DLR-IB-RM-OP-2018-17

Leveraging Incremental Localization Information for Adapting Feature-based Map Matching Methods

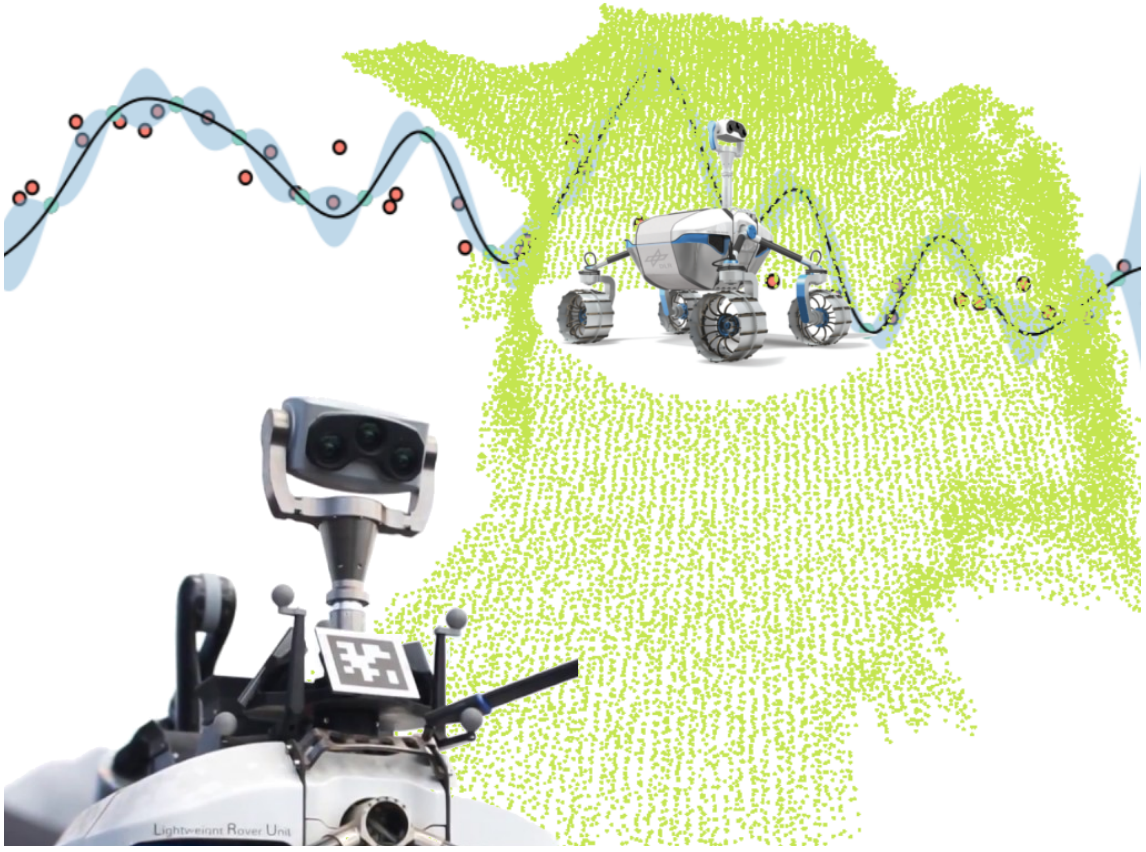
Masterarbeit

Matthias Holoch

Deutsches Zentrum
DLR für Luft- und Raumfahrt

Leveraging Incremental Localization Information for Adapting Feature-based Map Matching Methods

by
Matthias Holoch



Master's thesis
January 2018

FZI

Master's thesis, FZI
Department of Computer Science, 2018
Gutachter: Prof. Dr.–Ing. R. Dillmann, Prof. Dr.–Ing. T. Asfour

Abteilung Technisch Kognitive Assistenzsysteme
FZI Research Center for Information Technology

Karlsruhe Institute of Technology

Leveraging Incremental Localization Information for Adapting Feature-based Map Matching Methods

Master's thesis of

Matthias Holoch

Department of Computer Science
Institute for Anthropomatics
and
FZI Research Center for Information Technology

Reviewer:	Prof. Dr.–Ing. R. Dillmann
Second reviewer:	Prof. Dr.–Ing. T. Asfour
Advisors:	M.Sc. M. Große Besselmann
	M.Sc. M. Schuster
	M.Sc. M. Vayugundla

Research Period: 01. August 2017 – 31. January 2018

Affirmation

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, *Matthias Holoch*
January 2018

Abstract

The current state of the art of SLAM methods are sufficient to enable a specific robot to navigate a specific, static environment. However, the performance strongly depends on an expert to chose the right parameters. This thesis focuses on feature-based map matching, which is often used as a part of a SLAM solution to detect loop closures, thereby reducing the accumulated error of the robot's pose. In practice, the parameter-dependence constrains the robot's autonomy severely, since it can only safely navigate environments of which the rough structure is known beforehand. It also complicates using a shared map in heterogeneous multi-robot teams, because each robot system requires its own set of parameters to reliably navigate its environment. Finally, tuning the many parameters manually is tedious and often quite hard, even with expert knowledge about the system. The high dimensionality of the parameter space and long evaluation durations make tackling this problem with simple approaches, like a grid search, infeasible. To remedy this problem, a framework for automated parameter tuning is proposed that uses Bayesian optimization. It includes a measure for map matcher performance based on both the quality and the quantity of the matches, which only requires ground truth transformations between submap pairs. Labels that tell whether a submap pair should be matchable are not required. The proposed method aims to leverage transformations estimated by a robot's incremental localization system as pseudo ground truth, to calculate the measure during the optimization process. Since most autonomous robots use an incremental localization system, those transformations are available anyway. The proposed solution is evaluated on a simulation dataset and a dataset recorded with a real robot system. The experiments show that the proposed solution is capable of providing map matcher parameters with at least similar performance than a reference parameter set that has been hand tuned for the evaluated datasets.

Kurzfassung

Der momantane Stand der Forschung im Bereich SLAM ist ausreichend, um einem bestimmten Robotersystem die Navigation in einer bestimmten, statischen Umgebung zu ermöglichen. Allerdings ist diese Fähigkeit momentan davon abhängig, dass ein Experte die richtigen Parameter für das Robotersystem und die entsprechende Umgebung wählt. Diese Arbeit beschäftigt sich mit merkmalsbasierten Verfahren zur Zusammenfügung von Teilkarten, welche häufig in SLAM Systemen genutzt werden um Schleifen zu schließen. Die Abhängigkeit solcher Verfahren von den gewählten Parametern führt in der Praxis zu einer starken Einschränkung der Autonomie eines Robotersystems. Ein Roboter kann nur dann sicher in einer Umgebung navigieren, falls genug Informationen über ihre Struktur zur Verfügung stehen, um einen dazu passenden Parametersatz zu wählen. Außerdem erschwert es die Nutzung einer gemeinsamen Karte in heterogenen Roboterteams, da jedes Robotersystem potentiell unterschiedliche Parameter für eine optimale Kartierungsleistung benötigt. Zudem ist das manuelle Finden eines Parametersatzes oft schwierig, auch, wenn Expertenwissen über das System vorhanden ist. Durch die hohe Dimensionalität des Parameterraums und durch lange Evaluationszeiten sind einfache Ansätze, wie eine Rastersuche, nicht geeignet. Um dieser Problematik entgegen zu wirken, wird in dieser Arbeit ein Verfahren vorgestellt, welches mit Hilfe von Bayesscher Optimierung automatisch einen passenden Parametersatz findet. Dazu wird ein Gütemaß für die Kartierungsleistung vorgestellt, das die Qualität und Quantität der berechneten Transformationen bewertet und lediglich Ground Truth Transformationen von Teilkartenpaaren benötigt. Daten, die definieren ob ein Teilkartenpaar zusammengefügt werden können sollte sind nicht notwendig. Das entwickelte Verfahren nutzt durch Koppelnavigationsmethoden bereitgestellte Transformationen als pseudo Ground Truth für den Optimierungsvorgang. Da irgendeine Form von Koppelnavigation auf fast allen autonomen Robotersystemen eingesetzt wird, sind diese Informationen sowieso verfügbar. Das vorgestellte Verfahren wird auf einem Simulationsdatensatz und einem Datensatz, der von einem Robotersystem aufgenommen wurde, evaluiert. Dabei konnte gezeigt werden, dass das Verfahren allein mit Hilfe der Transformationen der Koppelnavigation automatisch Parameter findet, die ähnlich gut funktionieren wie ein von Hand gefundener Referenzparametersatz.

Contents

1. Introduction

Simultaneous Localization and Mapping (SLAM) is the problem of concurrently constructing a map of the environment, while using that same map to localize an agent. Autonomous robots need a map and a localization method for other navigation-related tasks like path planning and obstacle avoidance. SLAM methods are necessary for autonomous robots, which act in unknown or little-known environments, since they enable navigation in areas without a-priori information like a pre-constructed map. Such applications include planetary exploration, where a high level of autonomy increases the robot's effectivity because of communication delays, or search and rescue applications, where quick action in a recently altered environment is required and creating an updated map is part of the robot's task. Other robotic applications may also utilize SLAM, for example to avoid the great effort of manually building a map suitable for robot localization. SLAM solutions normally combine two different types of localization methods: First, incremental localization methods, such as wheel- or visual-odometry. Such methods only measure small pose increments (e.g. robot speed), relative to the robot's last pose. While this is very precise for small distances, it also implies that measurement errors will accumulate until the estimated pose becomes unsuitable for building a consistent, global map. The second type of localization methods use the environment's landmarks to estimate the robot's pose. Especially for semi- or unstructured environments, this generally yields less precise results. However, the error of those methods is mostly independent of previous pose estimates and it allows recognizing previously visited locations. Such a "loop-closure" can be used to retroactively reduce the accumulated error of the robot's estimated pose. One way to approach that type of localization method is map matching. The global map can be viewed as a graph of submaps, where nodes are submap origins and edges are the estimated transformation between them. While incremental localization methods are well suited for providing edges between subsequent submap pairs, a map matching pipeline can determine whether an arbitrary submap pair overlaps and, if so, estimate a transformation for it. This can be done for example by finding and associating specific geometric patterns in both submaps, such as parts of furniture in indoor environments, or stones, walls and trees in outdoor environments. Note that currently, most of those methods do not understand the semantic meaning of those patterns.

Previous research on map matching gave birth to algorithms that work great for a specific type of robot (i.e. sensor setup and perspective) and environment. They usually use a pipeline that consists of multiple, interdependent steps, extracting a set of descriptors for each submap. Those descriptors are high-dimensional vectors which encode information about a small area in a submap, such that the distance of two descriptors (in descriptor-space) is small if, and only if, the area which they encode is similar. Therefore, similar descriptors from different submaps are likely to be at the same position in the environment. This is used in the map matching pipeline, by comparing

the descriptors of the submap pair and finding corresponding (i.e. similar) points in both submaps. With enough correspondences, a transformation can be estimated for the submap pair, which places corresponding points close to each other.

However, current state of the art SLAM methods lack robustness, needed especially for long-term use in differently structured environments. A detailed survey of current problems, including the problem discussed here, and directions of research in SLAM is given in [11]. One of the reasons for those shortcomings is that the pipeline usually has a huge number of parameters. They greatly influence its performance and depend on the environment's structure, the robot's perspective and its sensors. An example for such a parameter is the descriptor search radius, which determines the size of the area that is encoded in each descriptor. Those parameters currently have to be tuned by hand for each environment type and robot system: Different environments contain different significant features and different robot systems may have different sensor modalities and another perspective on the environment. This process takes an expert a lot of time and requires at least a little bit of a-priori information about the environment (i.e. its rough structure).

The problem is even more severe for matching submaps of different robot systems, which would help to enable having a shared, global map on multi-robot missions with different types of robots. For example, a flying robot can quickly survey the mission environment and create a coarse map from above. Simultaneously, a wheeled robot can explore and map the most interesting areas in detail and interact with objects there. The two robot systems are mapping from different viewpoints and with different sensors, which yields maps with different levels of detail and resolutions. This means that, even if they use the same map matching pipeline, they will require different parameters to optimally work on their own map. In turn, potentially a third set of parameters is necessary to optimally match submap pairs from the two different maps.

In order to alleviate the need for manual parameter tuning, this work aims to leverage information from incremental localization methods to improve the performance of a given map matching pipeline. The proposed solution uses relatively well-known transformations between submap pairs to automatically estimate a set of parameters that works well for the specific environment and robot system. It includes a performance measure adapted for automatic optimization and a method for estimating a function from parameter-space to performance-space, using Gaussian Processes. Further details on the exact problem statement can be found in Section 1.1.

1.1. Problem Statement

The idea to use incremental localization information is straightforward: It is an information source that is readily available on most advanced robot systems, because all SLAM systems use some form of incremental localization. Also, self-motion information is vital for animals to learn visually guided behaviors [17], which suggests that it is a salient information source to learn from.

While a robot moves through an environment, it generates multiple submaps. The incremental localization's pose estimation will be used as an initial guess for the transformation between those submaps. While the precision of the robot's pose estimation will severely decrease without using map matching, the transformations between *subsequent* submaps are generally more pre-

cise than transformations given by map matching solutions. The ultimate goal is to quickly find a working set of map matching pipeline parameters, by only using those transformations. Optimally, the parameters should be found before any loop-closures are required by the robot. This way, the algorithmic pipeline could automatically adapt to a completely unknown robot system and environment while the robot is exploring that environment.

However, this imposes harsh restrictions on the used algorithms: By only using subsequent submap pairs, the training dataset would be very small and does not contain examples for submap pairs which need to be matched for loop closing. Computational power on an active robot system is often used up by other systems, so the algorithm would need to be very efficient. Also, the available time the algorithm has until a solution is required, depends on the path the robot takes and the quality of its incremental localization.

Therefore, this work will first focus on easier problems with less constraints. These preliminary steps will be useful by themselves and, additionally, yield information about the feasibility of the online solution:

1. **Offline, with ground truth:** There are no time constraints on finding a good parameter set and the calculations take place on one or multiple separate machines. All transformations between submaps are known, for example by using a dataset that contains the robot's ground truth trajectory. A solution for this step will not increase a robot's autonomy for completely unknown areas. However, it will free roboticists of manually tuning parameters for each new environment and robot system, as long as an adequate ground truth source is available during a test run. A sufficient ground truth can be generated, for example, with a tracking system or differential GPS.

2. **Offline:** No ground truth for the robot's trajectory is available. This means that the training dataset contains only submap pairs for which the transformations can be estimated with sufficient precision by on-board methods. Those submap pairs are usually only subsequent submaps, for which the robot's incremental localization system can estimate relatively precise transformations. Other means of getting some transformations of submap pairs could include inter-robot matches, e.g. via markers on the robot, if multiple robots are moving about in the environment. For solving this step, the module that learns the relationship between parameters and map matching performance needs to generalize from a severely restricted training dataset. A solution for this step will increase the robot's autonomy if it can afford to change into a optimization-state and wait until a set of parameters are found, after briefly exploring its environment. In any case, the benefits of the previous step remain, with the added bonus that no ground truth robot trajectory is required.

3. **Online:** The training dataset is the same like in the previous step, but the whole process of finding a set of parameters needs to run on the robot while all its other systems are running. This means the solution's efficiency is key: It needs to find parameters quickly, without using too much processing power. Also, the solution would need to be tightly integrated into the specific robot system, to make sure the optimization does not disturb other processes,

vital to the robot's safe performance. Since the severity of the time constraints strongly depends on the mission type, this last step is more gradual. Additionally, pretraining under the conditions from the previous steps could be used to greatly reduce the search space.

1.2. Contributions

The key contributions of this thesis are:

- A map matcher performance measure, based on the number of matches and their quality on a specific dataset. It only requires ground truth transformations between the submap pairs.
- The design of a system architecture to use the aforementioned performance measure and Bayesian optimization for automatically finding a good set of map matcher parameters.
- An evaluation of the proposed system on both a simulated and a real robot dataset.
  - It is shown that the system can find parameters, which perform better than the hand-tuned reference parameters, when subsequent submaps with ground truth transformations are used as training data (step 1 of the problem statement).
  - The experiments further show that the system can still find parameters, which perform similar to the hand-tuned reference parameters, even when the ground truth transformations are replaced by transformations from the incremental localization solution (step 2 of the problem statement).
- An open-source implementation of the system, which will be released on Github under the BSD 2-Clause license, to facilitate further research.

1.3. Outline

In Chapter 2, related research, which can be used to automatically optimize parts of a robot mapping system, is discussed. This also includes both optimizing different parts of the system, like automatic sensor calibration, and other ways to optimize feature-based map matching methods. Chapter 3 describes basic theoretical models in the areas of Bayesian optimization and feature-based map matching, upon which this thesis builds. An experienced reader may want to skip to Chapter 4, which describes the proposed solution to the problem statement as defined in Section 1.1. Chapter 5 contains information about the used reference system in Section 5.1 and evaluates the proposed method. The thesis concludes with Chapter 6, which discusses the usefulness of the proposed method and traces open questions for future research.

2. Related Work

This chapter is split into three parts and discusses different approaches on how to optimize parts of a robot's perception system automatically. The original goal of the thesis was to find some way to make the robot's perception system more robust, by using an information source that is already available. The chosen source was incremental localization information, since all autonomous robot systems use some kind of that localization method. Other information sources like assumptions about the environment's structure were exploited in previous publications, to automatically calibrate or optimize the parameters of a sensor system. Those methods are briefly explored in Section 2.2. Incremental localization information is well suited as an information source to optimize feature-based map matching. However, whether it is better to optimize the map matching pipeline's parameters as a black box or, for example, only a single element of the pipeline, was not clear at the beginning of the work on this thesis. Section 2.1 covers general parameter optimization techniques, which includes the type of method utilized by the proposed solution. Alternative approaches are covered in Section 2.3. It describes latest machine learning techniques for learning feature descriptors, which are an essential part of the map matching process. The section focuses on methods that could be adapted to learn feature descriptors from incremental localization information.

While this thesis could only explore one kind of the previously mentioned methods, they are in no way exclusive. On the contrary, a robot system without a solid sensor calibration will quickly fail. Likewise, while optimizing map matching parameters helps a lot, the map matching pipeline can only be as good as the methods it uses. Therefore, finding better feature descriptor methods is also an important task for improving the robustness of the robot's perception.

2.1. Parameter Optimization Techniques

There are many different approaches to this topic, with vastly different use-cases and varying advantages and disadvantages. This web resource [35] gives a good overview over different techniques with references to publications of the respective field. Generally, parameter optimization techniques try to find the extremal value of a possibly non-convex objective function

$$f : X^d \to \mathbb{R} \tag{2.1}$$

with an set of constraints C [21], by solving

$$\max_{\mathbf{x} \in C \subset X^d} f(x). \tag{2.2}$$

For this thesis' use-case, all $X_i, i \in \{1, \ldots, d\}$ are continuous ($X_i = \mathbb{R}$) and the constraints C restrict their values to a compact interval. Each X_i corresponds to a parameter of the map matching pipeline and its compact interval $C_i \in C$ is the range of its possible values. The fact that evaluating the objective function has extremely high time costs and that its mathematical form is unknown is another important trait of this use-case. Therefore, many techniques like naive approaches (e.g. uniform grid search), complete search strategies, evolution strategies, or methods that require more knowledge about the objective function were not further considered. However, there are still many different viable approaches remaining. For example publications about the algorithm configuration problem like [24] show promising results for configuring parameters SAT and MIP solver algorithms, as to minimize their runtime. While adapting such methods for this use-case is an interesting prospect, it would go beyond this thesis' scope. Instead, this thesis uses a relatively simple Bayesian optimization approach with Gaussian Processes, since it tends to yield satisfying results with relatively small configuration effort and, most importantly, copes very well with costly evaluations [8]. With recent research into robot introspection (e.g. the IROS 2017 workshop about Introspective Methods for Reliable Autonomy [1]), more publications related to this thesis' approach appeared. In [22], a method to optimize laser- and visual odometry systems was proposed, which also utilizes Bayesian optimization. Its intended use-case is very close the one of this thesis, except that it optimizes a different part of the robot's perception system. To do so, it utilizes the estimator's posterior distribution as described in [23] to get a training signal for the optimizer without requiring ground truth. An introduction to the Bayesian optimization methods used in this thesis can be found in Section 3.3.

2.2. Automated Sensor Calibration

The most ubiquitous automated parameter optimization method in robot perception is automated sensor calibration. When considering robot perception, the sensor calibration is one of the most basic elements. Since all other perception systems are based on the sensor's output, it is arguably the most important element to get right. In contrast to the problem of optimizing map matcher parameters, the way a specific sensor captures the information of the environment can be described analytically. By assuming a specific sensor model, those approaches only work with that specific type of sensor. Nevertheless, their core idea about getting an information source to use as a training signal can often be transferred to other sensor models.

In [29], the authors propose a method for calibrating a multi-beam LIDAR sensor in three steps. It requires an Inertial Measurement Unit (IMU) and an arbitrary, static environment to take measurements from while moving. The method is capable of estimating the sensor's pose relative to the IMU (extrinsic calibration), as well as the angles, distance-offsets and a remittance model of each individual beam (intrinsic calibration). For this, the assumption is used that points tend to lie on contiguous surfaces, rather than being distributed randomly. Another publication [33] proposes a method for rotating LIDAR sensors, which exploits redundancies within the recorded point cloud. For example for the KaRoLa scanner depicted in Figure 3.3, a full 360° rotation will record everything twice.

The classic way to calibrate cameras is to manually hold a checkerboard pattern in front of the sensor at different angles and distances. On a robot system with a manipulator, this can be automated by having a checkerboard-like marker on the manipulator and moving it in front of the camera, as described in [44].

As of December 2017, besides the publication [22] mentioned in Section 2.1, to the best of our knowledge, no publications exist which proposed methods to automatically optimize other parts of a robot perception system necessary for SLAM. Since there already are several published methods for sensor calibration, this thesis only focuses on optimizing the map matching pipeline.

2.3. Deep Convolutional Neural Net Features

This section contains related work on methods for optimizing a specific element of the map matching pipeline: The feature descriptor (see Section 3.2.5). Since Krizhevsky et al. won the ImageNet challenge in 2012 with their deep Convolutional Neural Network (CNN) [25], CNNs dominate the image recognition domain. However, their ability to recognize complex patterns is not limited to 2D data. The dominance of CNN based descriptors in the 2D image matching domain suggests that such methods could also be of great use in the domain of 3D map matching. Recent publications, like 3D Match [51] in the 3D reconstruction domain, show that CNNs can also handle 3D data very well. A learned descriptor could automatically adapt to describe salient geometry in different environments, if suitable training data is available. Or maybe a powerful descriptor, suited for many different environments, could be trained with a sufficiently diverse training dataset. Two things were of special interest while researching: From which sources and how the authors created training data, and the way they prepared the 3D data for the neural network.

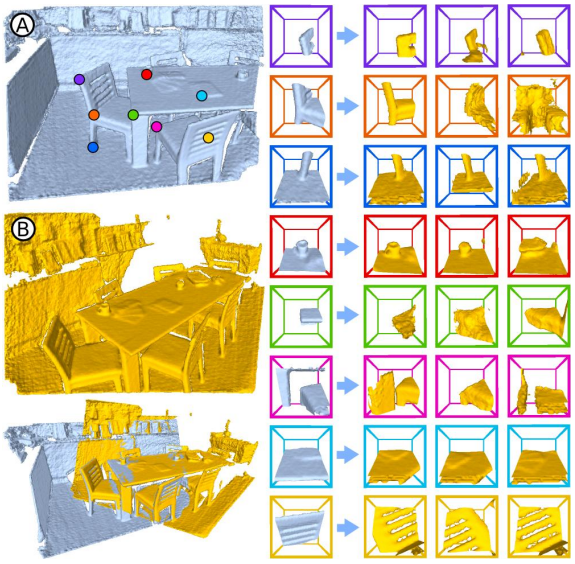


Figure 2.1.: The left column shows the two submaps A and B and the registration result at the bottom. On the right side, surface patches from the points marked in A are visualized, together with their nearest three neighbors in descriptor space from B. From [51]

3D Match [51] uses a CNN to learn a feature descriptor for 3D reconstruction (see Figure 2.1).

