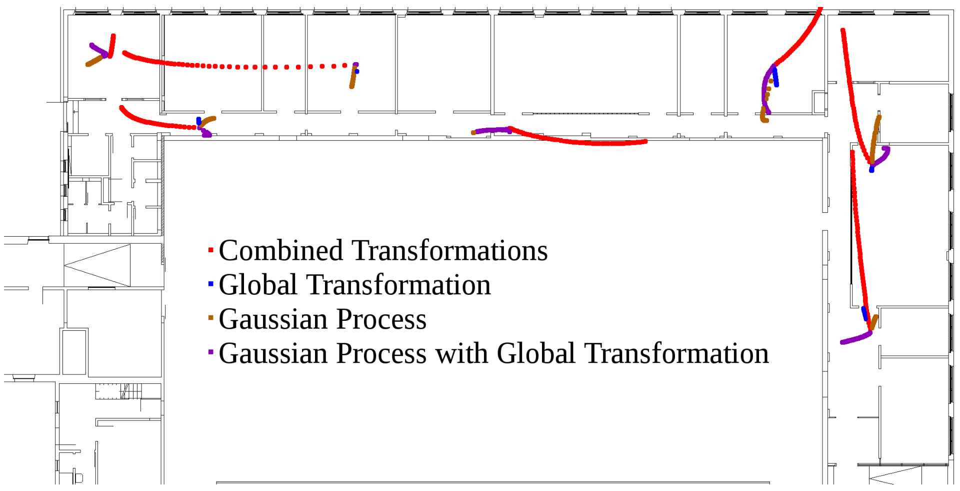


Figure 6.26.: The performance of the different approaches by mapping a point from the floorplan onto the gridmap and back several times. The red points are the CT, the blue points are the GT, the brown points are the GP, and the purple points are the GPGT.

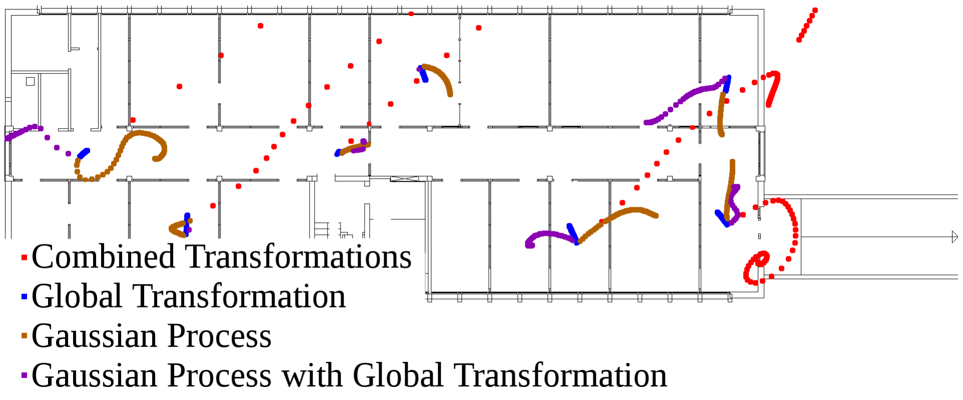


Figure 6.27.: The results from the back and forth mapping of points. The blue points represent the GP. The brown points are from the GP, whereas the purple points illustrate the GPGT. The red points show the result of the CT approach.

0.02 m to 1.10 m. The mean error is around 0.40 m. The worst approach is the CT which are the red points. Here, the errors range from 1.32 m up to 13.32 m. The mean error is around 7.00 m. The brown points represent the GP approach. Any point in close range to reference points end up in close proximity of the start point. However, if the point moves away from the given reference points it starts to shift. The smallest error is 0.99 m and the highest error is 3.16 m. The mean error is 2.00 m. The last approach GPGT, which the purple points represent, is slightly better than the GP. Here, the error ranges from 0.07 m to 2.71 m with a mean error of 1.20 m.