This problem is equivalent to 3D map matching. The network architecture is based on AlexNet [13]. It consists of eight convolutional layers and ends with a single pooling layer to compute a 512-dimensional feature descriptor. As input-data, a $30 \times 30 \times 30$ voxel grid of Truncated Distance Function (TDF) values is used. The choice of the voxel size parameter determines the size of the area described by the $30 \times 30 \times 30$ voxel grid. The TDF values are clipped to 1 and then flipped, so that they range from 0 (far away from a surface) to 1 (on surface). This form is compatible with multiple common 3D data representations, including point clouds, depth maps (both used in 3D map matching) and 3D meshes (often used in object recognition). Training data for their approach consists of surface point pairs from different submaps and a label to tell whether they correspond to the same physical point. Network training is done in a Siamese fashion, with two identical streams, sharing their weights. Each stream calculates the feature descriptor of a surface point. A loss function is used, so that point pairs that correspond to the same physical point minimize their distance in the resulting descriptor-space, and point pairs which do not increase their distance. To generate a training dataset, the authors extracted correspondences from existing high-quality RGB-D reconstructions. On a robot system with incremental localization, transformations between submaps with low uncertainty, i.e. subsequent submaps, could be used to generate training correspondences, instead.

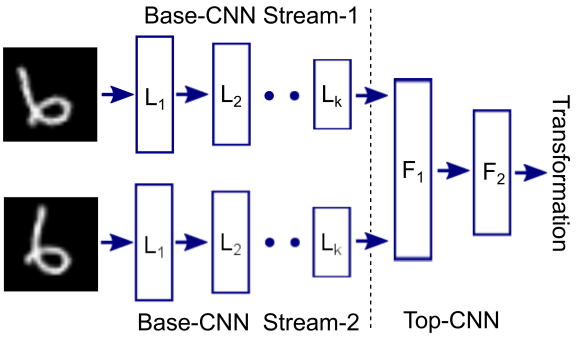


Figure 2.2.: Visualization of the proposed training method, using only a transformation given by egomotion information. The two Base-CNN Streams share their weights and each calculate a feature descriptor L_k of the given image path. The Top-CNN gets a concatenation of both descriptors and calculates a rough estimate of the transformation between the input image pair. The translation error is used as a training signal for the whole network. After training, the Top-CNN is discarded and a single Base-CNN is used to calculate feature descriptors, e.g. for scene recognition. From [4]

The authors of Learning to See by Moving [4] propose a method for using egomotion information as a training signal for feature descriptors. This is in contrast to having class labels like with 3D Match, where each pair of 3D voxels either represents the same location or not. The approach is inspired by living organisms, which develop their visual perception in order to move around and interact with their environment. Also, there is research that indicates that egomotion information may be crucial for the development of visually guided behavior in kittens [17]. Since most robot systems already have a way to generate egomotion information, i.e. through incremental localization, such methods would be very useful. The proposed method uses a CNN to generate a feature representation for 2D image patches, but the technique could be easily adapted to work on 3D data

as well. It is important to note that the feature representation was evaluated on scene and object recognition tasks. Figure 2.2 visualizes the training method they used for their toy-example on the MNIST dataset. Classical approaches would train a single Base-CNN Stream $L_1, \ldots, L_k$, using labeled images to calculate a training signal, depending on whether the label was estimated correctly. Instead, the two Base-CNN Streams are trained in a Siamese fashion, sharing their weights. Each Base-CNN Stream gets fed similar image patches, which are slightly transformed using a known transformation. Their output-layers L_k are appended and fed into another CNN (Top-CNN), which learns to create a rough estimate of that transformation from the two output-layers. The estimated transformation is only used to force the Base-CNN Streams to learn a useful representation and is not used for any further tasks. Since the transformation between the image patches is known, a transformation error can be estimated and used as a training signal for the whole network, including the Base-CNN Streams. For the MNIST dataset, the transformation was applied artificially. During their other experiments, they used images from datasets with attached egomotion information, for example from the KITTI dataset [15]. After the training, the Top-CNN is discarded and the Base-CNN's output layer L_k is used as a feature representation for recognition tasks.

While CNN-based, environment-adaptive feature descriptors are an interesting concept, this thesis instead focuses on optimizing the whole pipeline as a black box. That is due the fact that even with a perfect descriptor, other parts of the pipeline could break submap matching when their parameters do not fit the robot system or environment. Also, the descriptor methods described in this chapter still have some parameter similar to a search radius which needs to be tuned somehow (i.e. 3D Matches' voxel size).

3. Theoretical Background

This chapter describes the methods on which the proposed solution is built upon. Section 3.1 briefly introduces the notation used in this thesis. In Section 3.2, the general structure of feature-based map matching methods are described. The concrete map matcher used during this thesis' experiments, however, is described in Section 5.1. Section 3.3 describes the Bayesian optimization technique used in the proposed solution.

3.1. Notation

In this thesis, scalar values are denoted by non-bold, lower-case letters like a. Bold, lower-case letters, like $\mathbf{x}$, are used to denote vectors. Matrices are denoted with bold, upper-case letters like $\mathbf{X}$. Finally, sets are denoted with non-bold, upper-case letters like D.

3.2. Feature-Based Map Matching

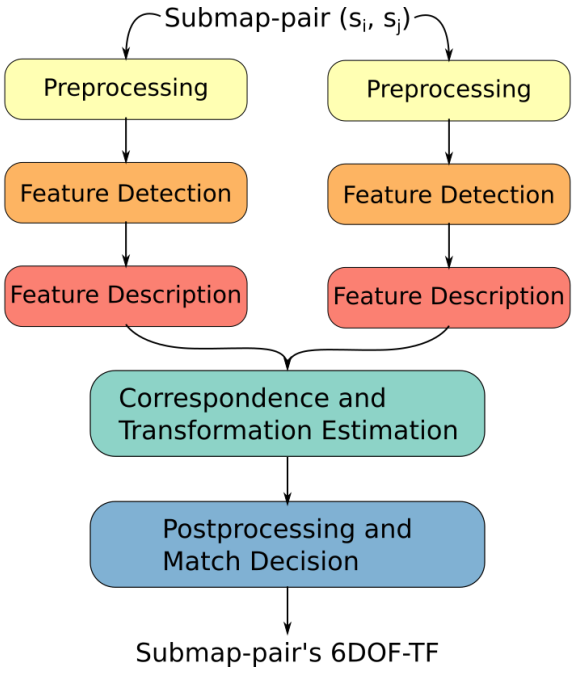


Figure 3.1.: The general structure of a map matching pipeline, from a submap pair to the resulting transformation.

This section describes the general structure of feature-based map matching solutions for 3D point cloud data as visualized in Figure 3.1 and its application for SLAM. Similar solutions are also used for other types of input-data, for example 2D image matching for panoramic image

stitching [9]. A description of the concrete system used in this thesis' experiments is given in Section 5.1. However, the proposed solution is in no way specifically tuned to that system.

3.2.1. Context: SLAM Systems

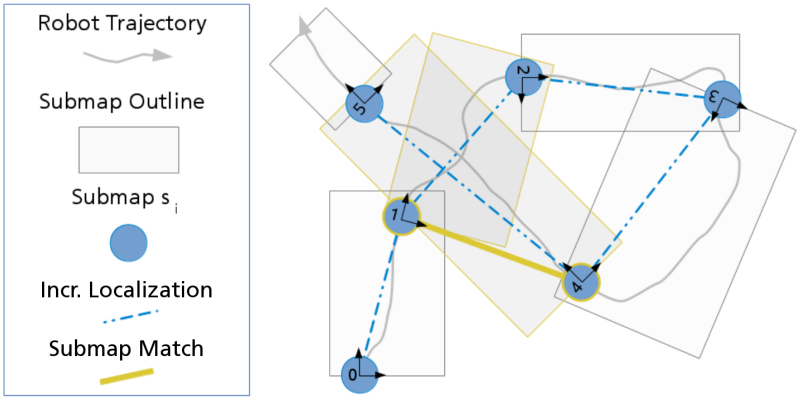


Figure 3.2.: A small, exemplary SLAM graph with multiple submaps, connected by transformations from the incremental localization system (blue, dotted edges). The two highlighted submaps were matched by the map matching pipeline, resulting in a loop-closure (yellow edge). Slightly adapted from [40].

Feature-based map matching is normally utilized as part of a SLAM solution on a robot system. Its main task is loop-closure detection to reduce the accumulated error of incremental localization methods. Figure 3.2 visualizes a robot's trajectory, with the submaps it creates along its path. The goal of map matching is to determine whether a submap pair (s_i, s_j) matches, i.e. that its 6-DOF transformation T_{ij} can be estimated. In the most basic case, all submap pairs are considered for map matching. However, especially for bigger environments, heuristics can be used to exclude submap pairs. This reduces the computational load and the chance to get false positives. For example, matching subsequent submaps usually yields transformations with much higher error than the transformation supplied via incremental localization. Another such example are submap pairs that have a very low chance of overlapping, because the incremental localization estimates their origins to be very far away, even when considering the accumulated error. In the context of SLAM, incremental localization and map matching are part of the SLAM front-end, abstracting sensor data into models that are well suited for estimation [11]. The information from the front-end is often recorded into a factor-graph [26], expressing interdependence of variables. For example, after matching two consecutive submaps, there will be two edges connecting their origins in the factor graph: One from the map matcher and one from the incremental localization. The SLAM back-end is used to estimate optimal values for a set of variables, based on the factor-graph's information. Those variables include transformations between submaps to create a consistent, global map, and the robot's trajectory. A popular framework for the SLAM back-end is, for example, g2o [27].

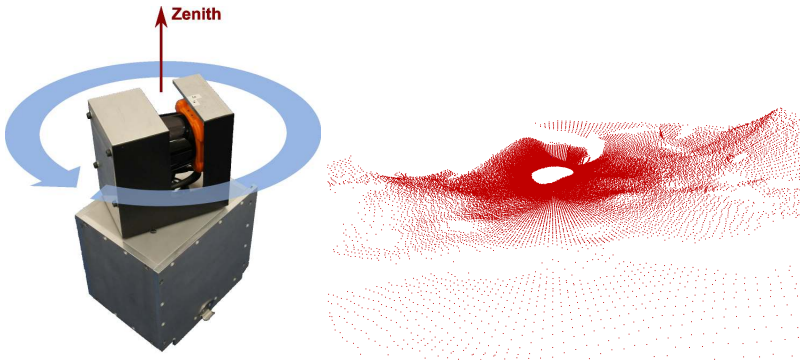


Figure 3.3.: KaRoLa (*Ka*rlsruhe *Ro*tating *La*serscanner, left image) creates 3D panoramic point clouds of uniform angular resolution (right image). From [32].

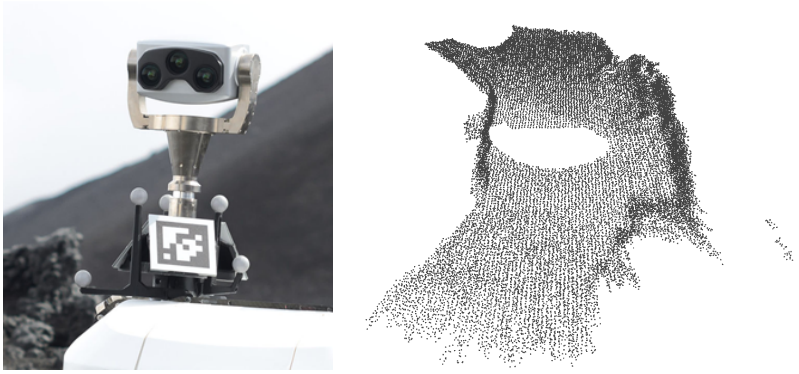


Figure 3.4.: The LRU's (see Section 5.1) stereo camera system on a pan tilt unit (left image) creates point clouds of uniform Cartesian resolution (right image). Note that the point cloud is not directly generated by the sensor like this. However, it is the point cloud a map matching system using this sensor system will receive as input.

3.2.2. Data Generation

While data generation is not usually considered part of a map matching solution, it is important for understanding how information about the environment is presented to the map matcher. During most of this thesis, it suffices to assume that each submap is represented by a single 3D point cloud. However, depending on the specific sensor setup, this point cloud may be created by multiple, separate measurements. If that is the case, those measurements are usually consolidated into a single point cloud during data generation. For example when using a rotating 2D laser scanner like depicted in Figure 3.3, the stepper motor's motion is known and the 2D scan lines can be registered into a 3D point cloud. In case of stereo-camera systems like in Figure 3.4, incremental localization can be used to align multiple frames, as long as its error stays small enough. If the localization error relative to the submap's origin increases over some threshold or if some maximum driving distance is reached, the submap is finished. No more points will be added to its point cloud. Instead, a new submap generation process will be started. Some approaches, including the reference system (see Section 5.1), supply additional information per submap. For example, for the keypoint filtering step in Section 5.1.3, an OctoMap [20] representation is used. It is a probabilistic, octree-based data structure, which is capable of distinguishing unknown space from free space.

Data generation depends on many different parameters, which usually need to be chosen by the user. In all cases, some calibration parameters for the sensors are needed. When multiple measurements are accumulated using incremental localization, the threshold at which to start a new submap is another important parameter. While those parameters can have a huge impact on the map matcher's performance, also considering them for optimization goes beyond the scope of this thesis. Approaches for automated calibration are described in Section 2.2.

3.2.3. Preprocessing

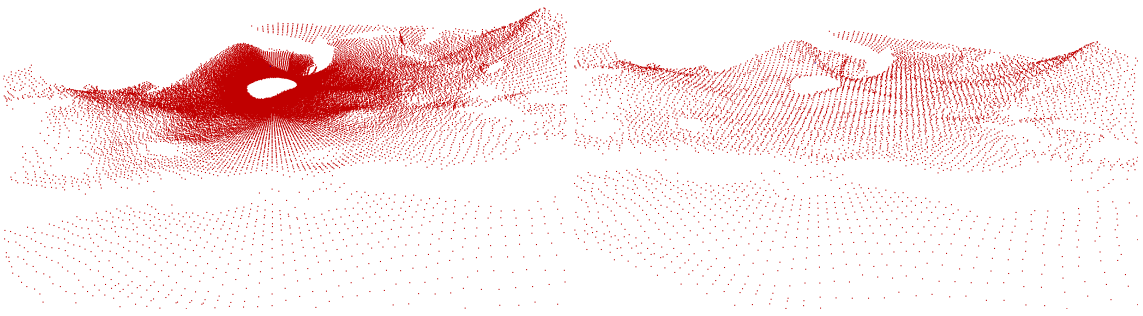


Figure 3.5.: A 3D point cloud before (left) and after (right) applying a voxel grid filter.

Preprocessing is the first step of a map matching pipeline and is done for both submaps separately. It usually starts with a voxel grid filter, to reduce and homogenize each point cloud's resolution as visualized in Figure 3.5. The filter overlays the point cloud with a grid of voxels with a parameterizable (usually cubic) side-length. All points contained in a voxel are then averaged to a single point. The choice of the voxels' side-length determines the resulting point cloud's resolution. Smaller side-lengths will leave more geometric details, higher values will reduce the computational load and remove more noise for the following steps of the pipeline.

Since most methods for reasoning about 3D geometry require the use of point-normals, it makes sense to calculate them only once, as part of the preprocessing step. Normal estimation can be thought of as fitting a plane into the point and its neighborhood, then using that plane's normal as the point's normal. For example an eigenanalysis of the covariance matrix of the point's local neighborhood can be used to estimate the normal [34]. While there are multiple methods for estimating a point's normal, all use neighboring points. Therefore, the normal estimation step requires as parameter either a search radius or the number of nearest neighbors used. Smaller values will preserve small changes in curvature and have a lower time-cost, bigger values will make normals more stable w.r.t. noise, especially in sparser regions of the point cloud.

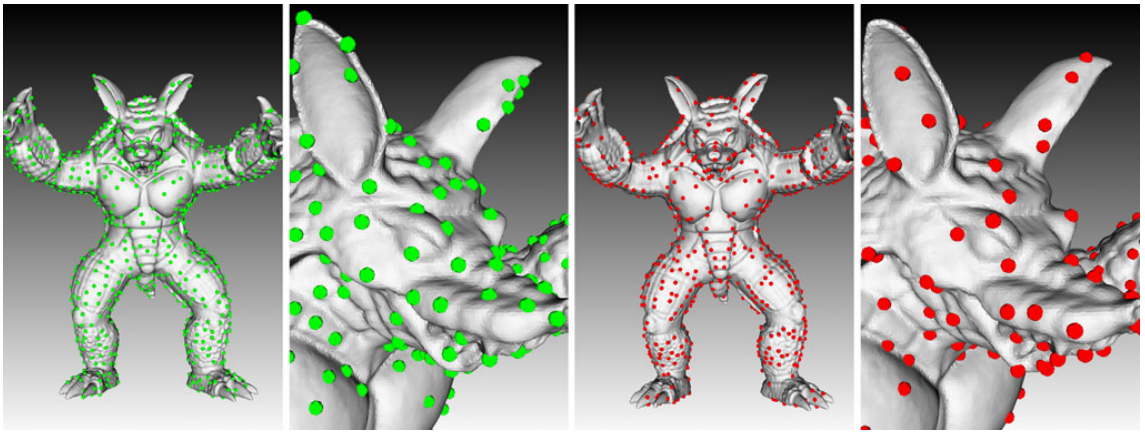


Figure 3.6.: Example keypoints detected on the Armadillo model of the Stanford 3D Scanning Repository [2]. Keypoints on the left image pair were detected using local surface patches as described in [12] and those on the right pair by Intrinsic Shape Signatures (ISS) [52]. From [48].

3.2.4. Feature Detection

After preprocessing, a feature detection method is used to calculate a saliency value for each point. This value is used to find a subset of points, which are well suited to be described by the descriptor method (see Subsection 3.2.5). Those points are referred to as keypoints or feature points. There are multiple reasons to not simply calculate a descriptor for every point of the point cloud, most prominently because of the time that would take. Important measures for keypoint quality are *repeatability* and *distinctiveness*. Repeatability measures the keypoint's independence of changes in the sensor's viewpoint or the illumination conditions (if relevant). Given a pair of submaps with overlapping areas, keypoints in those areas should roughly be at locations in each submap that correspond to the same physical location. Distinctiveness describes a keypoint's ability to be effectively described and matched, which will yield better correspondences. However, there is no method to measure keypoint distinctiveness in an isolated way, since it depends on the feature description method used. Most keypoint estimation methods were developed for recognizing and registering objects like those shown in Figure 3.6. A survey and performance evaluation of multiple such methods has been published in [48]. Feature detection methods always use a set of neighboring points to determine the saliency of the point p_i. The neighborhood's size is dependent on a support radius parameter. Smaller values reduce the time-cost of finding keypoints and tend to make the saliency score dependent on smaller changes in geometry. Higher values tend to produce more stable keypoints. Often, this step also depends on a separate parameter, which determines the keypoint density. Keypoint density can for example be controlled by using non-maximum suppression, which will only allow one keypoint (with the highest saliency score) in a specific radius. Depending on the feature detection method used, the keypoint density can also be directly derived from the support radius instead. A lower keypoint density reduces the time-cost of calculating and matching feature descriptors and potentially produces a higher ratio of good keypoints. By choosing a higher keypoint density, the chance of not having important keypoints decreases. Depending on the transformation estimation step's robustness, having redundant and

partially wrong correspondences may not a big issue.

3.2.5. Feature Description

In this step, a descriptor is calculated for each keypoint. A descriptor is a vector that encodes certain attributes of a point and its neighborhood. Desired descriptor properties are *descriptiveness*, *compactness* and *robustness* against certain nuisances. Descriptiveness measures, how well a method encapsulates information around a point, so that the yielded descriptor is different, for different areas, and similar for the same area, or an area indistinguishable from it. A feature description method is robust, if the yielded descriptors are insensitive to noise or variations in the point cloud's resolution. In [16], compactness is defined as the descriptiveness per vector-element. The dimensionality of the feature descriptor influences the memory costs of storing them and the time costs of distance calculations. Therefore, it is desirable to keep the feature descriptor's dimensionality as low as possible without losing important information. Like with feature detection, most descriptor methods were developed for object recognition and registration tasks. A survey and performance evaluation of multiple such methods has been published in [16]. Feature description methods usually only depend on a support radius which determines the size of its neighborhood. The choice of this parameter depends on the size of salient features in the environment.

3.2.6. Correspondence and Transformation Estimation

This is the first step in which information, previously calculated for each submap separately, is combined. With the perfect feature descriptor, keypoints represent the same physical point if and only if they are close in descriptor-space. Since no feature description method is perfect, there are several heuristics used to determine the correspondences. To find point-to-point correspondences between the submaps, for each keypoint of submap s_i, a set of k nearest neighbors (in descriptor-space) in the other submap s_j are considered if their distance is below some threshold. One way to proceed is to only take those correspondences which are reciprocal, i.e. those point-pairs that have each other as their nearest neighbor. However, maybe the correct correspondence could have been the one to the second nearest neighbor, so often more correspondences to the same point are allowed. There are one or more parameters which control which and how many correspondences will be found, for example the aforementioned distance threshold or the choice of k for the nearest neighbor search. Having more correspondences increases the risk of passing too many false correspondences to the transformation estimation, making it fail. However, if the parameter-values are chosen too conservatively, chances are that not enough correspondences can be found to robustly estimate a transformation.

Naively, estimating a transformation between the submap pair is done by minimizing the sum of euclidean distances of all corresponding points. Since there usually are quite a few wrong correspondences, including some, which would induce a vastly different transformation, more robust methods for transformation estimation are used. One of those methods is called *Random Sample Consensus* (RANSAC) [14]. It starts with a random correspondence-subset of size three, to estimate a transformation. To evaluate this transformation, the method distinguishes between inlier-

and outlier-correspondences, based on a threshold on the distance of the corresponding points after applying the transformation. A transformation with many inliers is likely to be a better choice than one with less. Also, the residual error is used as a measure for how well the transformation fits the inlier-correspondences. This process is repeated multiple times with different, random subsets of correspondences. After a fixed number of iterations or when some transformation quality criterion is met, the best transformation is chosen. Transformation estimation usually depends on parameters which express the necessary and possible transformation precision. In the case of RANSAC, the inlier threshold is used for this. A stricter threshold will reduce the probability of using wrong correspondences to estimate the transformation, making matches more likely to be correct. But a too strict threshold will also filter correspondences due to noise in the point cloud data, which may lead to missing correct matches.

To further improve the resulting transformation, usually *Iterative Closest Point* (ICP) [5] is used to minimize distances between close points of the complete point cloud, using the estimated transformation as initialization.

3.2.7. Postprocessing and Match Decision

Finally, the map matching pipeline classifies whether the submap pair matches. This is generally done based on information from the previous step, like thresholding the number of inlier-correspondences or the residual error, if using RANSAC. There are usually some additional, hand-crafted checks, implementing knowledge about the specific field of application, to influence the decision. For example, if the robot is not capable of traversing extreme elevation, correct transformations between submap pairs can only have relatively small roll and pitch angles. Depending on the choice of the parameter values in this step, the system can be tuned to either detect less matches with higher confidence or to detect more matches, including wrong ones. However, the specific value also depends on the parameters of previous steps and the environment: If there are not many correspondences to begin with, due to an environment with meager features or a strict threshold for the transformation estimation, the decision threshold needs to be less strict.

3.3. Bayesian Optimization

This section aims to introduce the reader to the Bayesian optimization technique used in this thesis. Note, however, that the field of Bayesian optimization contains many more techniques not described in this section. A good resource for a more general and in-depth introduction to the topic is [42]. The basic structure of this section and its notation is also inspired by that source.

Bayesian optimization aims to solve the problem of finding a global maximizer of an unknown objective function f

$$\hat{\mathbf{x}} = \underset{\mathbf{x}\in\chi}{\operatorname{argmax}}\, f(\mathbf{x}) \qquad [3.1]$$

where χ is some design space, often a Cartesian product of compact subsets of $\mathbb{R}$. The minimum requirements for the objective function f is, that it can be evaluated at arbitrary query points $\mathbf{x} \in \chi$. Observing f at a query point yields a noisy output $y \in \mathbb{R}$, such that $\mathbb{E}[y|f(\mathbf{x})] = f(\mathbf{x})$. In

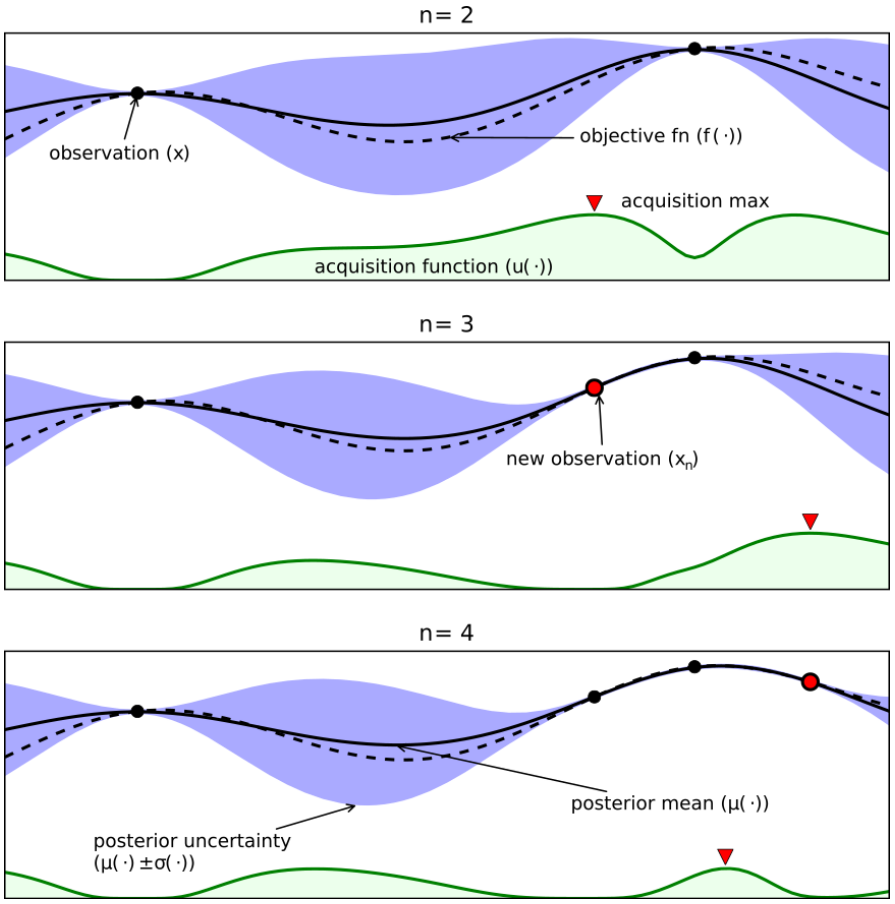


Figure 3.7.: Three iterations of an exemplary Bayesian optimization process. In the upper sections, the actual objective function is shown as a dashed line (usually unknown). The solid line is the mean of the surrogate model of the objective function and the blue area visualizes its uncertainty. New observations (red dots) further refine that model. The lower sections show the acquisition function, which combines known maxima (for exploitation) and the model's uncertainty (for exploration), determining the next observation's location. From [42].

this thesis, the Bayesian optimization process is formulated as a sequential search algorithm. At each iteration n, the algorithm selects a query point $\mathbf{x_{n+1}}$ at which to evaluate f to observe y_{n+1}. After N iterations, the algorithm's best guess $\mathbf{x}_\mathbf{N}^*$ for $\hat{\mathbf{x}}$ is returned. Note that $\mathbf{x}_\mathbf{N}^*$ is not necessarily the latest query point $\mathbf{x_N}$. Figure 3.7 visualizes that process on a simple exemplary problem with a known objective function.

A Bayesian optimization framework consists of two vital components. The first is a probabilistic surrogate model of the objective function, which gets initialized with our prior belief about probable objective functions. As new observations $(\mathbf{x_n}, y_n)$ are made, this model gets sequentially refined via Bayesian posterior updating. There are two basic classes of models for that component: Parametric models and nonparametric models. In the proposed method, the objective function is modeled by a *Gaussian Process* (GP, see Section 3.3.1), which is a nonparametric model.

The second part is an acquisition function

$$\alpha_n : \chi \to \mathbb{R}, \qquad [3.2]$$

induced by the first component's model of the objective function at iteration n. It is used to decide what the next query point $\mathbf{x_{n+1}}$ should be, by maximizing α_n

$$\mathbf{x_{n+1}} = \underset{\mathbf{x}\in\chi}{\operatorname{argmax}}\, \alpha_n(\mathbf{x}). \tag{3.3}$$

The theoretically perfect acquisition function would yield an optimal sequence of query points when maximized at each step, insofar as to minimize the number of necessary queries N to find $\hat{\mathbf{x}}$. However, commonly used acquisition functions are short-sighted heuristics, which combine the model's uncertainty and the model's estimation to balance exploration and exploitation. It is essential that the acquisition function can be efficiently evaluated, since it needs to be maximized to select a new query point. Acquisition functions considered in this thesis are described in Section 3.3.2.

3.3.1. Gaussian Process

A Gaussian Process $GP(\mu_0, k)$ is a nonparametric, generative model, fully characterized by its prior mean function and its covariance function, or positive-definite kernel. The prior mean function

$$\mu_0 : \chi \to \mathbb{R} \tag{3.4}$$

can be used to contribute expert knowledge about the objective function and the GP's estimation will revert back to $\mu_0(\mathbf{x})$ for $\mathbf{x}$s that are far away from any observations. In this thesis' experiments, the GP's mean function is constantly zero. Its covariance function

$$k : \chi \times \chi \to \mathbb{R} \tag{3.5}$$

describes the characteristics of probable objective functions, by defining how query points $\mathbf{x}$ and $\mathbf{x}'$ influence each other. The set of observations available to the GP are denoted as $D_n = \{(\mathbf{x_i}, y_i) | i \in [1,n]\}$. For GP regression, the observed values $\mathbf{y} = \{y_1, \ldots, y_n\}$ of $\mathbf{f} = \{f(\mathbf{x_1}), \ldots, f(\mathbf{x_n})\}$ are assumed to be normally distributed, given $\mathbf{f}$ with variance σ

$$\mathbf{y}|\mathbf{f}, \sigma^2 \sim \mathcal{N}(\mathbf{f}, \sigma^2\mathbf{I}), \tag{3.6}$$

and with $\mathbf{I}$ being the identity matrix. This allows the model to cope with noisy observations of the objective function. Of course, for $\sigma = 0$ exact observations can be modeled as well. Observations D_n induce a mean vector $\mathbf{m}$, with $m_i = \mu_0(\mathbf{x_i})$ and a covariance matrix $\mathbf{K}$, with $K_{i,j} = k(\mathbf{x_i}, \mathbf{x_j})$. With them, the GP's prior distribution can be written as

$$\mathbf{f}|\mathbf{X} \sim \mathcal{N}(\mathbf{m}, \mathbf{K}), \tag{3.7}$$

where $\mathbf{X}$ is the matrix of query points $\mathbf{x_i}$ of the observations D_n. When predicting the objective function at an arbitrary query point $\mathbf{x_q}$, the GP's posterior mean and variance functions are used.

They are

$$\mu_n(\mathbf{x_q}) = \mu_0(\mathbf{x_q}) + \mathbf{k}(\mathbf{x_q})^T (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} (\mathbf{y} - \mathbf{m}), \text{ and} \quad [3.8]$$

$$\sigma_n^2(\mathbf{x_q}) = k(\mathbf{x_q}, \mathbf{x_q}) - \mathbf{k}(\mathbf{x_q})^T (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{k}(\mathbf{x_q}), \quad [3.9]$$

with $\mathbf{k}(\mathbf{x_q})$ being the vector of covariance terms $k(\mathbf{x_q}, \mathbf{x_i})$, for $i \in [1, n]$.

Covariance Functions and Data Fitting

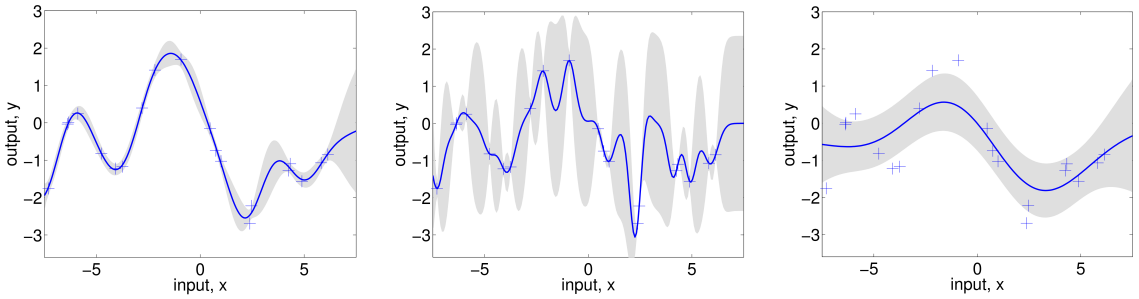


Figure 3.8.: Visualization of one-dimensional GP function regression with different length scale parameters l_1, using the $Matern_\infty$ covariance function. From left to right, the length scale values $l_1 = 1$, $l_1 = 0.3$ and $l_1 = 3$ were used. From [36].

There are many different covariance functions, including some for modeling periodic functions. For a detailed overview over covariance functions, see [3]. In this thesis, the Matern class of covariance functions is used, because it is a very flexible class of stationary covariance functions. Since stationary covariance functions are invariant to shifts, they can be written as

$$k(\mathbf{r}), \mathbf{r} = \mathbf{x} - \mathbf{x}'. \quad [3.10]$$

The different $Matern$ covariance functions are usually differentiated by their smoothness parameter $v > 0$. It is named that way, because the class of functions a GP with a Matern kernel yields, are differentiable $\lfloor v - 1 \rfloor$ times [36]. The exponential covariance function can be seen as a special case of the Matern covariance function class with $v = \frac{1}{2}$. And for $v \to \infty$, the resulting Matern covariance function represents the squared exponential kernel. Chapter 4.2.1 in [36] contains an in-depth explanation of Matern covariance functions and others. For the experiments of this thesis, the covariance function

$$k_{Matern\frac{5}{2}}(\mathbf{r}) = \sigma_f^2 \exp(-\sqrt{5\mathbf{r}^T \mathbf{L}\mathbf{r}})(1 + \sqrt{5\mathbf{r}^T \mathbf{L}\mathbf{r}} + \frac{5}{3}\mathbf{r}^T \mathbf{L}\mathbf{r}), \quad [3.11]$$

is used, where $\mathbf{L}$ is a diagonal matrix of length scales $l_i, i \in [1, d]$ and σ_f the function's amplitude. The length scales $l_1, \ldots, l_n$ represent how close samples have to be, to influence each other per dimension of the design space χ. Examples for different length scale values in the covariance function are visualized in Figure 3.8. The inverse of the length scales can also be interpreted as the relevance of the respective dimension. For example for a very large length scale value l_i, the covariance function's value becomes independent of the data r_i of the respective dimension. Covariance

functions with length scales as hyperparameters implement *Automatic Relevance Determination* (ARD) [31], which can be used to automatically remove irrelevant input from the inference [50]. The chosen covariance function is relatively flexible and allows modeling a broad range of functions. For future work, it may interesting to determine whether there exist common patterns in the function that maps map matcher parameters to a performance measure. If that's the case, this knowledge could be encoded in a covariance function, which would in turn decrease the number of necessary observations for a good estimate of that mapping. However, showing that these patterns exist independent from different environments and robot systems may prove difficult.

In the following, the remaining free parameters $\theta = (\sigma_f, \ldots, l_d)$ will be referred to as the GP's hyperparameters. Those hyperparameters θ are determined, by maximizing the log marginal likelihood of the observations D_n available to the GP. The log marginal likelihood is given by

$$\log p(\mathbf{y}|\mathbf{X},\theta) = \underbrace{-\frac{1}{2}\mathbf{y}^T\mathbf{K}_\mathbf{y}^{-1}\mathbf{y}}_{\text{data fit}} \underbrace{-\frac{1}{2}\log|\mathbf{K}_\mathbf{y}|}_{\text{complexity penalty}} -\frac{n}{1}\log 2\pi, \qquad [3.12]$$

with $\mathbf{K_y} = \mathbf{K_f} + \sigma_n^2\mathbf{I}$ being the covariance matrix for the noisy observations $\mathbf{y}$ of the objective function values $\mathbf{f}$. It is called the *marginal* log likelihood since the unknown objective function f is marginalized out. The equation can be interpreted in three parts. The data fit term is the Mahalanobis distance between the model predictions and the data, and is the only term which depends on the observed targets $\mathbf{y}$. The second term introduces a complexity penalty, which helps to prevent overfitting and depends only on the covariance function. Finally, the last term is a linear function which decreases the log likelihood with the number of observations n, for normalization. Most covariance functions, including the Matern covariance function used in this thesis, are differentiable with respect to their hyperparameters. As long as that is the case, the log marginal likelihood can also be differentiated and optimized with standard gradient-based optimization methods. In this thesis, the an improved version [30] of the L-BFGS-B algorithm [10] is used for that purpose.

3.3.2. Acquisition Function

The acquisition function

$$\alpha_n : \chi \to \mathbb{R}, \qquad [3.13]$$

of the Bayesian optimization framework determines the next query point $\mathbf{x_{n+1}}$. In the areas of experimental design and decision theory, the function α_n is often called expected utility. Its most significant task is to carefully weigh exploration, by choosing query points in uncertain areas, and exploitation, by choosing query points in areas where the objective function f is likely to have its maximum. This thesis uses the *Upper Confidence Bound* Algorithm for Gaussian Processes (GP-UCB) [43]

$$\alpha_{UCB}(\mathbf{x}) = \mu_n(\mathbf{x}) + \beta_n\sigma_n(\mathbf{x}), \qquad [3.14]$$

where β_n is a scalar, determining the weight of the GP's uncertainty $\sigma_n(\mathbf{x})$. Changing β_n causes the querying to focus more on exploration or exploitation. During each of this thesis' experiments,

the value $\beta_n = \kappa$ was fixed for all iterations of that experiment. Evaluating how to best change β, depending on the remaining time-budget of the optimization process remains for future work.

4. Approach

This chapter describes the proposed method for optimizing a map matching pipeline, by utilizing known transformations between submaps. Section 4.1 describes the different modules of the proposed method, how they interact with each other and which frameworks were used to implement them. Most modules represent existing techniques, which were described in Chapter 3. However, the performance measure is a novel method, which is described in Section 4.2. Section 4.3 covers viable information sources to use during the optimization. The chapter closes with a summary of all of the system's hyperparameters in Section 4.4.

4.1. System Architecture

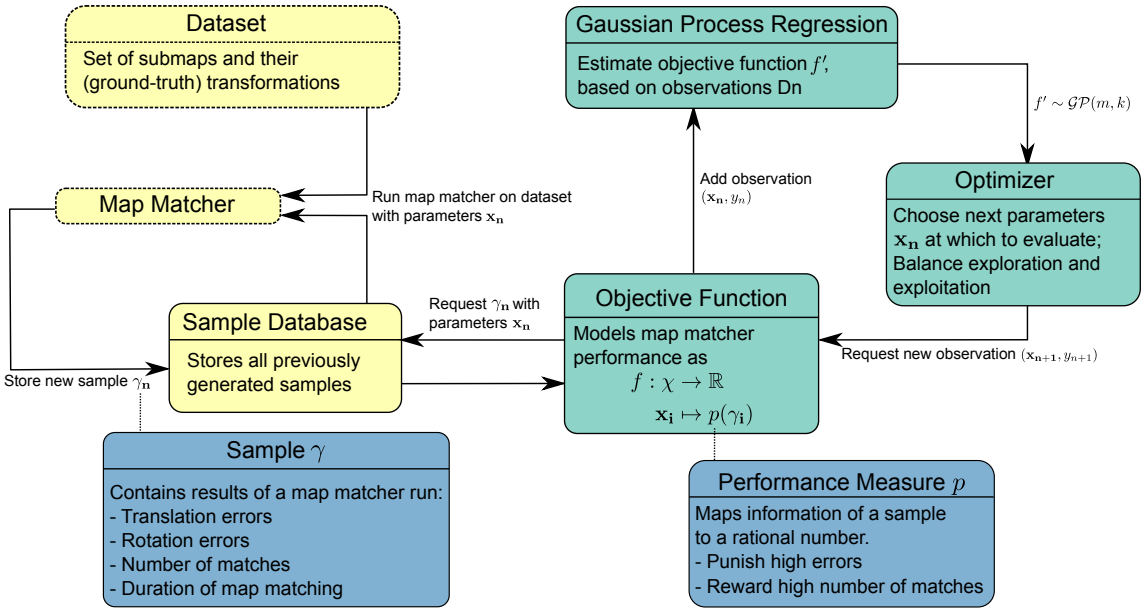


Figure 4.1.: An overview of the system architecture of the proposed method. The system can be separated into two parts: The map matcher part (yellow) and the Bayesian optimization part (turquoise). Additional information is visualized in blue.

Conceptually, the system is split into two parts. The Bayesian optimization (see Section 3.3) part is capable of maximizing an objective function $f : \chi \to \mathbb{R}$. For the thesis' use-case, the objective function's design space χ is the map matcher's parameter space. By finding $\hat{\mathbf{x}} \in \chi$, which maximizes f, the best possible set of parameters is found. Of course, the proposed method will not necessarily find the *global* maximum of f. The other part of the system consists of the map matcher itself (see Section 3.2), a dataset, and the *Sample Database*. A concrete map matching pipeline is used to define the objective function f through a set of discrete observations D_*, since its behavior

cannot be described analytically. For this thesis' experiments, the map matcher was parallelized on multiple machines over the network via ssh, to make the sample generation faster. The dataset consists of a collection of submap pairs with known transformations. Given a set of parameters $\mathbf{x} \in \chi$, the map matcher processes the dataset and outputs a set of properties, which describe the map matcher's performance. This process takes somewhere between a minute and several hours, depending on the size of the dataset and the choices for $\mathbf{x}$. The *Sample Database* receives tuples of $\mathbf{x}$ and the respective map matcher output, stores them and thereby makes them available to the *Objective Function* module. By using the performance measure described in Section 4.2, those properties can be mapped to $\mathbb{R}$, to yield an objective function suitable for Bayesian optimization. A python3 implementation of this system will be released on GitHub under the BSD 2-Clause license. It is built to optimize Robot Operating System (ROS) parameters, since the reference robot system for the experiments is a ROS-based system. It can be used with a specific map matching pipeline of your choice, by implementing an interface function for generating new observations.

The following paragraphs describe each module in detail, roughly following the overview in Figure 4.1 from left to right.

A complete dataset consists of one or multiple sub-datasets. Each sub-dataset contains a set of submap pairs with known transformations. The dataset encodes both information about the robot's sensors and the environments in which it was recorded. By optimizing on a dataset which contains sub-datasets from different environments or ones recorded by different robot systems, more general parameters can be learned. However, the bigger the dataset, the more time the map matching process and thereby the optimization process will take.

The *Sample Database* stores all observations D_* that were ever generated by the map matcher over the course of multiple experiments on the same dataset. An observation, as stored in the *Sample Database*, is a tuple $(\mathbf{x}, \gamma_\mathbf{x})$. $\mathbf{x}$ is the parameter set and $\gamma_\mathbf{x} = (m, \mathbf{e^t}, \mathbf{e^r})$ is the sample, which contains corresponding map matcher output. A sample consists of the number of matches m, as well as their respective translation and rotation errors $\mathbf{e^t}$ and $\mathbf{e^r}$. From the perspective of the Bayesian optimization modules, it is completely transparent whether a map matcher run is started or a previously generated observation is returned. Whenever an observation is requested that is not in the database, the generation process gets started automatically. To be able to recognize samples exactly, each parameter is rounded to two decimal places. Otherwise, the same observation would never be requested twice, e.g. due to floating point inaccuracies. At first glance, this module may seem like a negligible convenience feature. However, for the scope of this thesis' experiments, not having to regenerate previously generated observations proved to be invaluable. Additionally, having all observations stored, was very helpful for visualizing the discrepancy between the actual objective function and its approximation by the Gaussian Process.

The *Objective Function* module represents the interface between the Bayesian optimization part and the map matcher part. Whenever the *Optimizer* evaluates the objective function at a specific location $\mathbf{x}$, it requests the respective observation $(\mathbf{x}, \gamma_\mathbf{x})$ from the *Sample Database*. Note that the Bayesian optimization part of the system works on observation $(\mathbf{x}, y)$, with $y = p(\gamma_\mathbf{x}) \in \mathbb{R}$ instead of observations $(\mathbf{x}, \gamma_\mathbf{x})$ like the *Sample Database*. The performance measure p (see Section 4.2) is used to map the information of a sample $\gamma_\mathbf{x}$ to $\mathbb{R}$. The objective function also isolates map matcher

parameters from optimized parameters. The optimized parameters are exactly the design space χ. However, the complete set of map matcher parameters may be bigger. This can be the case for experiments where only a specific, small subset of parameters should be optimized. Additionally, implementations of map matcher algorithms tend to have a plethora of parameters which barely influence its performance or are only relevant for other parts of the robot system. Those need not and should not be optimized. The objective function takes care to set all non-optimized parameters to a specified default value before passing them on to the *Sample Database*. This way, the *Sample Database* always works with the complete set of map matcher parameters and the set of optimized parameters can be easily altered without invalidating the database.

The *Gaussian Process Regression* module creates an estimation of the objective function, based on a set of observations $D_n \subseteq D_*$. That estimation f' consists of a mean and a variance for each $x \in \chi$. For more information on the theoretical workings of that module, see Section 3.3.1. The scikit-learn python module[1] is used as implementation for this part of the system.

The *Optimizer* module uses the estimation f' to determine where the next query location $\mathbf{x_{n+1}}$ should be. This is done using the acquisition function α_{UCB} with fixed β, as described in Section 3.3.2. Then, the *Gaussian Process Regression* module can create an improved estimation, based on the new set of observations $D_{n+1} = D_n \cup (\mathbf{x_{n+1}}, y_{n+1})$ The whole process can potentially continue indefinitely, however, the *Gaussian Process Regression* module suffers very long estimation durations at roughly $n > 300$. Of course, the sample generation process takes a lot of time as well. With the datasets and map matcher used in this thesis, even experiments that ran over night, seldom produced over 200 observations. Therefore, the important trade-off in this module is between exploration and exploitation to get the most out of the few observations that can be made with a reasonable time budget. When focusing too much on exploitation, a good set of parameters x may never be found if the *Optimizer* is caught in a low local maximum. On the other hand, with too little exploitation, the *Optimizer* loses its ability to focus on the most promising areas of the objective function, thereby wasting time. How to chose this trade-off also depends on the size of the time budget available for the optimization process. There are possibilities to further improve the querying behavior of the *Optimizer*, for example by reducing β over time. This way, the *Optimizer* would first focus on exploration, when β is still big. As more time passes, the *Optimizer* would focus its exploration only on the most promising areas. However, this functionality is not yet part of the system and remains for future work. An open-source python module called BayesianOptimization[2] is used as implementation for this part of the system.

4.2. Modeling Map Matcher Performance

Finding a way to mathematically describe the map matcher's performance is essential for the optimization process. The performance measure induces the objective function f the *Optimizer* will try to maximize (see Section 3.3). It is important to note that the *Optimizer* will "cheat" as long as that results in higher values for f. For example, a badly designed performance measure could yield

[1] `http://scikit-learn.org/stable/modules/gaussian_process.html`, last accessed Jan 2018
[2] `https://github.com/fmfn/BayesianOptimization`, last accessed Jan 2018

high values for sets of parameters with no matches at all, since that implies no errors were made during matching. The behavior of the proposed performance measure in practice is visualized in Section 5.4. Note that the performance measure does not necessarily allow a direct comparison between different map matcher systems or different datasets. It is only used to compare the performance of a fixed map matcher system on a fixed dataset, while changing the parameters of the map matcher system.