6.5.2. Building 79

The results of building 79 in Figure 6.27 are close to building 74. The GT approach, which is represented by the blue points, has a mean error of 0.54 m which is a bit higher than in building 74. The smallest error is 0.22 m whereas the largest error is 1.02 m. On the other hand, we have the GP which has a mean error of 2.14 m (brown points). The error ranges from 0.77 m up to 4.15 m. Better results achieves the GPGT. The purple points represent the behavior. At some points the transformation is stable. Hence,

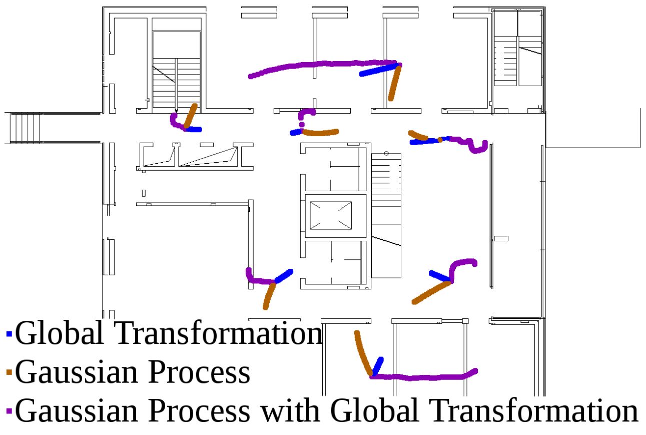


Figure 6.28.: The back and forth mapping results of building 106.

after several transformations the point is still in close proximity of the start. However, there are also points which are far away from the start area. Here, the smallest error is 0.15 m whereas the highest error is 4.71 m. The mean error is 1.71 m. The CT approach, which is illustrated by the red points, produces higher errors. The error ranges from 4.00 m up to over 50.00 m. Hence, the results are not very reliable.

6.5.3. Building 106

Figure 6.28 illustrates the result of the back and forth mapping on the building 106 map. The GT approach (blue points), GP (brown points), and GPGT (purple points) produce similar errors. However, they behave different. The GT has an error range from 0.71 m to 1.98 m. The errors of the GP are slightly higher. They start at 1.23 m and end at 2.36 m. The highest error has the GPGT. Here, one point is 7.9 m away from the start point. However, it still has well performing points. The best one has an error of 0.86 m.

6.5.4. Factory Floor

The results of the approaches on the factory floor data are very different. Figure 6.29 illustrates the trajectories of the points. The GT approach (blue points) did not move a lot. The error ranges from 0.12 m to 0.928 m. On the

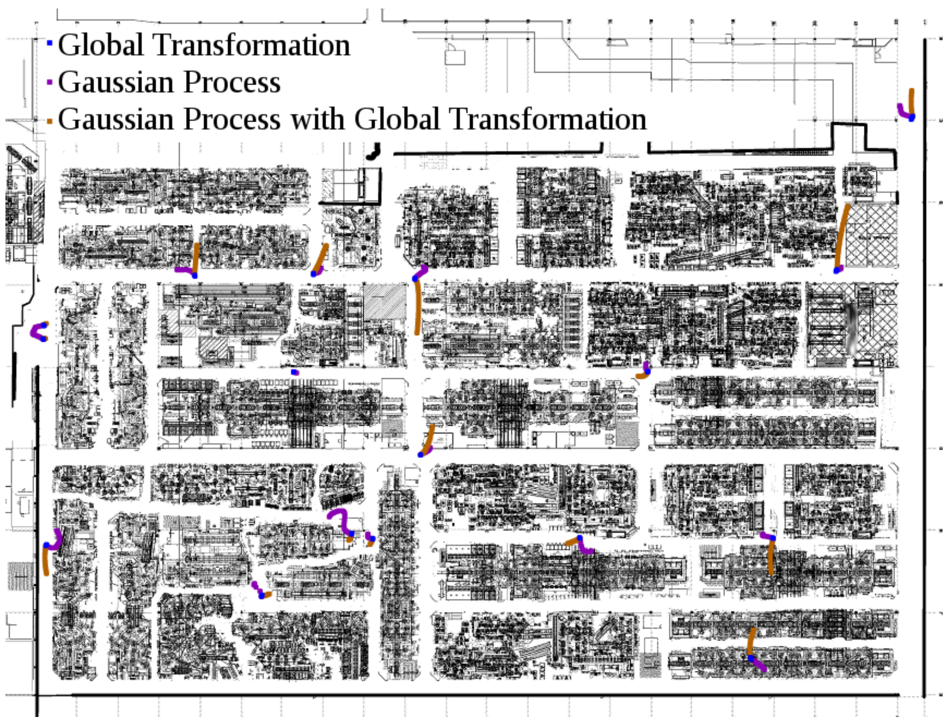


Figure 6.29.: The results on the warehouse with the back and forth mapping experiment.

other hand, the GP (brown points) performed worse. Its error ranges from 0.172 m to 18.93 m. Therefore, it has some areas in which a point does not move a lot but also areas with high movement. The GPGT is between the GT and the GP. It has an error ranging from 0.90 m up to 8.33 m.
We can see that the GT approach offers the best performance in this experiment. However, we could improve the performance to a zero error by taking the inverse global transformation. But to have a better comparison to the other approaches, where we always have to calculate the transformation in both directions, we neglect this possibility. The GP and GPGT approach perform similar. On the other hand, the CT approach is not very reliable in this experiment.

7 Conclusion

It is more intuitive for an operator to instruct and monitor a robot with the help of a floorplan rather than with a gridmap. Therefore, we proposed several approaches to map a point from a floorplan to its respective position on the gridmap and vice versa. First, we automatically generated a set of reference pairs on the floorplan and gridmap. These points served as anchor points on which our several approaches built the mapping. The first one computed a global transformation. It is robust on any floorplan-gridmap combination where the gridmap is well aligned. However, if the gridmap possesses local variations then the three other approaches perform better. One of it is a weighted combination of multiple transformations. The idea is that every local variation has its own transformation and, depending on the distance between a point and the local variation, it gets a higher or lower weight. The other two approaches base on a Gaussian process. One of it is a Gaussian process without preprocessing the data. Hence, it gets a set of reference pairs as training points and, afterward, computes the most likely corresponding point. The last approach uses also a Gaussian process but introduces an additional precomputation step. Due to problems with points, which are not in close proximity of neighbors, we first apply a global transformation to every point and then calculate the difference of the transformed point to the actual reference point. We train the Gaussian process with these difference vectors. At the end, we first transform a point with the global transformation and add the most likely difference vector from the Gaussian process to it.

In the experiments we evaluated our four approaches on four different data sets. Three of the data sets are office environments and one of it is a factory floor. In the first experiment we compared the performance of the different approaches to a ground truth map and analyzed the error in multiple ways. In another experiment we looked into the stability if we map a point from the floorplan to the gridmap and backwards multiple times. In the evaluation we have seen that the global transformations outperforms the other approaches on every data set where the gridmap is not bent or skewed. However, if we run the approaches on bent gridmaps, the Gaussian process outperforms the other three. On this data set the Gaussian process approaches outperform the other two. Therefore, the performance depends on the kind of gridmap. But, in any case, the mapping is close enough to the correct point and an operator is able to use a floorplan, which is more intuitive, to navigate a robot through the environment and monitor it.
Future work could include the refinement of the reference pairs. The better these points are the better the resulting transformations will be. Especially the Gaussian processes would improve, as we have seen in one experiment where we used the manually set reference points.