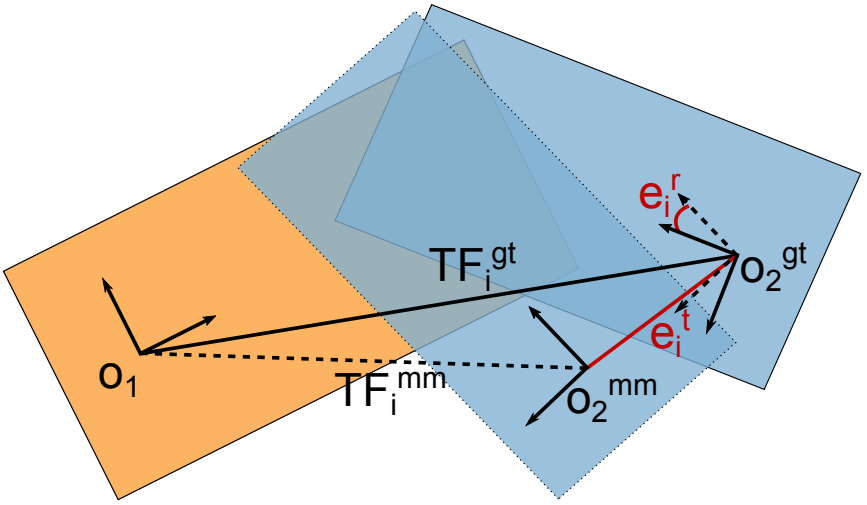


Figure 4.2.: Visualization of translation error e_i^t and rotation error e_i^r of a matched submap pair. The two blue rectangles both represent the second submap, once aligned with the ground truth transformation $\mathbf{TF_i^{gt}}$ and once with the transformation estimated by the map matcher $\mathbf{TF_i^{mm}}$. The dashed coordinate system in $\mathbf{o_2^{gt}}$ represents $\mathbf{o_2^{mm}}$ translated so that it lies in $\mathbf{o_2^{gt}}$. The rotation error e_i^r is simplified to the case in which only one angle is estimated during map matching (usually the yaw angle).

The data available for describing the map matcher performance consists of a list of matches with their corresponding rotation and translation errors. While the performance measure could generally map to $\mathbb{R}$, having a measure normalized to $[0,1]$ has some advantages. One corresponds to a very good map matcher performance and zero to a very bad one. With that, the performance measure is a function

$$p : (\mathbf{e^t}, \mathbf{e^r}, m) \rightarrow [0,1] \qquad [4.1]$$

with $\mathbf{e^t}$ and $\mathbf{e^r}$ being vectors of translation and rotation errors respectively. Both have size m, which is the number of total matches the pipeline made for the dataset. e_i^t and e_i^r both correspond to the errors of the ith match. Consider the first submap of a matched submap pair i as the base coordinate system $\mathbf{o_1} = 0^6$. Further consider the second submap's origin relative to $\mathbf{o_1}$ as estimated by the map matcher as $\mathbf{o_2^{mm}}$, and $\mathbf{o_2^{gt}}$ as its origin's location as dictated by the ground truth. Then, e_i^t is the euclidean distance between $\mathbf{o_2^{gt}}$ and $\mathbf{o_2^{mm}}$. And e_i^r is the angle of the axis-angle transformation between them. In case only one angle is estimated during map matching, e_i^r is simply this angle's error as visualized in Figure 4.2.

The proposed performance measure consists of a weighted sum of two separate performance

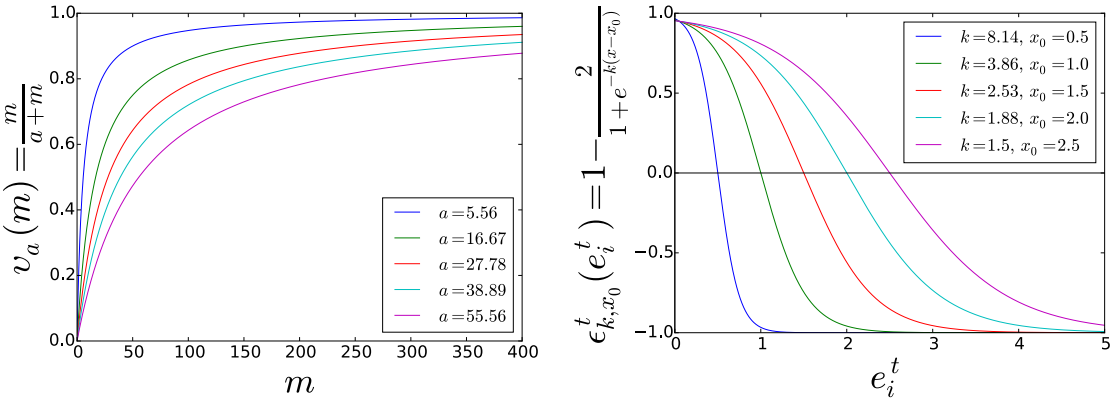


Figure 4.3.: Plots of the performance measure functions. On the left, $\upsilon_a(m)$ is shown, which rates the number of matches. On the right, $\varepsilon_{k,x_0}(\mathbf{e^t}, \mathbf{e^r})$ is shown, which rates translation and rotation errors.

measures

$$p(\mathbf{e^t}, \mathbf{e^r}, m) = w_\upsilon \cdot \upsilon(m) + w_\varepsilon \cdot \varepsilon(\mathbf{e^t}, \mathbf{e^r}), \tag{4.2}$$

with υ for the number of matches with weight $w_\upsilon \in [0,1]$ and ε for the errors with weight $w_\varepsilon = 1 - w_\upsilon$.

The number of matches is rated by

$$\upsilon_a(m) = \frac{m}{a+m}, \tag{4.3}$$

with a as a hyperparameter to adjust how many matches are considered good for a dataset as visualized on the left in Figure 4.3. $\upsilon_a(m)$ has a very high slope close to 0, which corresponds to the fact that the first few matches are the most important ones. The SLAM back-end does not necessarily need a huge number of matches, but a few are essential to get enough loop closures (see Section 3.2.1). When tuning parameters by hand, conservative parameter choices are often preferred, since wrong matches may instantly break the global map optimization in the back-end. Therefore, it is a positive characteristic that $\upsilon_a(m)$ quickly flattens when some number of matches were found. After that, the function very slowly converges to 1. This way, parameter sets with more matches will still be slightly better, but only if they do not introduce higher errors.

The measure for rating errors is

$$\varepsilon(\mathbf{e^t}, \mathbf{e^r}) = \max(0, \frac{1}{m} \sum_{i=0}^{m} \max\left(\varepsilon^t(e^t_i), \varepsilon^r(e^r_i)\right)), \tag{4.4}$$

with $\varepsilon^t, \varepsilon^r$ being measures for translation and rotation errors, respectively. The overall badness of a match is estimated by the maximum of the two error types. All individual match ratings are summed up and normalized by the number of matches. Normalization is necessary to ensure that parameters with many matches are not at a disadvantage. The final maximum function guards against a negative value for some edge cases, since ε^t can be negative for very high translation

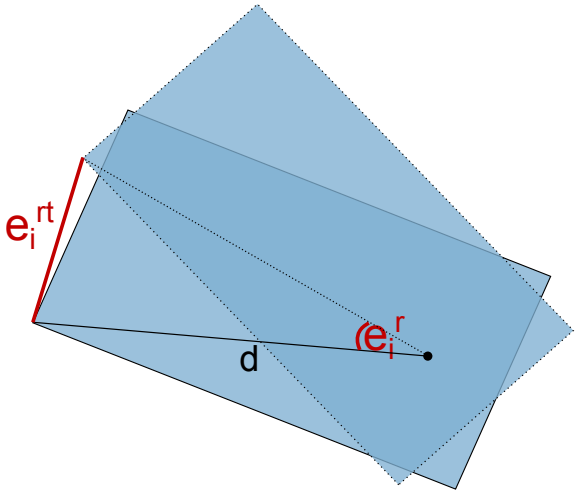


Figure 4.4.: A sketch of how the maximum translation error e_i^{rt} induced by rotation error e_i^r is calculated, using submap size d and some trigonometry (see Equation 4.2). The blue rectangles represent the same submap. The line with length d connects the submap origin with the farthest point in the submap. The one with dotted border is rotated by e_i^r, which induces the farthest point in the submap to be translated by e_i^{rt}.

errors. The measure for rotation errors is defined by using ε^t

$$\varepsilon^r(e_i^r) = \varepsilon^t(2d \cdot \sin(\frac{e_i^r}{2})), \tag{4.5}$$

and serves as a heuristic for the maximum translation error induced by the rotation error e_i^r as visualized in Figure 4.4. For this, the submap size d is necessary, which is the distance to the point farthest away from a submap's origin. Finally, to rate translation errors, a logistic function

$$\varepsilon^t_{k,x_0}(e_i^t) = 1 - \frac{2}{1 + e^{-k(x-x_0)}} \tag{4.6}$$

was adapted to map to $[-1,1]$ and has k and x_0 as hyperparameters. x_0 defines the point at which ε^t_{k,x_0} passes the x-axis and k changes the shape of the curve. Some examples for ε^t_{k,x_0} are plotted on the right side of Figure 4.3. The function has the highest slope around x_0 and converges to one or minus one for huge negative inputs or huge positive input, respectively. The slow convergence to one is beneficial, since variations close to $e_i^t = 0$ do not necessarily have meaning anymore: If the error gets close to the resolution of the point cloud or the matched features, those variations can only be caused by chance. Also, improving upon an already small error is not as important as the same improvement on a bigger error. That is due to the fact that single outliers with high transformation errors can cause a more severe degeneration of the global map in the SLAM back-end (see Section 3.2.1) than multiple small errors. However, when the errors get ridiculously large, this does not hold true anymore. Their impact on the global map can be prevented using robust error functions in the back-end. Intuitively, the system should only care to optimize the errors in the area where improvements are meaningful and the resulting transformations will assumed to be correct by the back-end.

The duration of the map matching process is currently not considered in this performance measure, but is definitely an interesting aspect to consider in the future. During the experiments of this thesis, the optimizer was prone to create sets of parameters which were relatively slow compared to hand-tuned parameters. Since the bounds of the design space χ were chosen so that it could not become too slow for practical application, this was not a huge problem. However, adding the duration to the performance measure would enable even more liberal choices for the bounds of the design space.

Another interesting option for future work on the performance measure is using a measure based on the quality of the global map. This would require using the SLAM backend of the robot system during the optimization process, increasing the system's complexity and duration for generation a new observation. However, this would get rid of the current performance measure's hyperparameters by implicitly encoding the SLAM backend's preference about the balance between many (potentially bad) matches and sparse, good quality matches. Since the definition of a 'good global map' may depend on the specific use-case for that map, the global map quality measure would probably introduce its own hyperparameters. For example if the main purpose of the robot's mission is creating a precise, high-resolution 3D map, small transformation errors would be worse than when the map is only used for localization purposes. Also, those hyperparameters should be easier to determine by the user and are not dependent on the size of the dataset like m^*.

4.3. Leveraging Incremental Localization Information

With the performance measure from the previous section, incremental localization information can be leveraged to optimize the map matcher. Per submap match, the performance measure uses another given (ground truth-) transformation to judge the quality of the map matcher's transformation estimate. This other transformation can come from any suitable source: Ground truth, incremental localization or even from detections of specific markers that are able to yield very precise transformations. In the case of ground truth information, all submap pairs of a dataset can be considered. This corresponds to step 1 of the problem statement in Section 1.1 and the respective experiment is described in Section 5.6.1. When restricting the map matcher on subsequent submap pairs, the transformation estimate from the incremental localization solution can be used as an alternative to the ground truth. This is due to the fact that incremental localization solutions are usually way more precise in a local area than map matcher systems. In case of the reference robot used for the experiments of this thesis, the creation of a new submap is bound to the incremental localization filter's uncertainty. It is set in a way that ensures, the standard deviation of the translation error between two subsequent submap origins never exceeds 10cm. This corresponds to step 2 of the problem statement in Section 1.1 and the respective experiment is described in Section 5.6.2. Finally, additional information sources like transformations from detections of specific markers could also be used. For instance in multi-robot teams, markers can be attached to each robot, to enable precise transformation estimates when two robots see each other. In case their positions are close enough that their respective submaps overlap, the transformation induced by the marker detection could also be used as training data for the optimization process. When enough of those

transformations are available in a dataset, a set of parameters capable of matching two different robot system's maps could be found. However, at the time of writing this thesis, no such data was available and this topic remains for future work.

4.4. Parameterization of the System

Since this thesis is partially about *reducing* manual parameter tuning labor, it is important to have a look at the new parameters the proposed system introduces. They will also be referred to as hyperparameters. This section will not discuss any parameters of the map matcher system used in this thesis' experiments. Those are discussed in Section 5.1 and are referred to as parameters, optimized parameters or complete parameters.

The most obvious hyperparameter is the choice of the design space χ and it determines, which subset of the complete map matcher parameters is optimized. χ also restricts the value range of each optimized parameter to a compact set with some lower and upper bound. The choice of χ is the most important hyperparameter and will have a huge impact on the runtime required for satisfying results. Also, depending on the map matcher implementation, bad values or combinations of values for some parameters may lead to unexpected crashes or seemingly indefinite map matcher durations. Fortunately, the choice of χ only depends on the used map matcher system and therefore needs only to be defined once. Afterwards, a well working set of parameters $\hat{\mathbf{x}} \in \chi$ can hopefully be found for arbitrary robot systems and environments, using that map matcher. The choice of χ for the evaluation is discussed in Section 5.2.

The *Optimizer* module depends on the type of the acquisition function α and its hyperparameters. In case of this thesis, since only α_{UCB} is used, the only relevant hyperparameter is β. It determines the trade-off between exploration and exploitation. When the *Optimizer*'s sampling behavior is satisfactory, β should be able to remain constant for different environments, robot systems and even map matcher systems. Only if the performance measure's value range changes, β would need to be adjusted. For the experiments of this thesis, this parameter is chosen as $\beta = 5$. The value was empirically determined by experiments on two parameters, which showed the χ will not get properly explored with a smaller value for β. Choosing β dynamically instead could be advantageous and remains for future work.

The *Gaussian Process Regression* module depends on the utilized covariance function. During this thesis experiments, only the $k_{Matern\frac{5}{2}}$ covariance function (see Section 3.3.1) is considered. Intuitively, the covariance function determines probable properties of the approximated objective function. The second hyperparameter is the observation noise σ. It allows the approximated objective function to deviate from the set of observation D_n. Since most map matcher pipelines utilize one or multiple elements which introduce some measure of randomness, for example RANSAC, non-zero observation noise is necessary. $\sigma = 0.005$ was chosen for the experiments of this thesis, based on the data from the Single Parameter Experiment 5.4. However, an elaborate evaluation of how to best determine values for those two parameters remains for future work.

The performance measure described in the previous section introduces new hyperparameters a, k, x_0 to define the functions υ and ε (see Equations 4.3 and 4.4). To make it easier to reason over

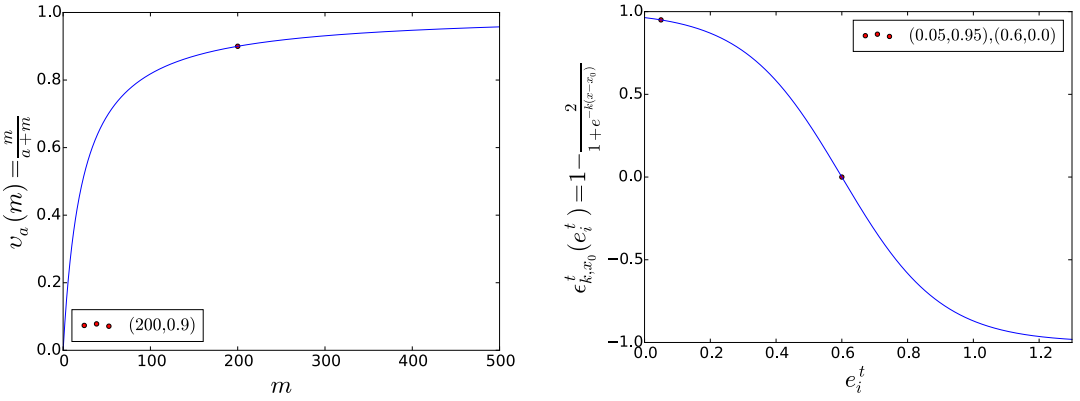


Figure 4.5.: Plots of the performance measure functions, parameterized by $e^* = 0.6$ and $m^* = 100$. On the left, υ_a's parameter a is chosen such that the function contains the point $(m^*, 0.9)$. On the right, ε_{k,x_0}'s parameter x_0 is set to e^* and k is chosen such that the function contains the point $(0.05, 0.95)$.

good values for those parameters, they have been replaced by different, more intuitive parameters. The measure for errors ε_{k,x_0} remains parameterized by x_0, but it can be thought of as the translation error e^*, at which improvements of the error should be most notable. The parameter k is instead defined such that ε_{k,x_0} always goes through the point $(0.05, 0.95)$. This way, the function evaluates to a value very close to one for errors values of about zero. The measure for the number of matches υ_a is instead parameterized by an expected number of matches m^*. By using m^*, a is chosen such that υ_a contains the point $(m^*, 0.9)$. The optimizer will therefore rather focus on reducing the errors, when the number of matches approaches m^*. Since υ_a never completely reaches one, another $\mathbf{x}$ with the same errors but more matches will remain slightly superior. The choice for m^* very strongly depends on the dataset used. If there are more submap pairs in the dataset which potentially can be matched, m^* needs to be bigger. e^*, however, is less sensitive to the dataset and can remain the same as long as the underlying SLAM system and its use-case remains the same. Only if the map's scale or the robustness of the SLAM system were to change significantly, e^* would need to be adjusted. The resulting performance measure functions for $e^* = 0.6$ and $m^* = 100$ are visualized in Figure 4.5. Finally, the weight w_υ of the number of matches needs to be defined. Giving the errors and the number of matches a balanced weight $w_\upsilon = 0.5$ is a good place to start, but if the resulting parameters are too error-prone or produce too little matches, the value can be adjusted. However, this was not necessary during the experiments of this thesis. Choices for e^* and m^* during evaluation, depending on the dataset used, are discussed in Section 5.3.

5. Evaluation

This chapter starts with a description of the reference system in Section 5.1, according to which the datasets used during evaluation were generated. It is followed by Section 5.2, which defines the set of map matcher parameters that are optimized (see Table 5.1). In Section 5.3, the two datasets and the reference parameter set $\mathbf{x}_{\mathbf{baseline}}$ are discussed.

The first experiment is described in Section 5.4, where each parameter is optimized separately. This experiment gives a glimpse of the high-dimensional data on which the proposed solution needs to work. Additionally, it allows the extraction of another reference parameter set $\mathbf{x}'_{\mathbf{1D}}$. The experiment in Section 5.5 shows the first optimization results of the proposed solution. However, during this experiment, the design space χ is only two-dimensional. This way, visualization of the objective function is still possible and correlations between the two optimized parameters can be seen more clearly. In Section 5.6, experiments that correspond to the practical application of the proposed solution are described. During the experiments, all map matcher parameters are jointly optimized. The experiments increase in difficulty, by reducing the amount of training data available. The chapter closes with Section 5.7, in which the resulting parameter sets are summarized and compared, and the experiment's implications are discussed.

5.1. Reference System

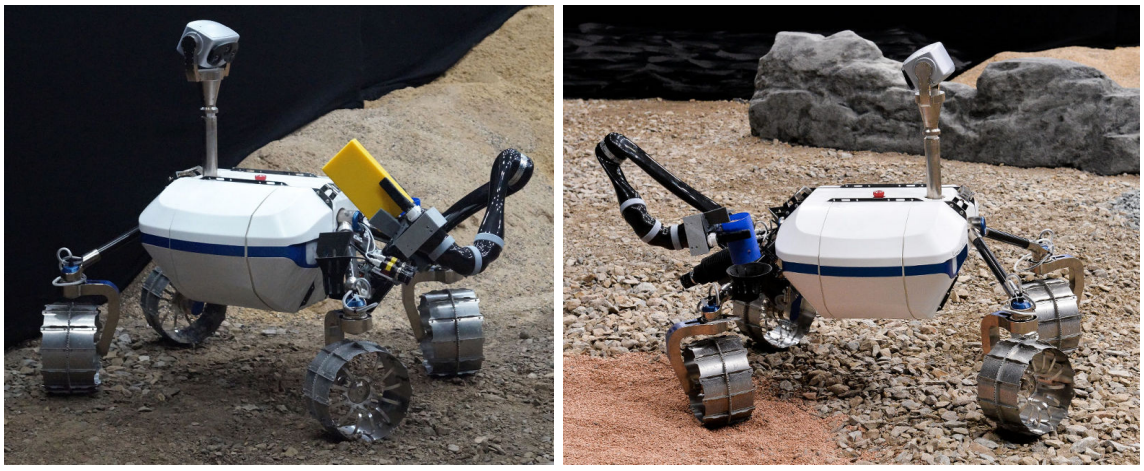


Figure 5.1.: The LRU during the SpaceBot Camp. From [39].

This section describes the hardware and the software relevant to the mapping system, which is currently used on the *Lightweight Rover Unit* (LRU). The LRU is used as the reference system for the experiments of this thesis. An overview over all aspects of the LRU is published in [49] and, more recently, in [41]. It has already successfully participated in multiple missions. For example

the SpaceBot Camp [39], a contest showcasing the state of the art for autonomous planetary exploration missions (see Figure 5.1). Another more recent example from 2017 is the ROBEX mission on Mt. Etna, Sicily, Italy. Its rough, unstructured terrain was used for a lunar exploration analogue mission, in which seismic measurement units were being placed autonomously. Figure 5.2 shows

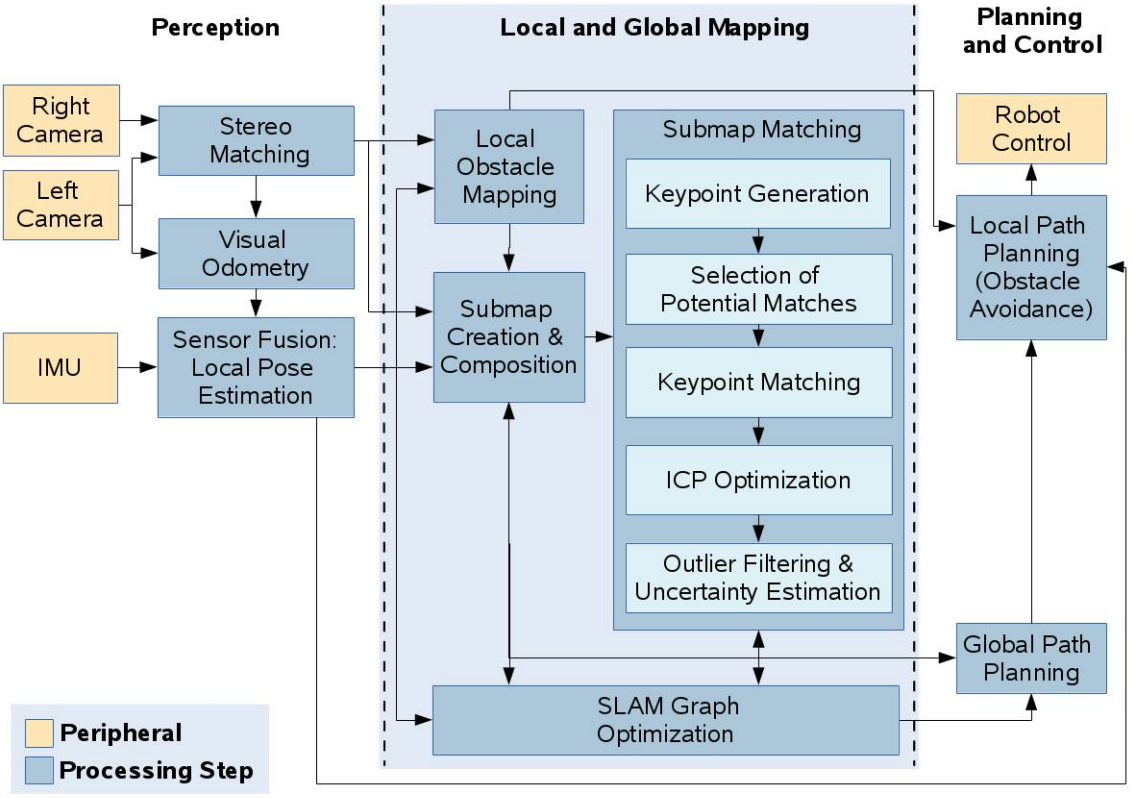


Figure 5.2.: Overview of the different modules which are relevant to the navigation system on the LRU. From [7].

an overview of the navigation system of the LRU. The Robot Operating System (ROS) framework is used as a middleware to connect the different modules. In recent publications [40], the system has been adapted to deal with multiple robots sharing their maps. The following sections describe the map matching components with regard to the framework introduced in Section 3.2. It focuses on aspects relevant for the optimization process. For a more detailed description, refer to the original publication [7] about the LRU's map matching system.

5.1.1. Data Generation

A black and white stereo camera pair with a baseline of 9 cm is used to generate depth images. An additional single RGB camera is used to map color information on the depth images for object detection. Those cameras are attached on a pan tilt unit to increase their effective field of view. The stereo matching is done on a Spartan-6 LX75 FPGA board, using *Semi-Global Matching* (SGM) [18]. All other calculations are done on a standard industrial computer with an i7–3740QM CPU (2.70GHz).