A
Appendix

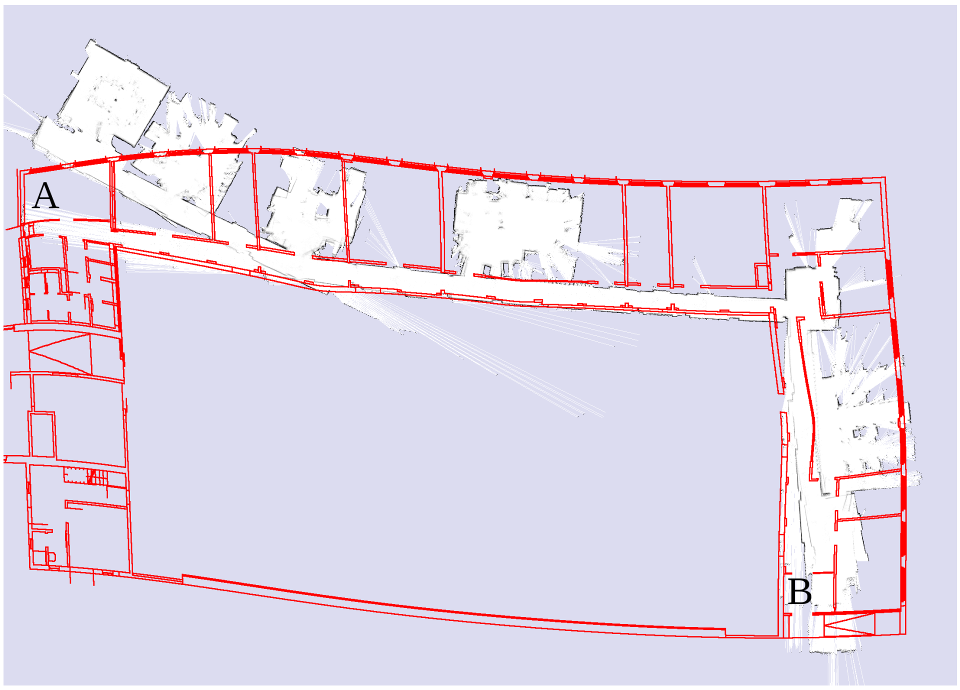


Figure A.1.: The CT approach cannot deal with outliers.

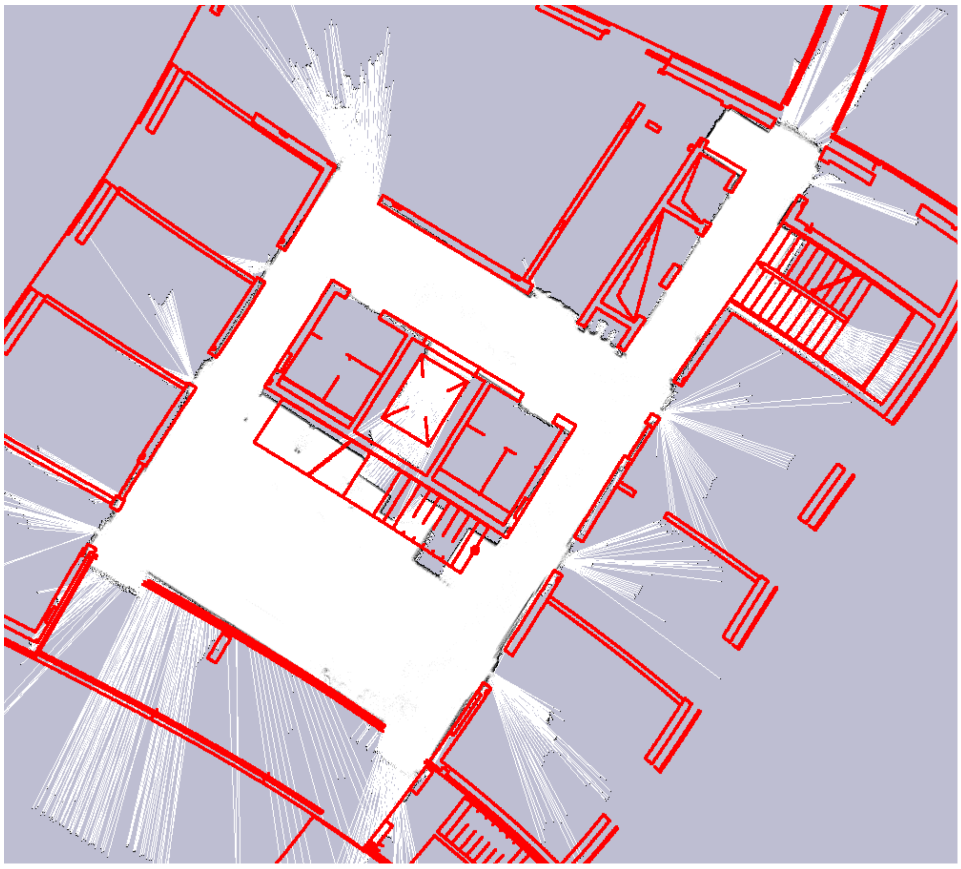


Figure A.2.: The GP sticks very close to the structure of the building. However, in border regions it shifts slowly away.

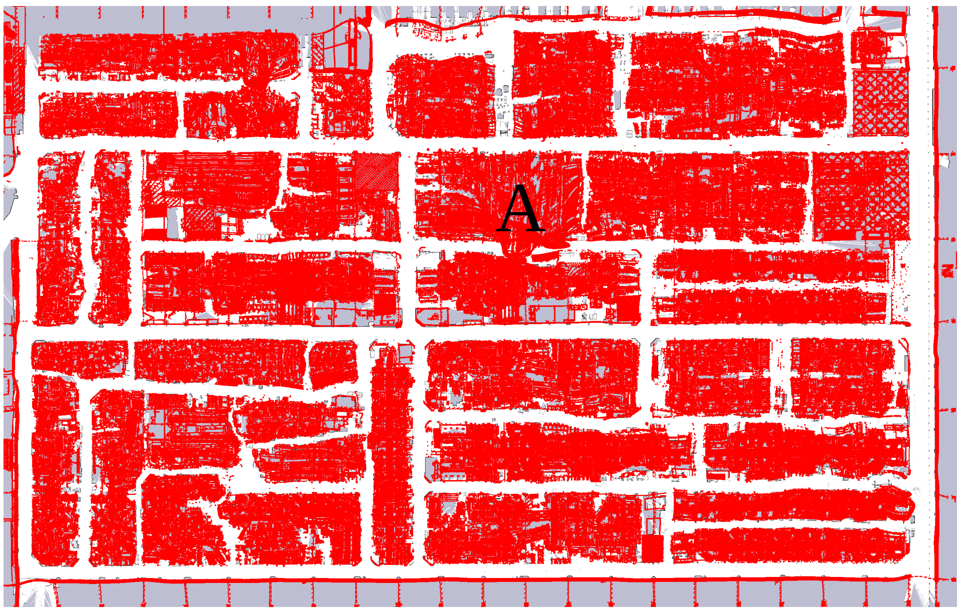
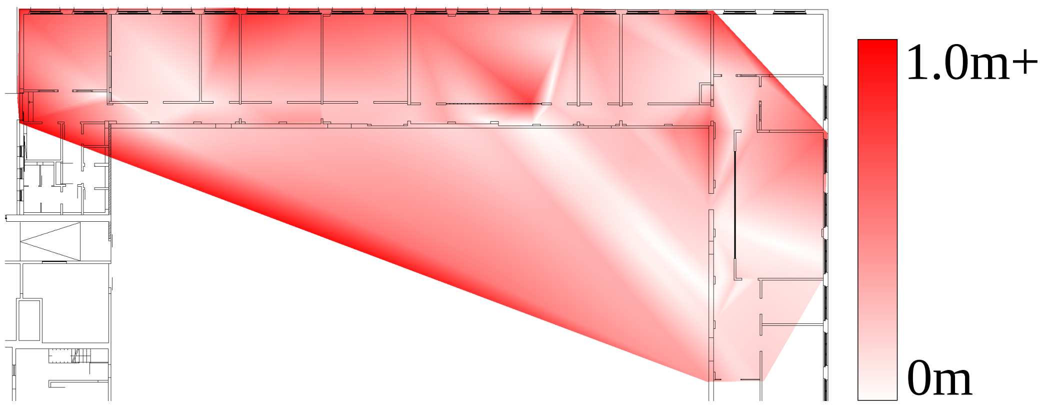
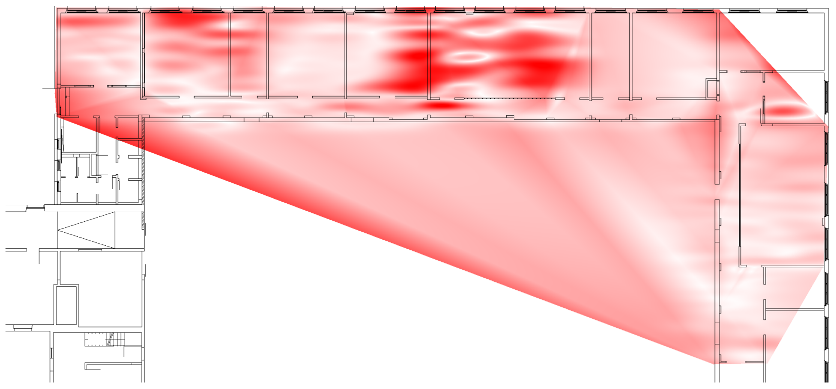


Figure A.3.: The GPGT aligns the floorplan to the gridmap in all positions, except one (Position A), very well.