The incremental localization solution combines two types of sensors. First, the depth images are

used in a visual odometry system [19]. Then, its estimates are fused with IMU measurements and wheel odometry, using a keyframe-based *Extended Kalman Filter* (EKF) with time-delay compensation [38]. The incremental localization is used to align multiple depth images into a single point cloud, which represents a submap.

5.1.2. Preprocessing

Before a submap is considered for map matching, several safety checks are performed. Those include a minimum number of points and a minimum size, to make sure that the submap contains sufficient information for map matching. Also a statistical filter [37] is used to remove sparse outlier points, caused by stereo measurement errors. It removes points based on the distribution of distances to their neighboring points. Normal estimation is done using local least-squares plane fitting, with a search radius r_N.

The parameter *Normal Estimation Radius* r_N will be part of the optimization process.

5.1.3. Feature Detection

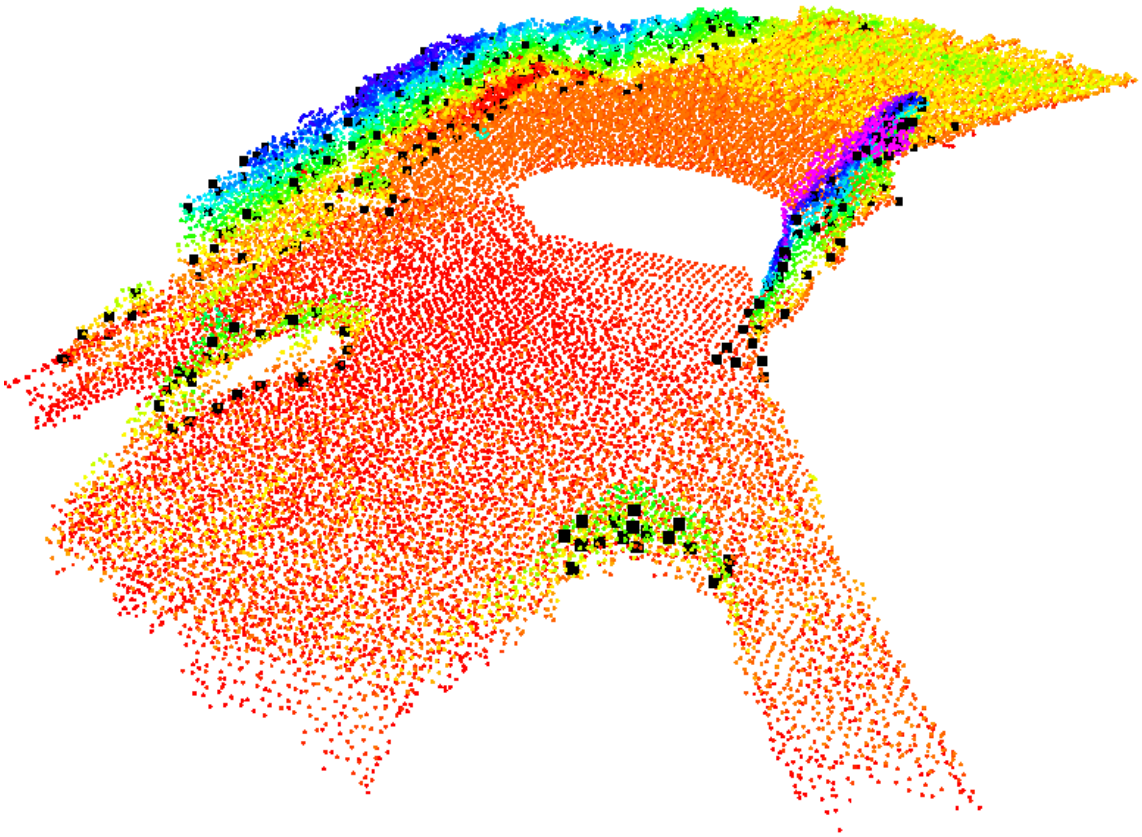


Figure 5.3.: Visualization of the detected keypoints of a submap, based on obstacle detections (black squares). The number of obstacles was reduced with a voxel grid filter with leaf size $0.2m$. The point cloud's color corresponds to its z-axis value.

In order to find points featuring a distinctive 3D geometry, this pipeline uses points from an

obstacle detection algorithm [6]. Its behavior deviates from the framework proposed in Section 3.2, because the obstacles are detected during the submap generation process. This is advantageous, since at that point, the algorithm can still use all information contained in the depth images. When the depth images are consolidated into a single point cloud, their viewpoints are lost. Using the viewpoint information, negative edges and slopes, like staircases and cliffs, can easily be detected. To reduce computational load, the number of obstacle points per m^3 is artificially reduced using a voxel grid filter with leaf size v_K. For each remaining obstacle point, a keypoint is selected via nearest neighbor search in the full point cloud. This results in keypoints like visualized in Figure 5.3.

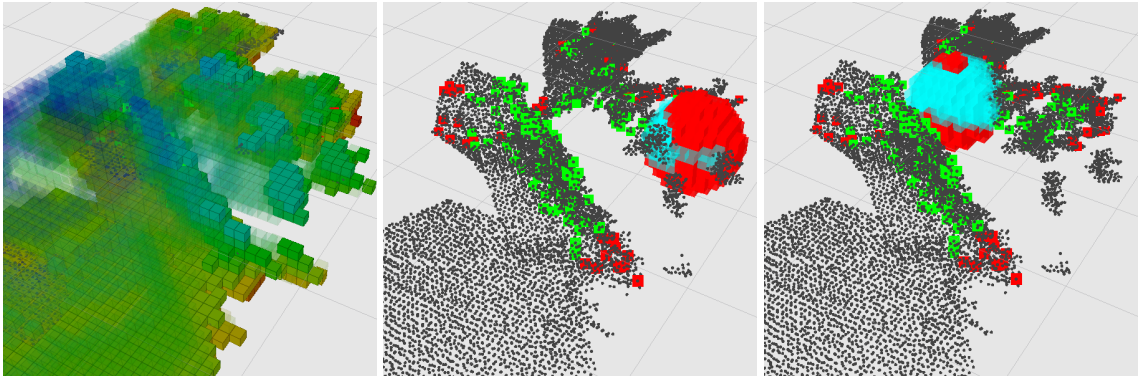


Figure 5.4.: Visualization of the keypoint filtering process (right image pair) and the OctoMap data structure (left image). The left image shows free space as transparent voxels and occupied space as opaque voxels. Each other voxel (not explicitly visualized) is unknown space. On the center and right images, two keypoints are considered for filtering. Around each keypoint, all voxels are visualized that would be encoded into that keypoint's descriptor, either in red (unknown) or in blue (known). The center image shows a keypoint that got filtered with $t_K = 0.6$ and $r_D = 0.5$. The right image shows a keypoint that did not get filtered with the same parameters.

After determining keypoints, an additional filter is used to remove keypoints too close to unknown regions. To do that, an octree-based representation of the submap is used that allows to differentiate between unknown, empty and occupied space. The data structure and two examples of the filtering process are visualized in Figure 5.4. To some extend, this prevents the feature descriptor from falsely encoding unknown regions as empty space. The filter step considers the ratio of known space to unknown space around each keypoint with the radius r_D used for feature description. If that ratio is under some threshold t_K, the keypoint is removed. For $t_K = 1$, no filtering will take place.

The parameters *Keypoint Voxel Grid Leaf Size* v_K and *Keypoint Filtering Threshold* t_K will be part of the optimization process.

5.1.4. Feature Description

CSHOT [47], an extension of SHOT [46], is used as a feature descriptor. The SHOT descriptor uses a histogram of orientations to encode information about the surface's topology. The CSHOT extension adds texture information to the descriptor to improve SHOT's descriptiveness. The feature description method depends on the descriptor search radius parameter r_D, as visualized in

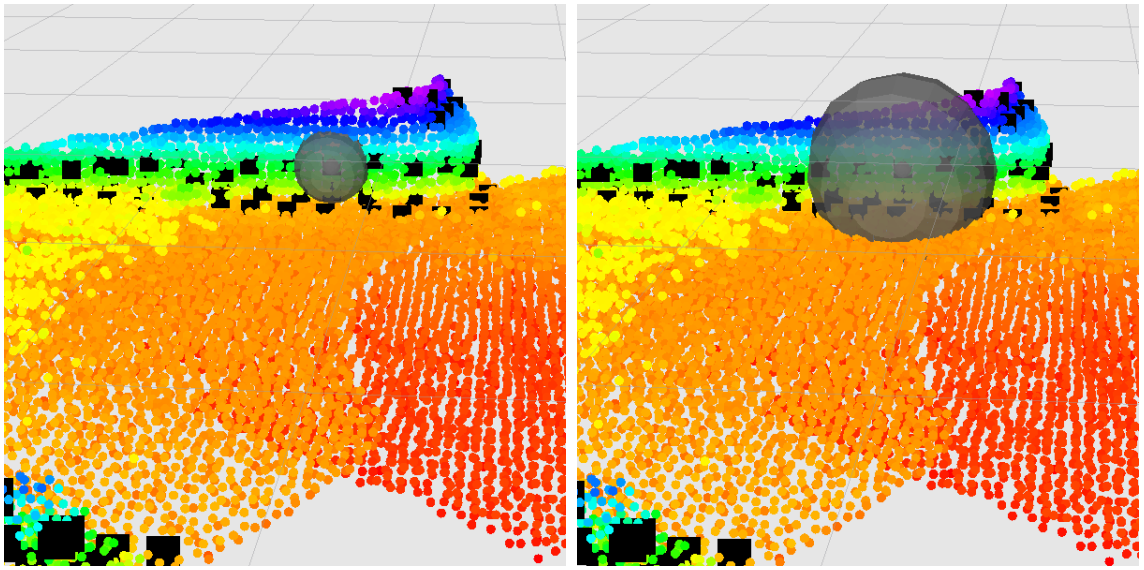


Figure 5.5.: Visualization of two different descriptor radii as transparent, gray spheres. The point cloud's color corresponds to its z-axis value. With $r_D = 0.2$, the descriptor can only encode that the keypoint lies on a flat surface, roughly perpendicular to the submap's origin. However, with $r_D = 0.5$, the fact that there is a 90° edge below the keypoint and that the surface ends above can also be encoded.

Figure 5.5.

The parameter *CSHOT Descriptor Radius* r_D will be part of the optimization process.

5.1.5. Correspondence and Transformation Estimation

First, an initial set of correspondences between point pairs (p_i^0, p_j^1) is created, where p_i^0 is a point from the first submap, and p_i^1 from the second submap. For each point from the first submap p_i^0, a correspondence is added to the closest n_C points in descriptor-space from the second submap, if their distance is lower than distance threshold t_C. The same process is repeated for the second to the first submap and duplicated correspondences are removed.

The parameters *Max Initial Correspondences* n_C, *Initial Correspondences Distance Threshold* t_C will be part of the optimization process

Using the resulting set of correspondences, intrinsically consistent subsets of correspondences are determined with a Hough3D voting [45]. The Hough3D algorithm requires each keypoint to have a local reference frame. It consists of the three eigenvectors obtained by applying an eigenvalue decomposition to the covariance matrix of that point's neighborhood. The size of this neighborhood is determined by the parameter r_{LRF}. The local reference frame is used to make the Hough3D voting process invariant to rotations. With the local reference frame, each keypoint knows a rotation-invariant translation to its submap's origin. This information is used, so that each correspondence can cast a vote where the first submap's origin should be relative to the second one. Those votes are clustered into bins of size v_{TF}, each representing one possible translation between the two submaps. Each bin with more votes then threshold n_{TF} is considered an intrinsically consistent subset of correspondences. Additionally, each remaining subset is filtered if its

corresponding keypoints are not distributed so that a stable transformation can be calculated from them. This is done by trying to fit a line into those keypoints, using RANSAC. In case they form a line, they are not fit for transformation estimation and removed from further processing. Finally, from the remaining subsets the one with the maximum number of correspondences is chosen. With this subset, RANSAC is used to estimate the complete transformation between the submap pair.

The parameters *Hough Voting LRF Radius* r_{LRF}, *Hough Voting Bin Size* v_{TF} and *TF Estimation Min Correspondences* n_{TF} will be part of the optimization process.

5.1.6. Postprocessing and Match Decision

The estimated transformation from the previous step is used as an initialization for the ICP algorithm, which refines the transformation estimate using the complete point cloud. Based on the thresholds of the previous sections, it is possible that no transformation can be estimated by Hough3D and RANSAC. Otherwise, the submap pair will be considered a match.

5.2. Defining the Design Space

This section gives a quick overview of the different parameters of the reference map matcher system, summarized in Table 5.1. Those are the parameters which will make up the design space χ, which the proposed system will optimize in during the following experiments. Note that two parameters are in fact discrete. Max Initial Correspondences n_C and TF Estimation Min Correspondences n_{TF} both expect values in $\mathbb{N}$ instead of $\mathbb{R}$. However, the Bayesian optimization implementation used for the evaluation does not explicitly support discrete parameters. When the *Optimizer* requests a new sample, values for those two parameters will likely be floating point values. The *Sample Database* will correctly round those values to integers. While this seems to work fine in the experiments conducted in this thesis, it may cause non-optimal behavior on the Bayesian optimization side. Therefore, correctly supporting discrete parameters remains an important topic for future work. See Appendix A for plots of how those parameters influence the map matcher performance.

5.3. Datasets and Baseline Parameters

Two datasets were used for the evaluation, one simulated outdoor dataset, and one recorded dataset of the DLR's robot lab. Both were simulated or recorded with the robot system described in Section 5.1. The simulation dataset depicts a moon-like environment, with relatively sparse features. It contains 40 separate runs of the simulated robot. During each run, on average, roughly 40 submaps were created, resulting in a total amount of about 1600 submaps. A visualization of the environment can be found in Figure 5.6. The second dataset was recorded with the actual robot system in the robot laboratory of the DLR. The environment consists of multiple artificial rocks. It contains 13 separate runs, each with 15 submaps on average. The environment is visualized in in Figure 5.7.

Symbol	**Parameter Name** / Notes	Bounds Lower	Bounds Upper	Pipeline Step
r_N	**Normal Estimation Radius** A smaller lower bound would be too close to the point cloud resolution. A higher upper bound slows down the map matcher too much.	0.1	2	Section 5.1.2
v_K	**Keypoint Voxel Grid Leaf Size** A smaller lower bound slows down the map matcher too much. The current upper bound already removes enough keypoints to prevent any matches.	0.05	0.5	Section 5.1.3
t_K	**Keypoint Filtering Threshold** Filtering is done via a ratio from zero to one.	0	1	Section 5.1.3
r_D	**CSHOT Descriptor Radius** A smaller lower bound would be too close to the point cloud resolution. A higher upper bound slows down the map matcher too much.	0.1	2	Section 5.1.4
n_C	**Max Initial Correspondences** A higher upper bound slows down the map matcher too much.	1	10	Section 5.1.5
t_C	**Initial Correspondences Distance Threshold** Distances in CSHOT feature space go from zero to one.	0	1	Section 5.1.5
r_{LRF}	**Hough Voting LRF Radius** A smaller lower bound would be too close to the point cloud resolution. A higher upper bound slows down the map matcher too much.	0.1	1	Section 5.1.5
v_{TF}	**Hough Voting Bin Size** A smaller lower bound would be too close to the point cloud resolution. A higher upper would certainly introduce too high errors.	0.1	1	Section 5.1.5
n_{TF}	**TF Estimation Min Correspondences** A smaller lower bound would allow models with insufficient information.	3	300	Section 5.1.5

Table 5.1.: A summary of the relevant parameters of the map matcher system used for the evaluation. Together with the lower and upper bounds, this table defines the design space χ in which the Bayesian optimization process takes place.

Using those two datasets and experiments during the LRU's missions, a set of parameters was found using manual parameter tuning. Lacking any established metrics for measuring global map quality, this set of parameters $\mathbf{x}_{\mathbf{baseline}}$ will be used to compare the proposed system's performance against. This comparison is a suitable way to evaluate the practical usability of the proposed solution, because the map matcher's performance with $\mathbf{x}_{\mathbf{baseline}}$ was sufficient for the LRU's missions. All significant sets of parameters discovered during this thesis' evaluation process are summarized and compared in Section 5.7. Additionally, $\mathbf{x}_{\mathbf{baseline}}$ is used to determine default values for parameters that are currently not optimized.

5.4. Single Parameter Experiments

During this set of experiments, the *Optimizer* module was not used at all. Instead, each dimension of the design space χ was sampled in a grid-like fashion. Note that the values of every param-

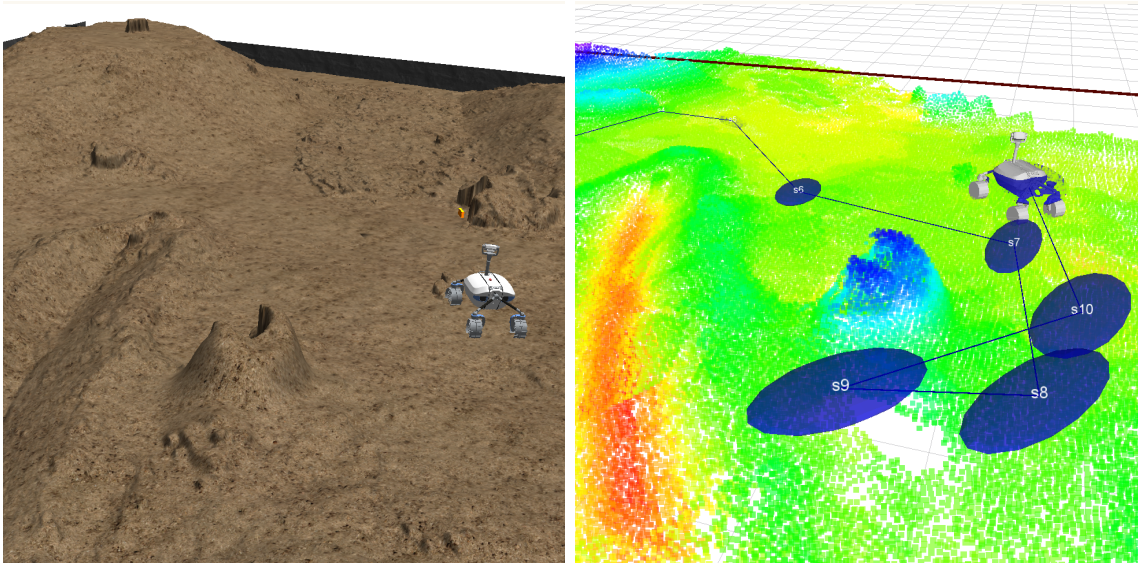


Figure 5.6.: A visualization of the simulation dataset. The left image shows a screenshot of the simulator. The right image shows a point cloud representation of multiple aligned submaps from the simulation environment.

eter that is not considered in the respective experiment, is set according to $\mathbf{x}_{\mathbf{baseline}}$. Plots of the resulting data can be found in Appendix A for both the simulation and the lab dataset. For both datasets, the hyperparameter of the error performance measure ε is set to $e^* = 0.4$ meters. This choice was motivated by the needs of the underlying SLAM system and the errors which usually occurred when using $\mathbf{x}_{\mathbf{baseline}}$. To find a good value for the hyperparameter of the matches performance measure υ, one has to consider the number of possible (correct) matches. Since the simulation dataset contains 40 runs, with 40 submaps each, the number of possible submap pairs is 40^2 per sub-dataset and 40^3 for the complete dataset (all runs). However, $m^* = 40^3$ would be way too large a value, because most of the submaps do not overlap. To find the optimal value for m^*, finding out how many submaps overlap with sufficiently salient features would be necessary. Doing this procedure manually would be incredibly labor intensive and a "sufficiently salient feature" is somewhat hard to define, since it depends on the very parameters which are to be optimized. Instead, the value is empirically set to $\mathrm{m}^* = 2500$ for the complete simulation dataset, based on results with the default parameter set. When using only subsequent submaps, the number of possible matches is reduced to about $40 \cdot 40 = 1600$. Since even subsequent submaps do not necessarily overlap in areas suitable for matching, the expected number of matches is set to $m^* = 1600 \cdot 0.3 = 480$. The factor of 0.3 is simply an educated guess of the fraction of matchable submap pairs in the dataset, based on the map matcher's performance with $\mathbf{x}_{\mathbf{baseline}}$. Following the reasoning for the simulation dataset, $\mathrm{m}^* = 200$ is chosen for the complete lab dataset. When only considering subsequent submaps for matching, the expected number of matches is set to $\mathrm{m}^* = 60$. Those hyperparameters will be used for all following experiments.

When comparing the difference of map matcher performance on the two datasets, it appears like the two environments do *not* require severely different sets of parameters. This is surprising, since the apparent need to tune parameters to specific environments was part of the initial motivation to

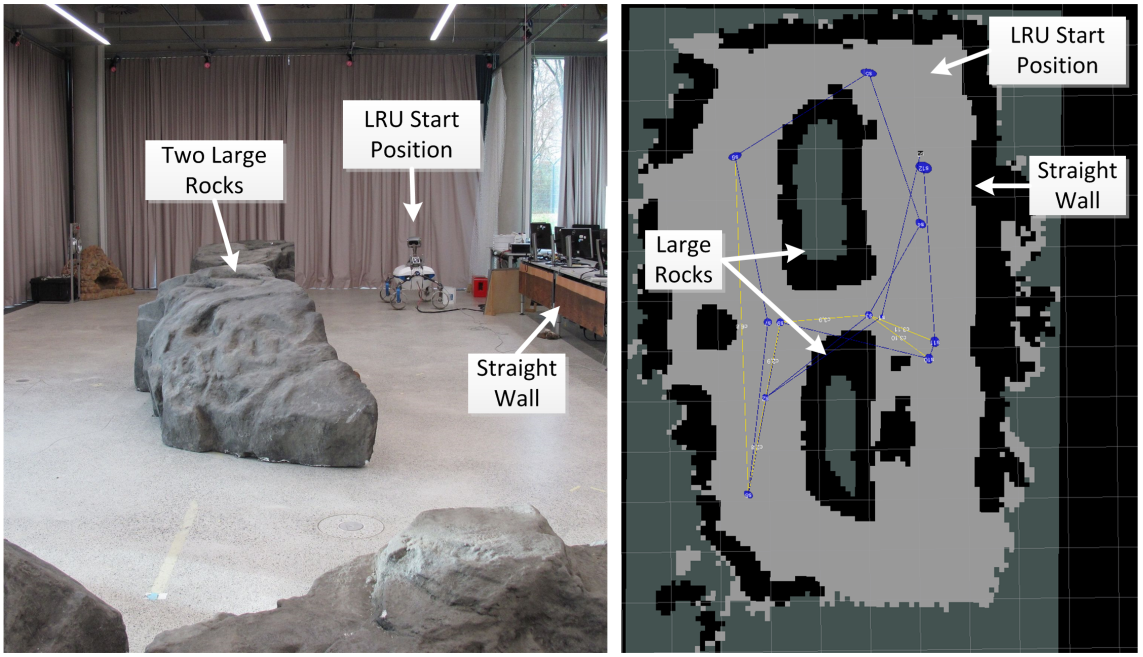


Figure 5.7.: A visualization of the lab dataset. The left image shows a photo of the laboratory environment in which the dataset was recorded. The right image shows grid map representation of the environment, with detected obstacles (black) in multiple, aligned submaps. In the visualized SLAM graph, blue ellipses are the submap origins, blue edges are transformations from incremental localization and the yellow edges are loop closures detected by the map matcher solution. From [28].

create this work. Yet, manual parameter tuning usually is not done by exhaustive sampling of the parameter space, since it takes a lot of time. To create the data visualized in Appendix A, about two weeks of computation time on five desktop PCs was required. However, slight differences can be spotted between the two datasets, for example for the Normal Estimation Radius parameter in Figures A.1 and A.2, or the CSHOT Descriptor Radius in Figures A.7 and A.8. To determine whether there is a set of map matcher parameters, which is mostly independent from the environment type, further evaluation on multiple datasets with more different environments is required.