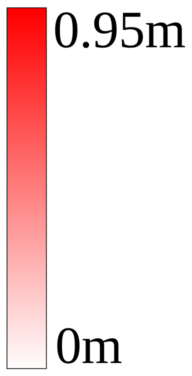


(a) Error Map of Building 74 *(b)* Scale

Figure A.4.: The CT approach suffers from outliers.

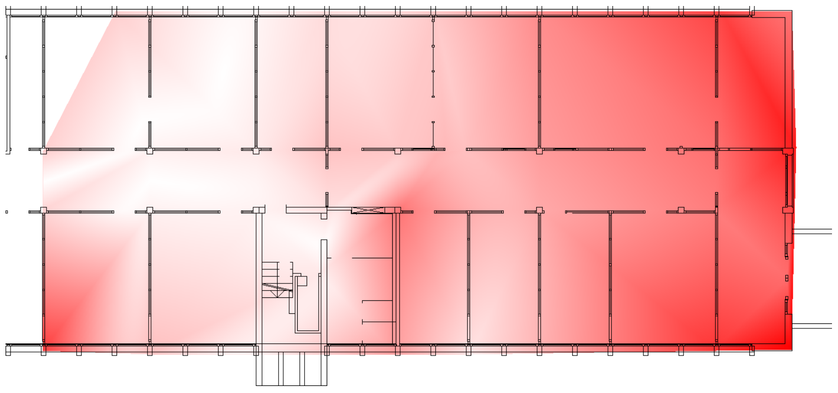


(a) Error Map of Building 74

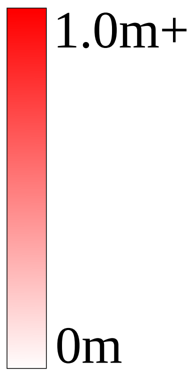


(b) Scale

Figure A.5.: The GPGT performs in areas where the algorithm has to interpolate the points like the GT approach.

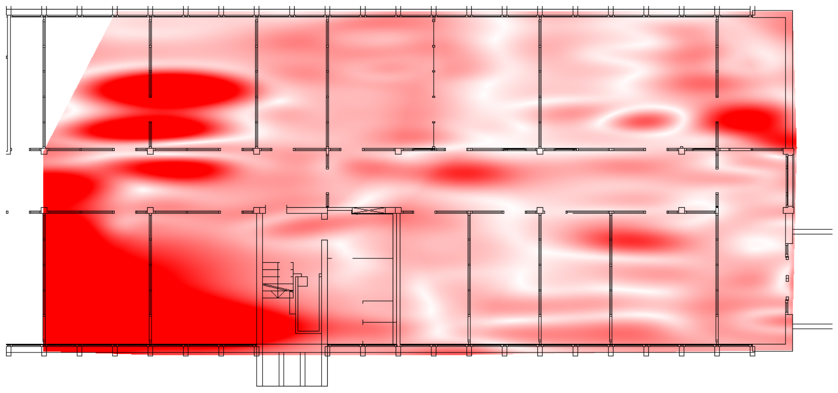


(a) Error Map of Building 79

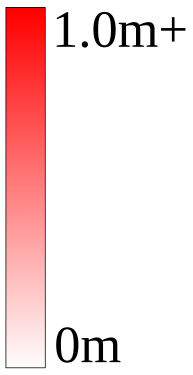


(b) Scale

Figure A.6.: The CT results on this data set in a shift. Hence, it does not represent the reference pairs on the right side very well.

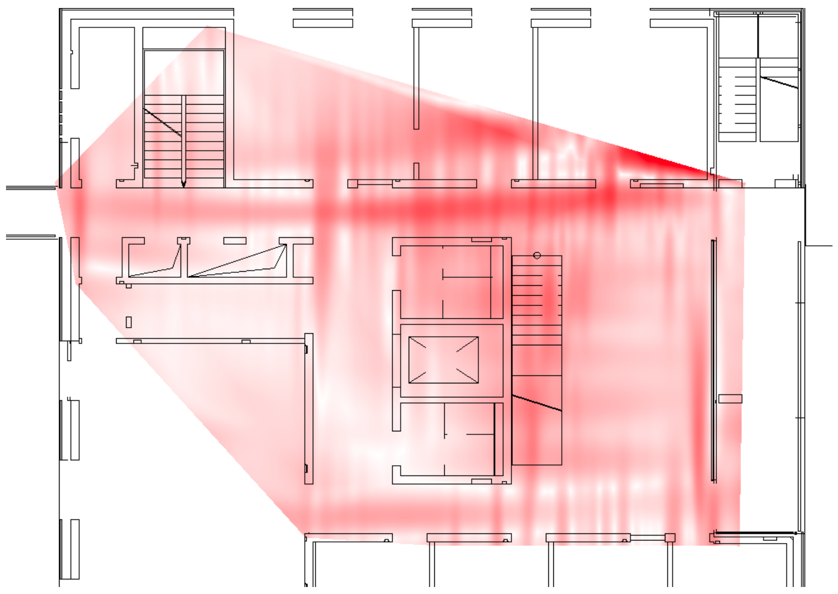


(a) Error Map of Building 79

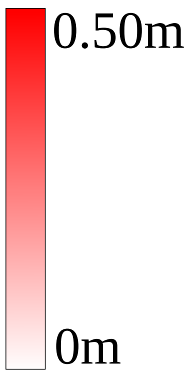


(b) Scale

Figure A.7.: As long as the GP has reference pairs in close proximity the result is reasonable. However, it cannot deal with the interpolation which is far away from reference pairs.

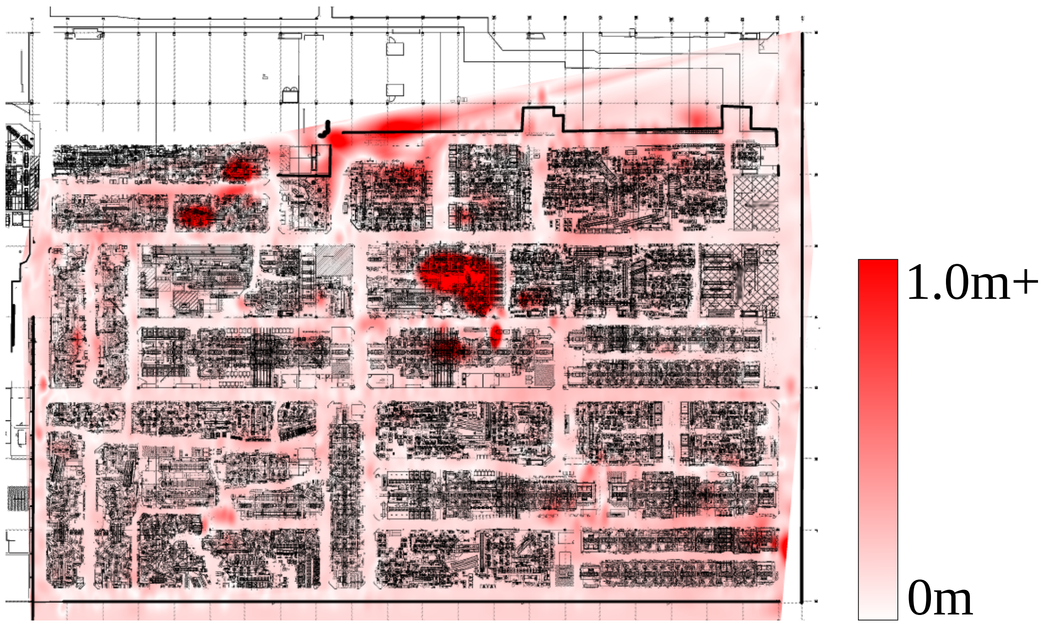


(a) Error Map of Building 79



(b) Scale

Figure A.8.: The wavy structure of the GPGT can be seen as a grid formation which iterates between white areas and red shaded areas.



(a) Error Map of the Warehouse *(b)* Scale

Figure A.9.: The GPGT behaves similar to the GT but has an overall higher error.

Bibliography

[1] Franz Aurenhammer and Rolf Klein. Voronoi diagrams. *Handbook of computational geometry*, 5:201–290, 2000.

[2] Patric Beinschob and Christoph Reinke. Advances in 3d data acquisition, mapping and localization in modern large-scale warehouses. In *Intelligent Computer Communication and Processing (ICCP), 2014 IEEE International Conference on*, pages 265–271. IEEE, 2014.