By using the data of this experiment, another set of parameters $\mathbf{x_{1D}}$ (see Section 5.7) can be extracted to compare future optimization results against. For each parameter, the value with the highest performance measure is chosen, according to the data in Appendix A. However, the resulting parameter set $\mathbf{x_{1D}}$ does not yield any matches, because all keypoints are filtered during the feature detection step described in Section 5.1.3. This is owed to the fact that the data of this experiment does not contain any information about how the different parameters influence each other. Especially the parameters Keypoint Filtering Threshold (Figures A.5 and A.6) and CSHOT Descriptor Radius (Figures A.7 and A.8) require joint optimization to yield an optimal value. As discussed in Section 5.1.3, they are both used to filter keypoints too close to unknown areas of the submap. The Keypoint Filtering Threshold determines the maximum ratio of known space to unknown space to be encoded in a keypoint's descriptor. And the CSHOT Descriptor Radius determines the size of the volume around a keypoint, which is used for the keypoint's descriptor. Since most submaps have only a limited size in z-direction, increasing the radius requires a more

relaxed filtering threshold. Section 5.5 shows the correlations between those two parameters in detail. In theory, similar effects should exist for most other parameters as well, since each pipeline step depends on the results of previous steps. By deactivating keypoint filtering in $\mathbf{x'_{1D}}$, this experiment's data can still be used to find a competitive parameter set for comparing further results against.

5.5. Parameter Pair Experiment

This section contains an experiment, during which the design space χ was two-dimensional. The reason for doing separate experiments with only two parameters is that they are already able to show correlations between those two parameters, while still being visualizable. With three or more jointly optimized parameters, the possibilities to visualize the *Optimizer*'s behavior or the Gaussian Process Regression's estimates become very restricted. However, completely exploring a two-dimensional design space via grid search, like in Section 5.4, would have taken about two weeks with the resources available during the writing of this thesis.

The experiment which produced the figures discussed in the section is based on the data from the lab dataset. The design space consists of the two parameters Keypoint Filtering Threshold t_K and CSHOT Descriptor Radius r_D. In total, the experiment ran for 166 iterations, which took about 30 hours on four desktop PCs. This is mainly due to the generation durations of the 166 observations.

Figure 5.8 shows how the estimated mean of the objective function evolves over time. After about 70 iterations (i.e. with 70 observations), the estimated mean only changes very little. However, as visualized in Figure B.5, the best set of parameters of this experiment $\hat{\mathbf{x}}_{\mathbf{2D}}$ (see Table 5.2) is only found later, at iteration 160. The reason for this could be the relatively high value $\beta = 5$ for the acquisition function's hyperparameter. This means, the *Optimizer* module heavily focuses on exploration and only starts exploiting promising areas when the estimate's variance is low in all regions. It is important to note that there is no guarantee $\hat{\mathbf{x}}_{\mathbf{2D}}$ is actually the *global* maximum of the objective function. The figures in Appendix B additionally visualize all known samples, the variance and the acquisition function for the 8th, 20th, 40th and the 160th iteration of this experiment.

Additionally, the experiment clearly shows how the two parameters influence each other in Figure 5.8. Consider the dark areas, which correspond to parameters that lead to a very low map matcher performance score. Note that they expand further into sections with a higher Keypoint Filtering Threshold (less keypoints filtered), only where the CSHOT Descriptor Radius is big as well. The map matcher performance becomes zero, because all keypoints are filtered. Since the CSHOT Descriptor Radius is used to determine the size of the area where the ratio between unknown and known space is rated, the Keypoint Filtering Threshold needs to become less strict (bigger) if the radius increases. Also, the resulting parameter values $t_K = 0.95$ and $r_D = 0.34$ in $\hat{\mathbf{x}}_{\mathbf{2D}}$ greatly differ from $t_K = 0.4$ and $r_D = 1.2$ in $\mathbf{x_{1D}}$. This shows that joint optimization finds a different maximum, which indicates that the two parameters are correlated. In fact, the final evaluation in Section 5.7 shows that $\hat{\mathbf{x}}_{\mathbf{2D}}$ performs better than both hand-tuned reference parameter

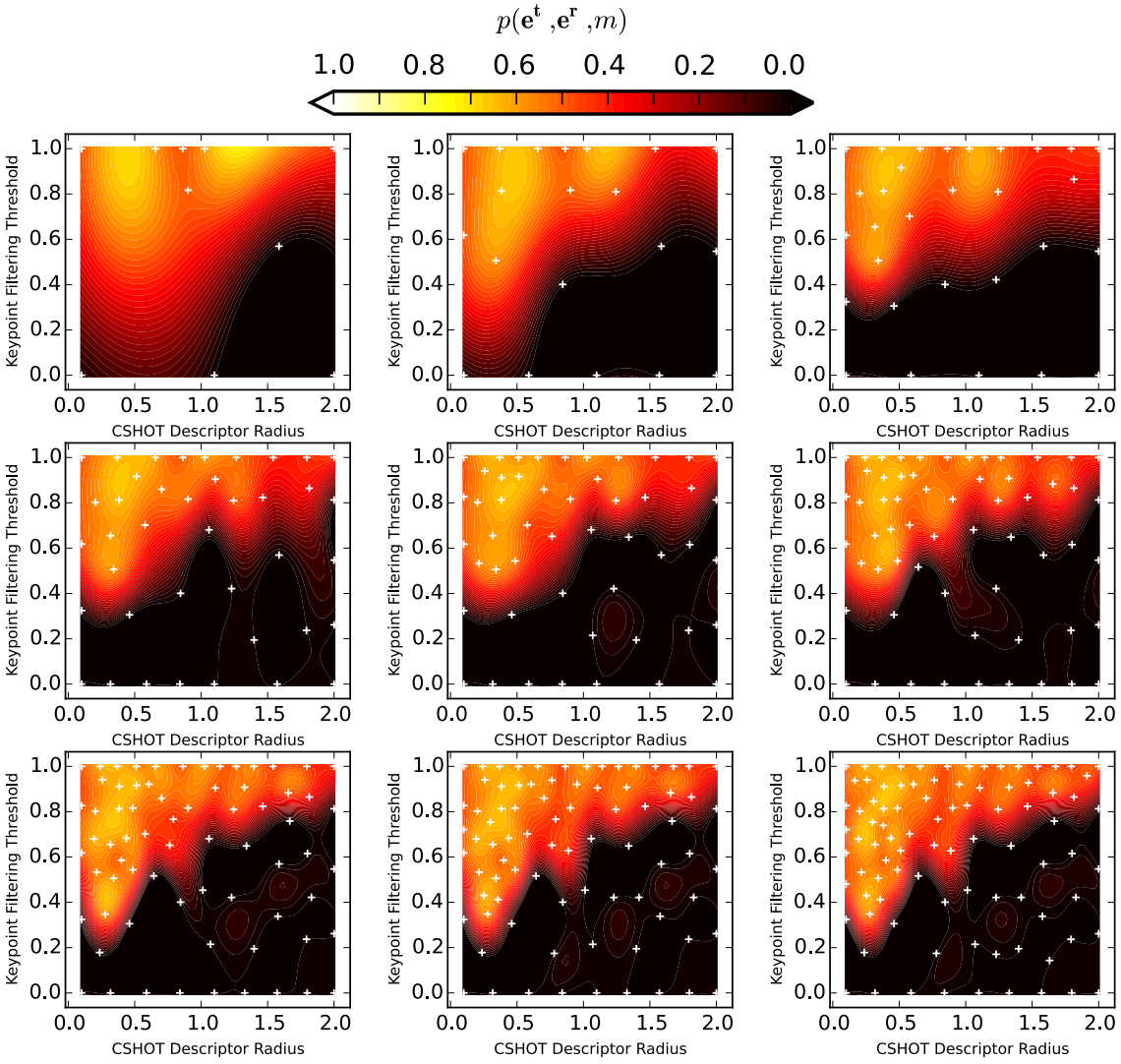


Figure 5.8.: Visualization of how the GP's estimate of the objective function changes as more observations become available. The number of observations (white crosses) increase in steps of 10, from 10 observations (top left) to 90 observations (bottom right).

sets $\mathbf{x_{baseline}}$ and $\mathbf{x'_{1D}}$. However, note all but the two optimized parameters in $\hat{\mathbf{x}}_{\mathbf{2D}}$ are using the same values as $\mathbf{x_{baseline}}$.

5.6. Joint Optimization Experiments

This section contains experiments, during which all map matcher parameters were jointly optimized. This means that these experiments optimize a nine-dimensional design space χ. A time budget of 24 hours was assigned to each optimization run to make the results comparable. After that time frame, the optimization process is stopped and the best parameter set found until then is noted. The purpose of each experiment and which training data was used for the optimization process is described in the respective subsections 5.6.1 and 5.6.2. The plots in those subsections are based on the training data, to see the data from the perspective of the *Optimizer*. However, for

the comparison and discussion in Section 5.7, the resulting parameter on both complete datasets, using the ground truth transformations. The resulting parameter values can be found in Table 5.2.

5.6.1. Using All Submap Pairs

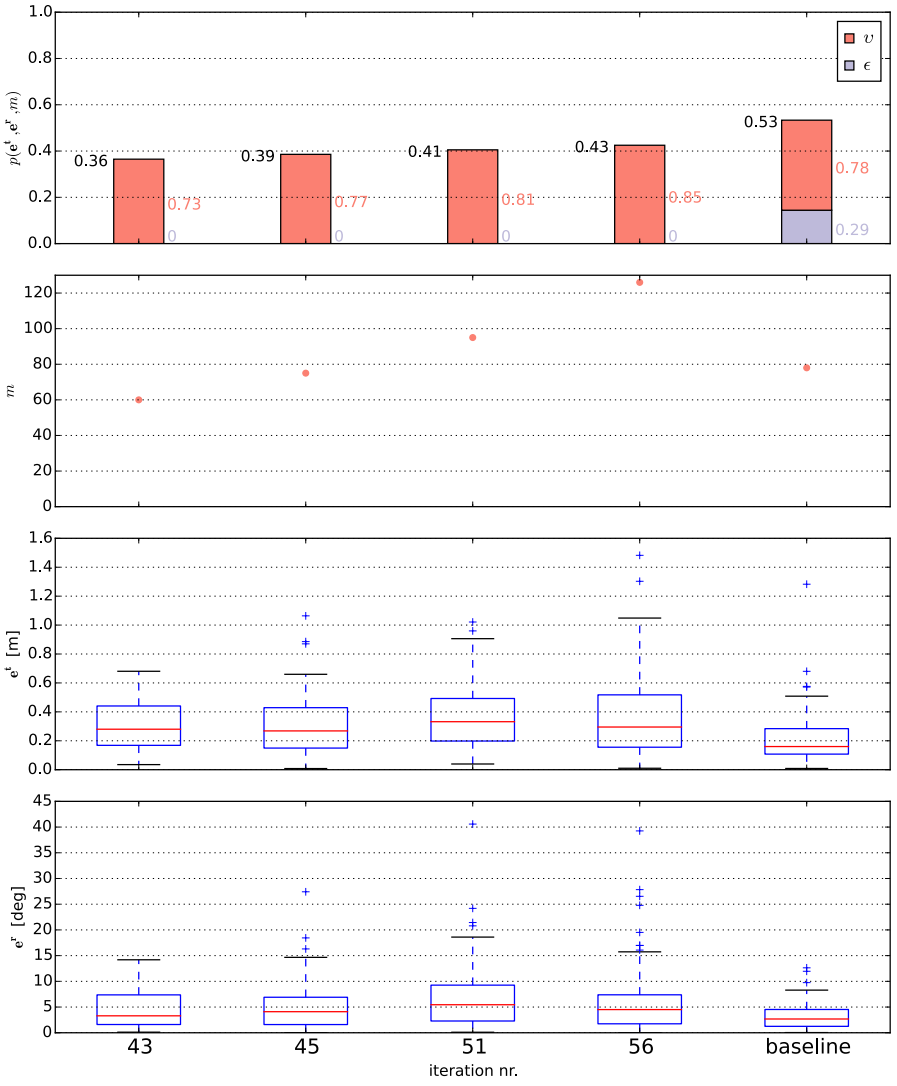


Figure 5.9.: Visualization of the best sets of parameters found during the experiment on the complete lab dataset ($\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9D}}$). An iteration is only visualized if it improves upon the previously known best set of parameters. Additionally, the parameter set $x_{baseline}$ is displayed on the right, to compare the optimizer results against.

During this section's experiments, the complete lab dataset was used for the optimization process. This requires ground truth information to define the correct transformation for each submap pair, so the performance measure can be calculated. The goal of this experiment is to determine whether joint optimization of a design space with nine dimensions is feasible. Additionally, its results could be used as a reference to judge whether reducing the size of the training dataset in further experiments has a negative impact on the resulting parameter sets. This experiment corresponds to step one of the problem statement in Section 1.1. The optimization run yielded

parameter set $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9D}}$.

In this experiment, the optimizer did not manage to find a parameter set that consistently produces good quality matches. In fact, it seems like the optimization process only managed to maximize the number of matches. The comparison to the other parameter sets in Section 5.7 shows that $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9D}}$ is the worst parameter set. Because using all submap pairs takes significantly more time than only using subsequent submaps, the number of observations possible during the 24 hours time frame were much lower. Only 64 observations were made during the given time budget. All other factors that changed from this experiment to the following ones should actually increase the difficulty of the optimization process. Additionally, the first optimization run with only subsequent submaps (see Figure 5.10 at the top) looks relatively similar for the first 64 iterations. In that experiment, the match quality gets optimized only after almost 100 iterations. Therefore, the reason for the bad results in this experiment is likely tied to the lower number of observations. This could be alleviated by giving the experiment a bigger time budget, by reducing the training data size or by making the map matching process more efficient. Another reason for the unsatisfying performance could simply be bad luck, since multiple parts of the optimizer are based on randomness, most prominently, the algorithm that maximizes the acquisition function. Additional runs of the same experiment with a different random number generator seed might therefore yield substantially different results.

5.6.2. Using Only Subsequent Submaps

The experiments of this section aims to show whether the second step of the problem statement in Section 1.1 can be achieved with the proposed solution. The performance measure was parameterized with $\mathrm{m}^* = 60$ and $\mathrm{e}^* = 0.4$.

First, three optimization runs were made on the lab dataset, using only subsequent submaps. The reason for doing multiple runs of exactly the same experimental setup is that the optimization process is a randomized method. This experiment is a good opportunity to check how strongly the proposed solution changes its results due to randomness, since it has a smaller sized training dataset. Therefore, a higher number of observations can be made with the same time budget. The resulting parameter sets are $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssgt}}$, $\hat{\mathbf{x}}^{\mathbf{2}}_{\mathbf{9Dssgt}}$ and $\hat{\mathbf{x}}^{\mathbf{3}}_{\mathbf{9Dssgt}}$, and the optimization process is visualized in Figure 5.10. Additional information for the experiments of this section is visualized Appendix C.

During the optimization process, the ground truth transformations between the subsequent submaps were used to calculate the performance measure. The experiment's goal was to see how the proposed solution fares when the training dataset is restricted to subsequent submaps. This restriction on the training data means less training data is available. More specifically, exactly those submaps are excluded, which a map matcher should match in its final use-case as part of a SLAM system. The results show that the smaller training data set size actually is beneficial. It leads to severely decreased sample generation durations. Since all experiments have the same, fixed time budget of 24 hours, this means that more observations are possible.

When testing the resulting parameter sets on the complete dataset (see Figure 5.12), they seem to have no problem with matching non-subsequent submaps. This means, the local information

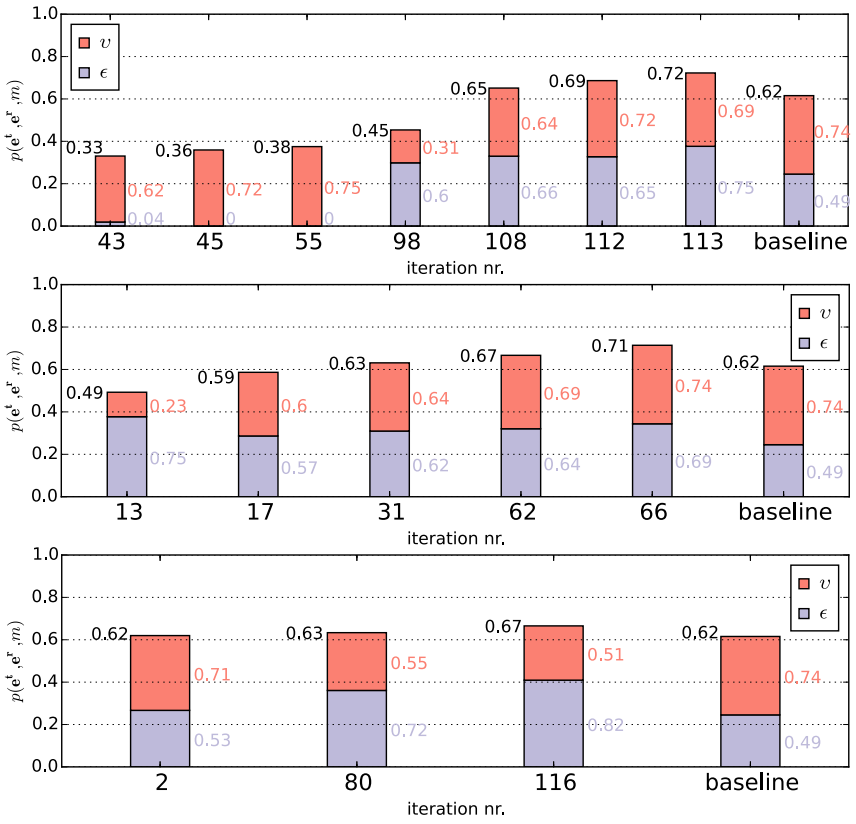


Figure 5.10.: Visualization of the best parameters found during three experiments on subsequent submaps with ground truth transformations and their performance on that training dataset. An iteration is only visualized if it improves upon the previously known best set of parameters. Additionally, the parameter set $x_{baseline}$ is displayed on the right, to compare the optimizer results against. The plots show all three runs from top ($\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssgt}}$) to bottom ($\hat{\mathbf{x}}^{\mathbf{3}}_{\mathbf{9Dssgt}}$).

given by subsequent submap transformations is sufficient to find good solutions for the global problem of matching all kinds of submap pairs. The resulting parameter sets all outperform the hand-tuned reference set $\mathbf{x}_{\mathbf{baseline}}$, according to the performance measure calculated on the training data in Figure 5.10. Also, the evaluation on both the lab dataset and the simulation dataset in Figures 5.12 and 5.13 shows that they also outperform $\mathbf{x}_{\mathbf{baseline}}$ on the complete datasets. This indicates that a parameter set ranking induced by performance measure values, which have been calculated using only subsequent submaps, tends to hold true for complete dataset as well. This is an important property, because it allows choosing a parameter set without having ground truth data to evaluate them on.

A second set of two optimization runs was performed, again using only subsequent submaps. However, this time, the transformations were given by the incremental localization solution of the LRU, instead. The resulting parameter sets are $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssi}}$ and $\hat{\mathbf{x}}^{\mathbf{2}}_{\mathbf{9Dssi}}$, and the optimization process is visualized in Figure 5.11.

By using the transformations from the incremental localization solution, the optimization process is completely independent from ground truth information. This would enable the solution to solve the second step of the problem statement in Section 1.1. However, those transformations contain the errors of the incremental localization system. Therefore, the performance measure, on

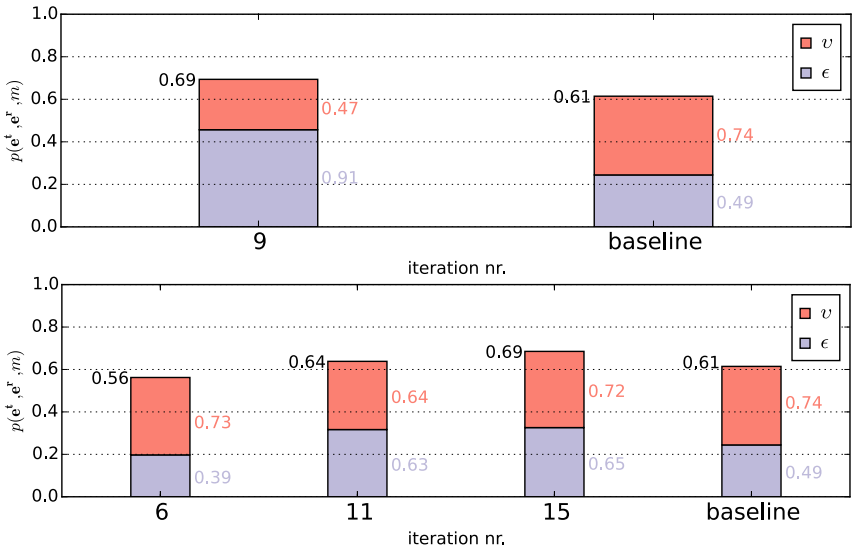


Figure 5.11.: Visualization of the best parameters found during two experiments on subsequent submaps with incremental localization transformations and their performance on that training dataset. An iteration is only visualized if it improves upon the previously known best set of parameters. Additionally, the parameter set $x_{baseline}$ is displayed on the right, to compare the optimizer results against. The plots show both runs, $\hat{\mathbf{x}}^1_{\mathbf{9Dssi}}$ at the top and $\hat{\mathbf{x}}^2_{\mathbf{9Dssi}}$ at the bottom.

which the optimization process is based on, also contains those errors. This may lead to matches that are considered to be closer to a perfect match than they are, or the other way around. But since the error of the incremental localization transformations is bounded to 10cm, the error of the performance measure is bounded as well (see Section 4.3).

Except the use of incremental localization transformations instead of ground truth transformations, the experimental setup was the same as the experiment above. However, the system's behavior looks quite different. For some reason, new, best parameter sets are only found during the first few iterations. The time budget was exactly the same and the both experiments ran for over 80 iterations. Slight differences in the number of iterations can be due to where the *Optimizer* module requests new observations, since the sample generation duration depends on the choice of the map matcher parameters. As with all experiments, the different behavior might have been caused by randomness. More experiments are required, to determine whether that is the case. Another reason could be the quality of the incremental localization transformation. This problem can be alleviated by basing the decision about which (subsequent) submap pairs to add to the training dataset on the uncertainty of the filter that estimates the transformations. Additionally, this could allow using submap pairs that are not direct neighbors as training data, as long as the filter's uncertainty is sufficiently low. While those non-subsequent submap pairs are unlikely to overlap, they can be used to test the map matcher against false positives during the optimization process. However, this goes beyond the scope of this thesis, but should be considered for future work.

Still, when considering the comparison on the complete datasets in Figures 5.12 and 5.13, the resulting parameter sets are only slightly worse than the hand-tuned parameter set $\mathbf{x}_{\mathbf{baseline}}$. This means that the proposed solution is viable to solve the second step of the problem statement, but can still be improved to yield even better results.

5.7. Discussion and Results

Parameter Set	p_{lab}	p_{sim}	r_N	v_K	t_K	r_D	n_C	t_C	r_{LRF}	v_{TF}	n_{TF}
$\mathbf{x_{baseline}}$	0.53	0.7	0.2	0.1	1.0	0.2	3	0.5	0.2	0.5	8
$\mathbf{x_{1D}}$	0	0	0.65	0.05	0.4	1.2	1	0.6	0.3	0.3	123
$\mathbf{x'_{1D}}$	0.64	0.74	0.65	0.05	1.0	1.2	1	0.6	0.3	0.3	123
$\hat{\mathbf{x}}_{\mathbf{2D}}$	0.67	0.73	0.2	0.1	0.95	0.34	3	0.5	0.2	0.5	8
$\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9D}}$	0.43	0.49	0.1	0.05	1.0	2.0	1	1.0	0.1	1.0	118
$\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssgt}}$	0.71	0.77	0.1	0.05	1.0	2.0	10	1.0	0.1	0.1	63
$\hat{\mathbf{x}}^{\mathbf{2}}_{\mathbf{9Dssgt}}$	0.58	0.65	0.1	0.05	1.0	2.0	10	1.0	1.0	0.1	41
$\hat{\mathbf{x}}^{\mathbf{3}}_{\mathbf{9Dssgt}}$	0.59	0.62	2.0	0.05	0.0	0.1	10	1.0	1.0	0.1	138
$\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssi}}$	0.51	0.68	1.77	0.15	0.99	0.79	1	0.73	0.49	0.63	98
$\hat{\mathbf{x}}^{\mathbf{2}}_{\mathbf{9Dssi}}$	0.55	0.65	2.0	0.05	1.0	2.0	10	1.0	1.0	0.1	72

Table 5.2.: This table summarizes all relevant parameter sets of this evaluation. p_{lab} is the performance measure on the lab dataset. p_{sim} is the performance measure on the simulation dataset. The other columns contain the respective map matcher parameter values, see Section 5.2.