[3] Andreas Birk and Stefano Carpin. Merging occupancy grid maps from multiple robots. *Proceedings of the IEEE*, 94(7):1384–1397, 2006.

[4] Boris Delaunay. Sur la sphère vide. a la memoire de georges voronoi. pages 793–800, 1934.

[5] Arnaud Doucet, SJ Godsill, and C Andrieu. *On Sequential Simulation-based Methods for Bayesian Filtering*. Department of Engineering, University of Cambridge UK, 1998.

[6] Sean Philip Engelson. *Passive Map Learning and Visual Place Recognition*. PhD thesis, Yale University, 2000.

[7] Gerald Farin and Diane Hansford. Lineare algebra: Ein geometrischer zugang, 2003.

[8] Martin A Fischler and Robert C Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and

automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[9] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, 1999:343–349, 1999.

[10] Arthur Gelb. *Applied Optimal Estimation*. MIT press, 1974.

[11] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 14(5):403–420, 1970.

[12] JE Handschin. Monte carlo techniques for prediction and filtering of non-linear stochastic processes. *Automatica*, 6(4):555–563, 1970.

[13] Andrew Howard. Multi-robot simultaneous localization and mapping using particle filters. *The International Journal of Robotics Research*, 25 (12):1243–1256, 2006.

[14] Luca Iocchi, Stefano Pellegrini, and Gian Diego Tipaldi. Building multi-level planar maps integrating lrf, stereo vision and imu sensors. In *Safety, Security and Rescue Robotics, 2007. SSRR 2007. IEEE International Workshop on*, pages 1–6. IEEE, 2007.

[15] Michael Karg, Kai M Wurm, Cyrill Stachniss, Klaus Dietmayer, and Wolfram Burgard. Consistent mapping of multistory buildings by introducing global constraints to graph-based slam. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 5383–5388. IEEE, 2010.

[16] Jonathan Ko, Benjamin Stewart, Dieter Fox, Kurt Konolige, and Benson Limketkai. A practical, decision-theoretic approach to multi-robot mapping and exploration. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 4, pages 3232–3238. IEEE, 2003.

[17] Rainer Kümmerle, Bastian Steder, Christian Dornhege, Alexander Kleiner, Giorgio Grisetti, and Wolfram Burgard. Large scale graph-based slam using aerial images as prior information. *Autonomous Robots*, 30(1):25–39, 2011.

[18] Charles L Lawson. Software for c1 surface interpolation. mathematical software iii (john r. rice, editor), pages 161–194, 1977.

[19] G oksel Dedeoglu and Gaurav S Sukhatme. Landmark-based matching algorithm for cooperative mapping by autonomous robots.

[20] Ali Gurcan Ozkil, Zhun Fan, Jizhong Xiao, Steen Dawids, Jens Klæstrup Kristensen, and Kim Hardam Christensen. Mapping of multi-floor buildings: A barometric approach. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 847–852. IEEE, 2011.

[21] Max Pfingsthorn, Bayu Slamet, and Arnoud Visser. A scalable hybrid multi-robot slam method for highly detailed maps. In *RoboCup 2007: Robot Soccer World Cup XI*, pages 457–464. Springer, 2008.

[22] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced Lectures on Machine Learning*, pages 63–71. Springer, 2004.

[23] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

[24] Martin Riedmiller and I. Rprop. Rprop - description and implementation details, 1994.

[25] Christof Rohrig and Sarah Spieker. Tracking of transport vehicles for warehouse management using a wireless sensor network. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3260–3265. IEEE, 2008.

[26] Donald B Rubin. An overview of multiple imputation. In *Proceedings of the survey research methods section of the American statistical association*, pages 79–84, 1988.

[27] Sebastian Thrun. A probabilistic on-line mapping algorithm for teams of mobile robots. *The International Journal of Robotics Research*, 20(5): 335–363, 2001.

[28] Sebastian Thrun and Yufeng Liu. Multi-robot slam with sparse extended information filers. In *Robotics Research*, pages 254–266. Springer, 2005.

[29] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[30] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2276–2282. IEEE, 2006.

[31] Yinghua Xue and Hongpeng Liu. Intelligent storage and retrieval systems based on rfid and vision in automated warehouse. *Journal of Networks*, 7(2):365–369, 2012.