This section compares all relevant sets of map matcher parameters of this evaluation. Table 5.2 contains all parameter sets, including the three sets determined by hand, $\mathbf{x_{baseline}}$ (see Section 5.3), $\mathbf{x_{1D}}$ and $\mathbf{x'_{1D}}$ (see Section 5.4). The parameter set $\hat{\mathbf{x}}_{\mathbf{2D}}$ was generated by optimizing on a two-dimensional design space χ in Section 5.5. The parameter set $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9D}}$ was generated by jointly optimizing all nine map matcher parameters on the lab dataset in Section 5.6.1. This corresponds to the first step of the problem statement in Section 1.1. In Section 5.6.2, $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssgt}}$, $\hat{\mathbf{x}}^{\mathbf{2}}_{\mathbf{9Dssgt}}$ and $\hat{\mathbf{x}}^{\mathbf{3}}_{\mathbf{9Dssgt}}$ were generated by only using subsequent submaps of the lab dataset. The section also contains the experiments that yielded $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssi}}$, $\hat{\mathbf{x}}^{\mathbf{2}}_{\mathbf{9Dssi}}$, where the transformation between the subsequent submap pairs were given by the incremental localization solution. This corresponds to the second step of the problem statement in Section 1.1.

The performance of the different parameter sets on the lab and simulation dataset was evaluated in Figures 5.12 and 5.13, using all submap pairs and their ground truth transformations. Note that some parameter sets, like $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9D}}$, were determined by using the complete lab dataset as training data. Others, like $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssgt}}$, $\hat{\mathbf{x}}^{\mathbf{2}}_{\mathbf{9Dssgt}}$ and $\hat{\mathbf{x}}^{\mathbf{3}}_{\mathbf{9Dssgt}}$, only used a subset of it. Still, even having only parts of the training dataset in the evaluation dataset may benefit overfitting. However, the simulation dataset was not used as training data in any experiment, making it suitable for checking against overfitting. Additionally, the parameter sets $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssi}}$ and $\hat{\mathbf{x}}^{\mathbf{2}}_{\mathbf{9Dssi}}$, which are arguably the most important ones for practical use of the proposed solution, didn't use any of the lab dataset's ground truth transformations.

The single parameter experiments (see Section 5.4 and Appendix A) indicated that the map matcher requires similar parameter sets for both datasets. Since all evaluated parameter sets perform in a similar way on both datasets, this seems to hold true. While the way the performance measure is calculated leads to different scores on different datasets for the same parameter set, when ranking the parameter sets according to the performance measure, their order tends to remain the same on both datasets. Only parameter sets that perform very similarly even on their

training dataset, like $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssi}}$ and $\hat{\mathbf{x}}^{\mathbf{2}}_{\mathbf{9Dssi}}$, or $\hat{\mathbf{x}}^{\mathbf{2}}_{\mathbf{9Dssgt}}$ and $\hat{\mathbf{x}}^{\mathbf{3}}_{\mathbf{9Dssgt}}$, have another order in the ranking on the simulation dataset than the ranking on the lab dataset.

The fact that $\hat{\mathbf{x}}_{\mathbf{1D}}$ does not yield any matches due to keypoint filtering shows, that separately optimizing map matcher parameters can lead to bad results. The parameters CSHOT Descriptor Radius and Keypoint Filtering Threshold may be an extreme example, but some form of dependency exists between almost all parameters. Therefore, joint optimization is necessary for finding optimal parameter values. On the other hand, joint optimization means that the search space is very high dimensional and a higher number of observations are necessary to cover it sufficiently. That too little observations deteriorates the optimization results can be seen with $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9D}}$ from Section 5.6.1.

A promising topic for future work would be developing a method to determine subsets of χ, which only contain parameters independent from each other. Then, two or more separate optimization processes can be executed, each optimizing a part of χ independently. This would severely decrease the size of the search space from which observations are drawn. Therefore, the optimization process would require less observations for the same level of quality. Of course, the size of the design space can also be reduced by designing the map matcher such that it requires less parameters.

The experiment that yielded $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9D}}$ shows, that the proposed solution may need more than 24 hours to find a good set of parameters, depending on the size of the training dataset. Note, that this optimization process used exactly the same data as training data, as was used to judge the resulting parameter set's performance in Figure 5.12.

The three parameter sets $\hat{\mathbf{x}}_{\mathbf{9Dssgt}}$ used a smaller training dataset and, instead of performing worse, they actually yielded very good parameter sets. All of them outperform the hand-tuned parameters $\mathbf{x}_{\mathbf{baseline}}$. The best performance is achieved by $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssgt}}$, which yields only slightly less matches while greatly improving their quality on the lab dataset (see Figure 5.12). On the simulation dataset (see Figure 5.13), the parameter set is superior to $\mathbf{x}_{\mathbf{baseline}}$ w.r.t. both match quantity and match quality.

The two parameter sets $\hat{\mathbf{x}}_{\mathbf{9Dssi}}$ were trained with incremental localization information instead of ground truth data. This causes a noticeable drop in performance on the lab dataset, but the quality of the resulting parameter sets can still compete with the hand-tuned parameter set $\mathbf{x}_{\mathbf{baseline}}$. The precise reason for that difference remains to be determined in future experiments. It could either be caused by variations due to randomness during the optimization process or by particularly erroneous transformations from the incremental localization solution.

The evaluation clearly shows that the proposed solution is capable of solving the second step of the problem statement in Section 1.1. However, the results could be further improved by several changes discussed in Section 6.2.

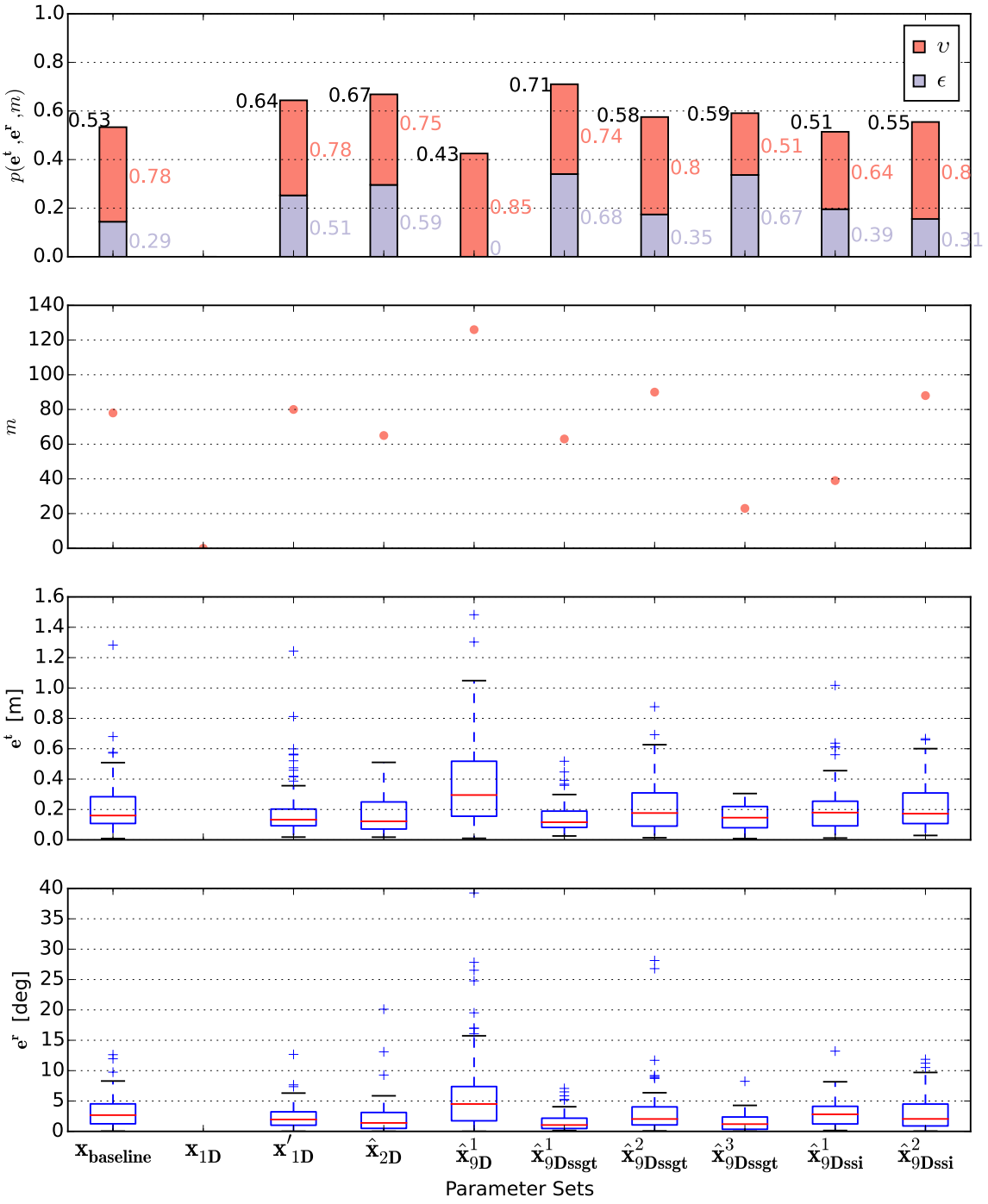


Figure 5.12.: Visualization of the performance of the different parameter sets from the evaluation chapter on the lab dataset. The performance measure p was parameterized with $\mathrm{m}^* = 200$ and $\mathrm{e}^* = 0.4$.

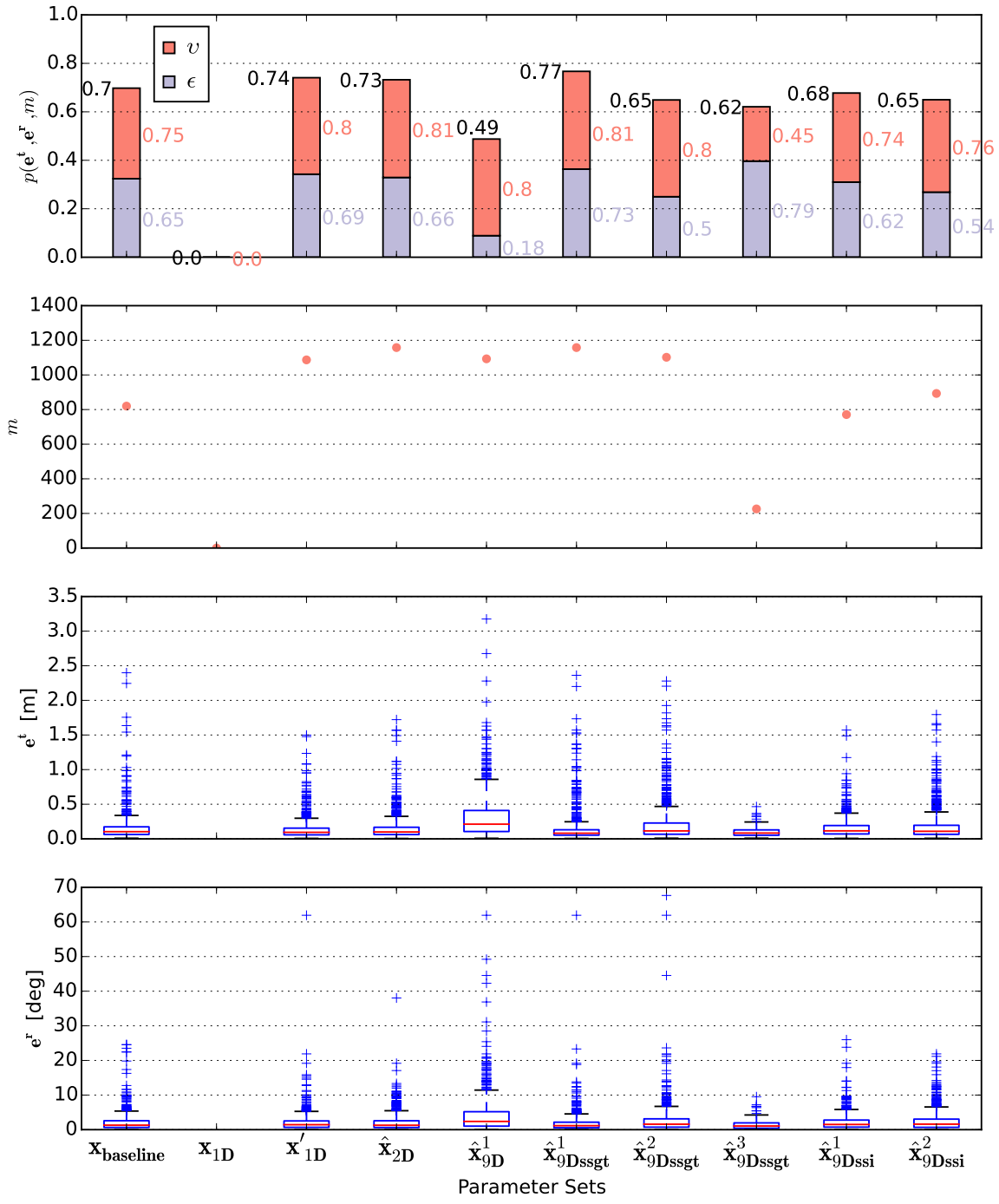


Figure 5.13.: Visualization of the performance of the different parameter sets from the evaluation chapter on the simulation dataset. The performance measure p was parameterized with $\mathrm{m}^* = 2500$ and $\mathrm{e}^* = 0.4$.

6. Conclusion and Future Work

In this thesis, a system was designed to automatically find good parameter sets for a feature-based map matcher. The evaluation shows that the proposed solution suffices for finding parameters with similar performance than parameters that were hand-tuned for the evaluated datasets. Since this can be done by only using subsequent submap pairs with pseudo ground truth transformations given by the incremental localization system, the proposed solution is capable of solving the second step of the problem statement in Section 1.1. Section 6.1 contains a summary of the evaluation results, highlighting the main conclusions about what can already be done with the current system and what its weaknesses are. In Section 6.2, the most significant ways to improve the proposed solution are described. It also includes promising directions for future research, to move towards an online solution for solving the third stage of the problem statement.

6.1. Conclusion

The experiments in Chapter 5 have shown that the proposed solution is capable of automatically finding good map matcher parameters within 24 hours, when optimizing on four desktop PCs in parallel. However, further experiments would lead to a better understanding about how many observations are necessary to get stable results. The experiments further show that the information provided by a robot's incremental localization solution is sufficient for determining good parameter sets. However, this introduces small errors to the performance measure, which leads to slightly worse results, compared to using ground truth transformations.

An important property of the proposed solution is that the amount of work required to generate training data is relatively small. The robot system simply has to move around its environment, and record submap pairs with transformations given by its incremental localization solution. No complicated tracking system needs to be set up to generate a ground truth trajectory. Since the third part of the problem statement in Section 1.1 was not solved in this thesis, the proposed solution can not be used on the robot itself to immediately adapt its parameters to new environments. However, with the current solution, a robot system can gather data in a new, completely unknown environment and send it to a computer cluster to do the optimization off-board. After a period of time, a working set of parameters for this environment should be available to the robot. This may already suffice for some use-cases, like planetary exploration, in which the robot can easily pause its mission for some time. Additionally, it severely decreases the necessary amount of manual work to make a robot system ready for autonomous navigation in a new environment or after changes to parts its system related to navigation, e.g. the sensor configuration or map matcher software components.

While the proposed solution saves a lot of time by automatically providing good sets of param-

eters, it itself requires time to set up on a robot system. This includes both the integration of the optimizer software with the robot's mapping system and finding a set of hyperparameters. In the experiments made during this thesis, the optimization process seemed to be relatively insensitive to changes of most of the hyperparameters. Thus, for practical use, it might be sufficient to leave many of them at their default values. But at least the design space χ needs to be defined by an expert who knows all the parameters of the map matcher. The choice of χ has a huge impact on the performance of the system, not only because it determines the size of the search space, but also because the parameter bounds determines how long the sample generation process takes. Especially huge search radii for normal or feature estimation will slow down the process greatly. However, choosing χ only needs to be done once for a map matching pipeline and multiple, different robot systems can use this definition for optimization on any number of different environments.

6.2. Future Work

To make the proposed solution useful to the robotics community, the integration of the optimizer on a new robot system needs to be as simple as possible. One way to achieve this would be to use a common robot framework like ROS. Since many robot system already use ROS, it is likely that a ROS-based implementation could directly interface with the existing mapping software. While this is mainly an engineering challenge, it will make further research in this area that much easier. Additionally, many components of a complex robot system could benefit from automated parameter tuning. Since the proposed method copes well with a high-dimensional search space and long evaluation durations, its implementation could be adapted for tuning parameters of other components as well. For example the proposed method in [22] uses a similar Bayesian optimization approach to optimize the parameters of visual and laser odometry systems. This suggests that by having a good, modularly designed software framework, the same Bayesian optimization code could be used to solve both problems. However, for each type of problem, an information source suitable for generating a training signal, like incremental localization information in this thesis, needs to be found and leveraged.

There are several possibilities to further improve the current approach. Using an acquisition function with dynamic β would better utilize the available time budget and decrease the probability of unsatisfying optimization results like $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9D}}$ (see Section 5.6.1). Also, by directly incorporating the uncertainty of the incremental localization methods when choosing submap pairs for the training dataset, the training dataset's quality could be improved. Further, removing the need for the performance measure's hyperparameter m^* would make the proposed solution easier to use. This could be done by a heuristic that sets it according to the size of the training dataset, automatically. Additionally, the duration of the matching process could also be made part of the performance measure. This would allow users to explicitly control how time efficient the map matcher needs to be, which depends on the computational power of the system that will perform map matching during missions.

A possible method to move towards solving the third step of the problem statement could be adding a second layer of intelligence, which is able to make quick decisions, on top of the current

approach. This layer could utilize the information from many different offline optimization processes. When enough parameter sets were learned previously, the problem gets reduced to simply choosing the best one for the current environment. However, since the online solution only adds value when optimizing for a new environment, it may be more productive to focus on improving parts of the map matching pipeline, instead. Research based on methods from Section 2.3 could lead to map matching pipelines that do not require different parameters for different environments.

A. Single Parameter Experiments

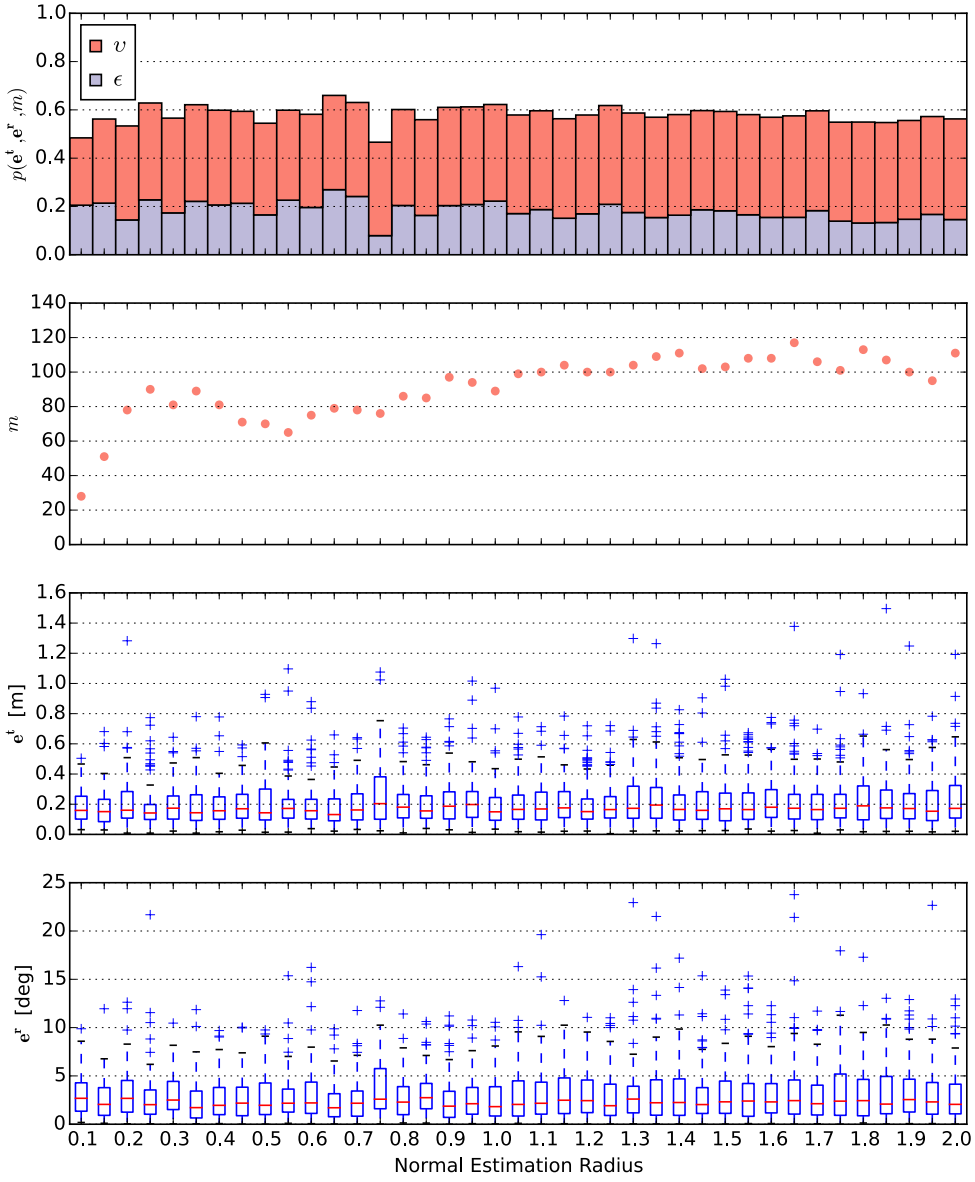


Figure A.1.: Visualization of how the Normal Estimation Radius parameter changes the map matcher performance on the complete lab dataset. The performance measure p was parameterized with $m^* = 200$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

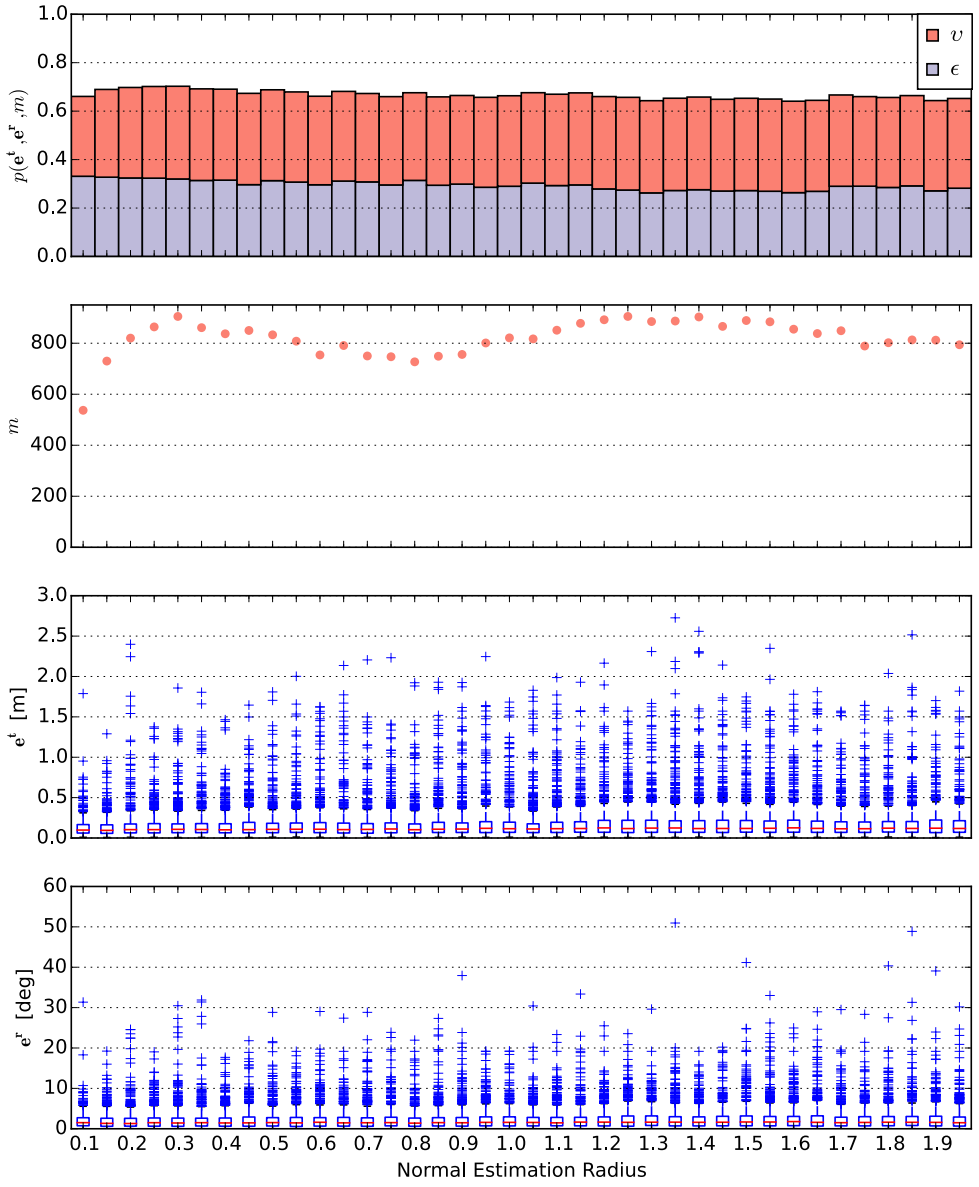


Figure A.2.: Visualization of how the Normal Estimation Radius parameter changes the map matcher performance on the complete simulation dataset. The performance measure p was parameterized with $m^* = 2500$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

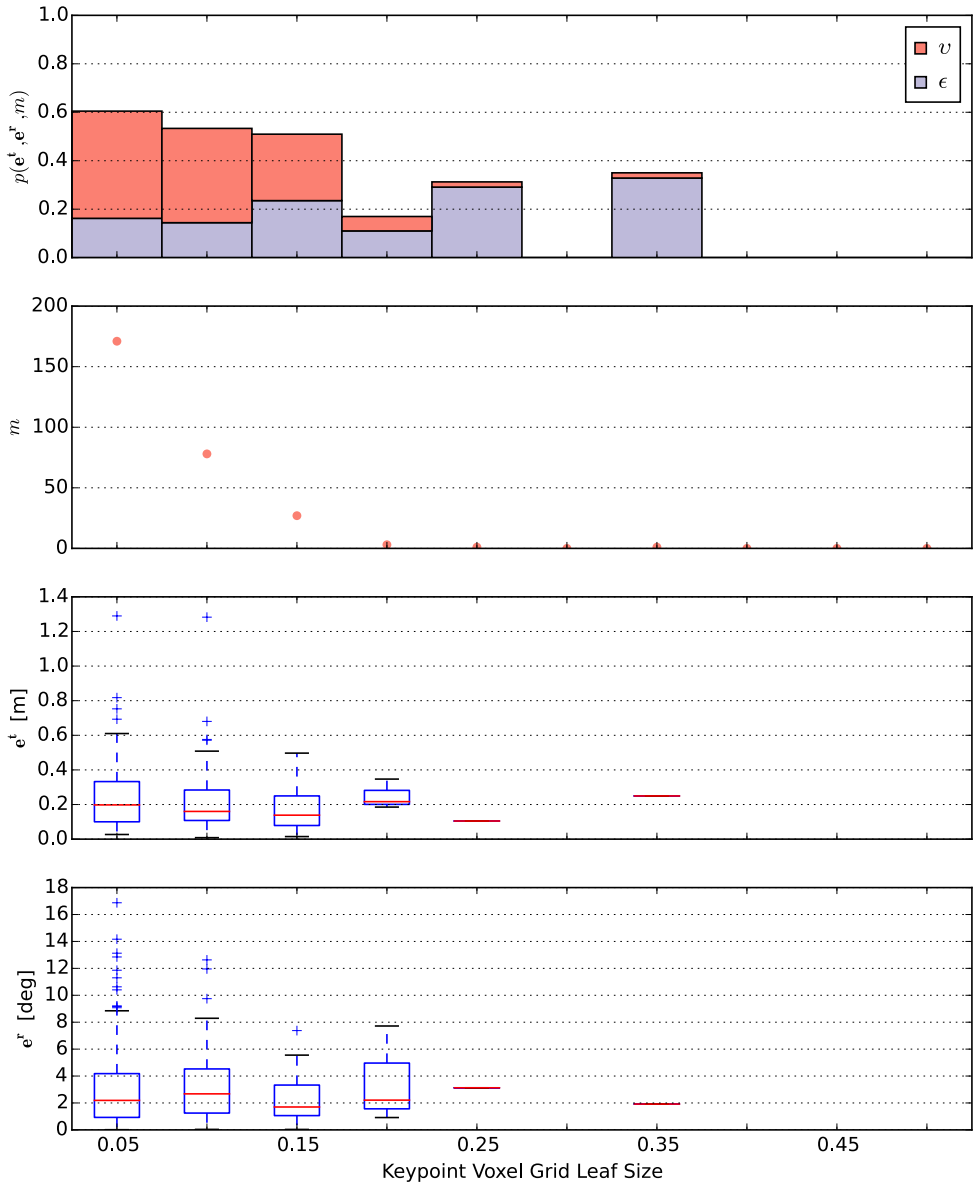


Figure A.3.: Visualization of how the Keypoint Voxel Grid Leaf Size parameter changes the map matcher performance on the complete lab dataset. The performance measure p was parameterized with $m^* = 200$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

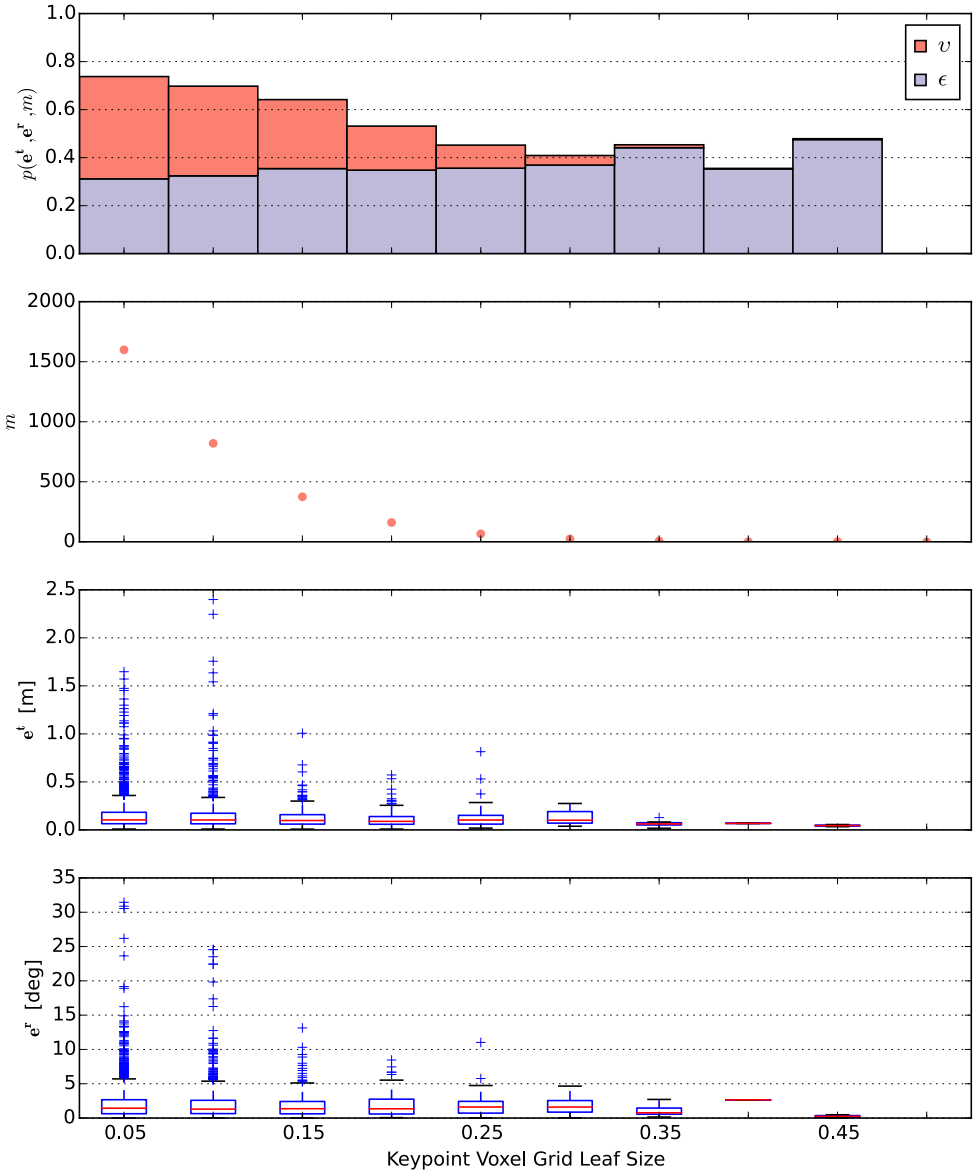


Figure A.4.: Visualization of how the Keypoint Voxel Grid Leaf Size parameter changes the map matcher performance on the complete simulation dataset. The performance measure p was parameterized with $m^* = 2500$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

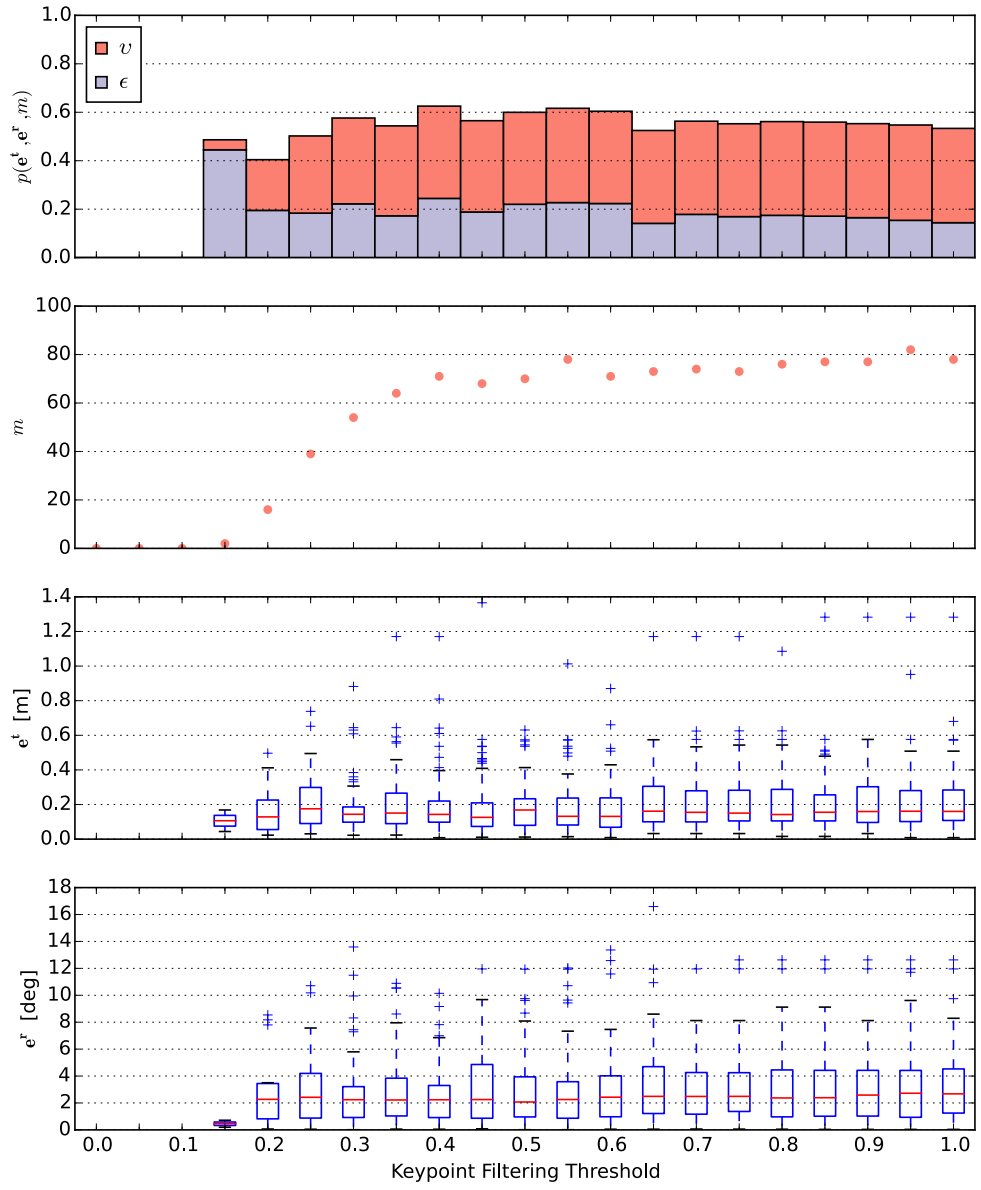


Figure A.5.: Visualization of how the Keypoint Filtering Threshold parameter changes the map matcher performance on the complete lab dataset. The performance measure p was parameterized with $m^* = 200$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

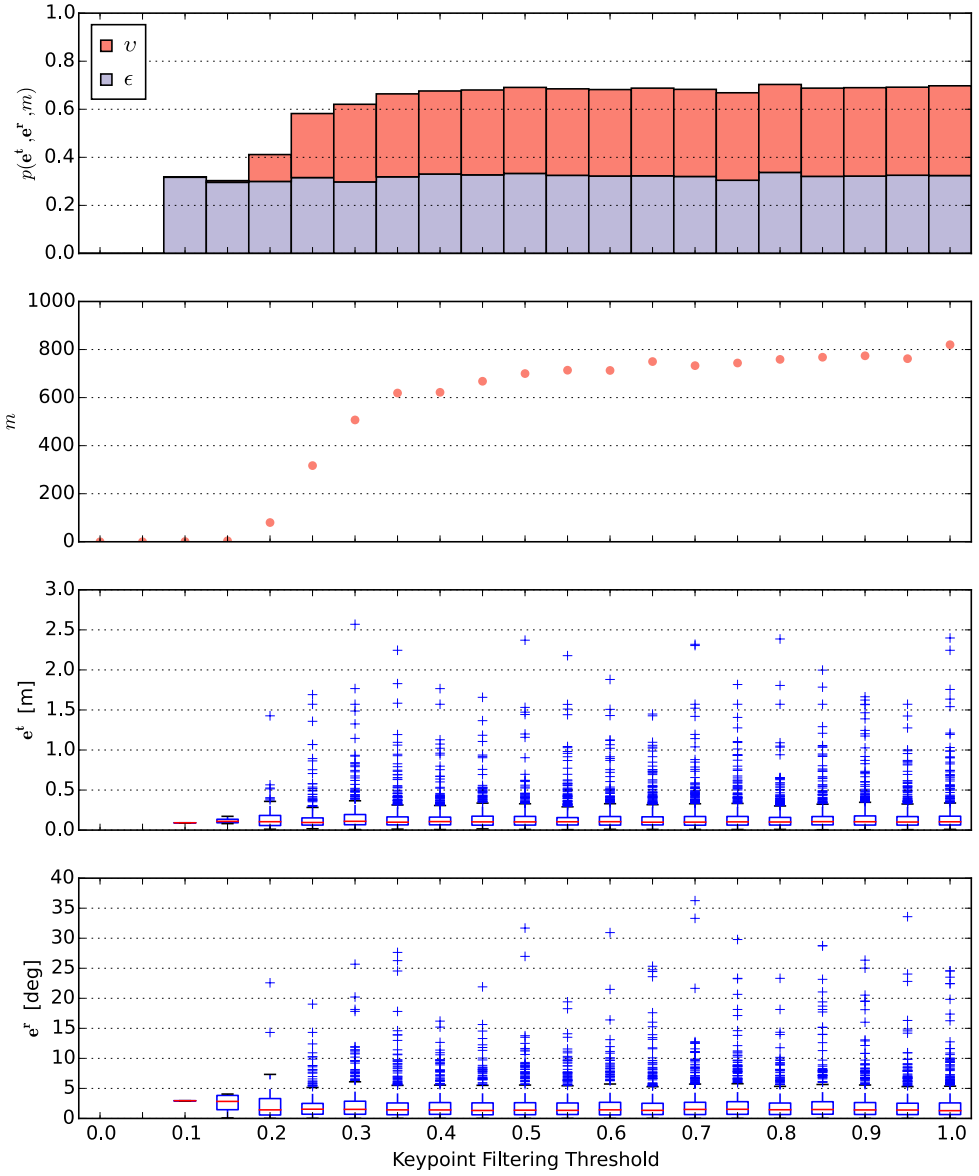


Figure A.6.: Visualization of how the Keypoint Filtering Threshold parameter changes the map matcher performance on the complete simulation dataset. The performance measure p was parameterized with $m^* = 2500$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

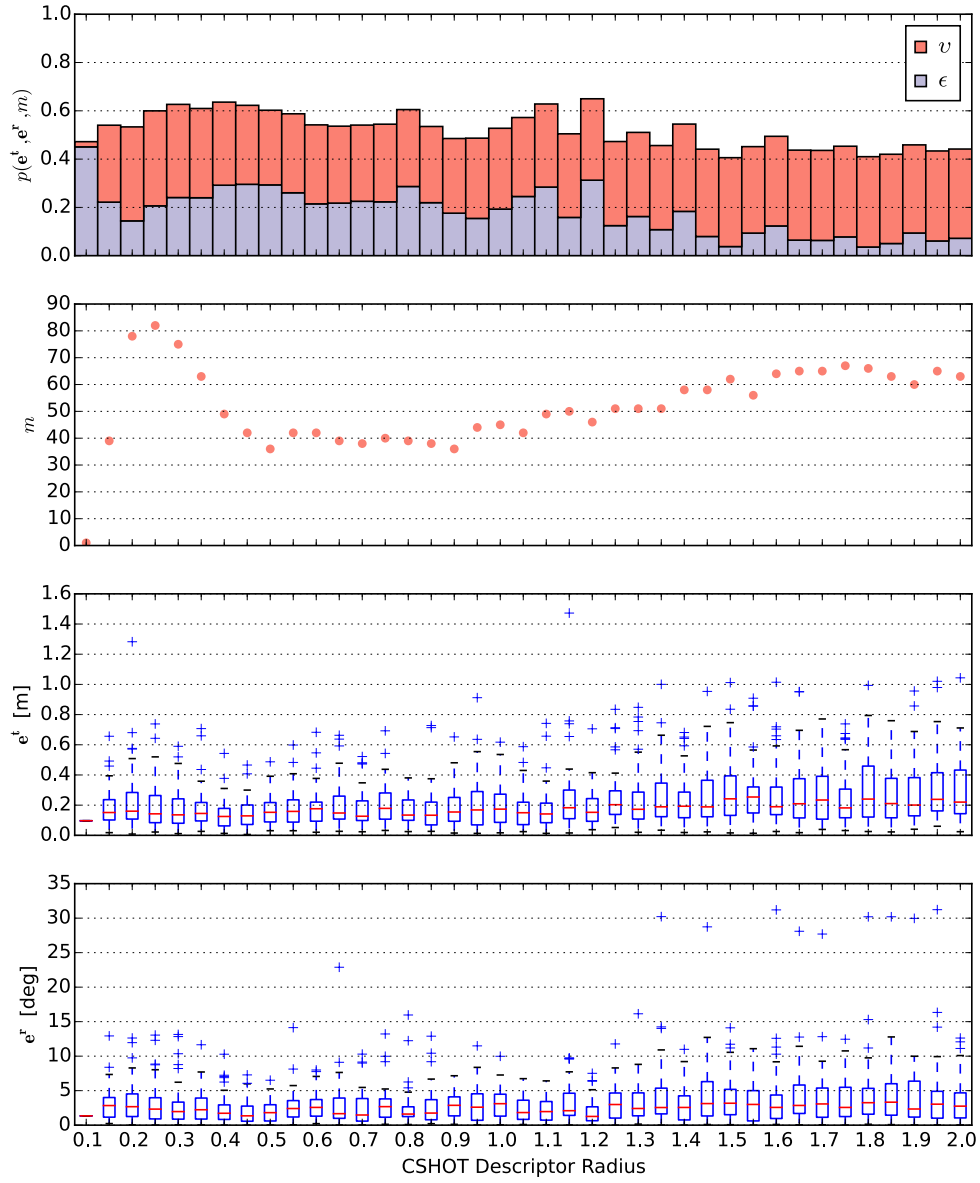


Figure A.7.: Visualization of how the CSHOT Descriptor Radius parameter changes the map matcher performance on the complete lab dataset. The performance measure p was parameterized with $m^* = 200$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

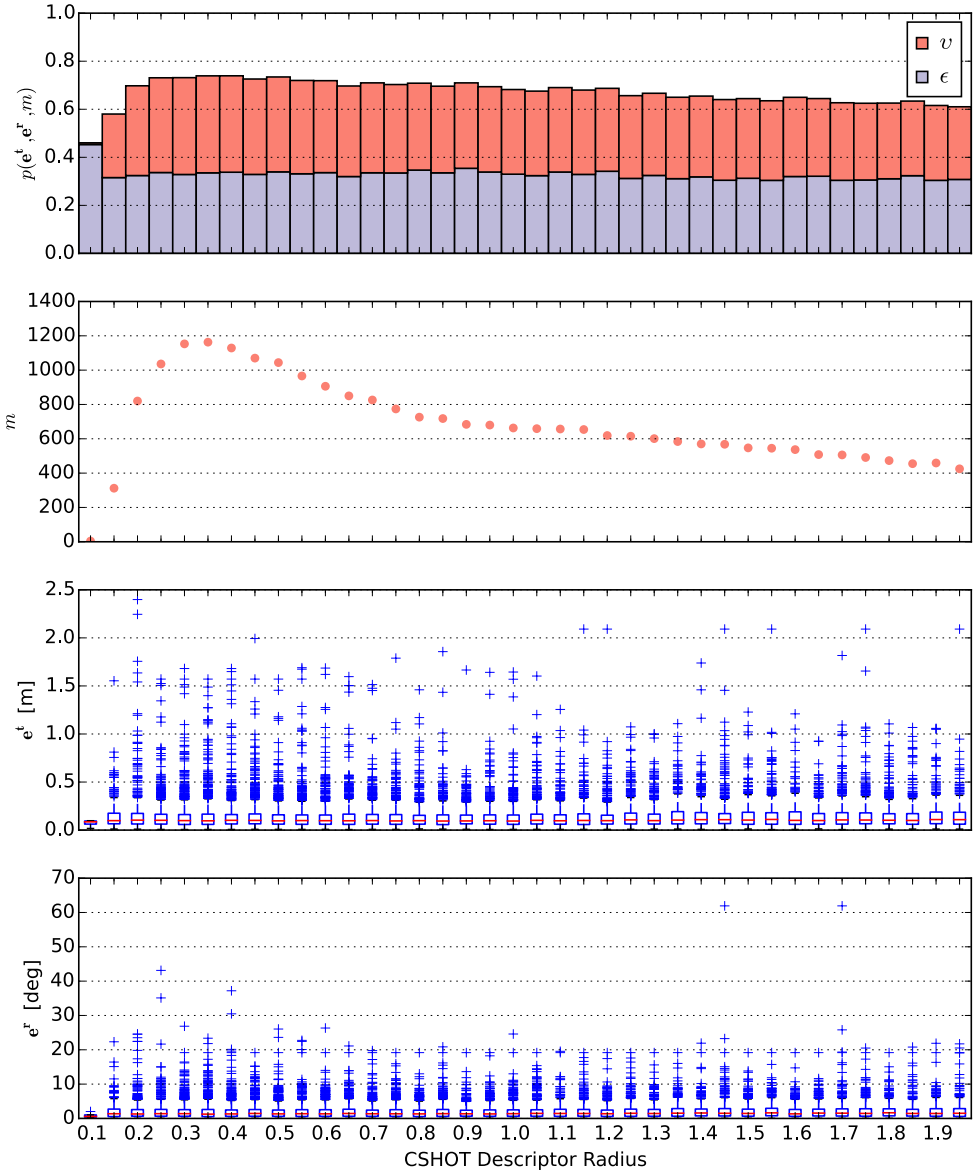


Figure A.8.: Visualization of how the CSHOT Descriptor Radius parameter changes the map matcher performance on the complete simulation dataset. The performance measure p was parameterized with $m^* = 2500$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

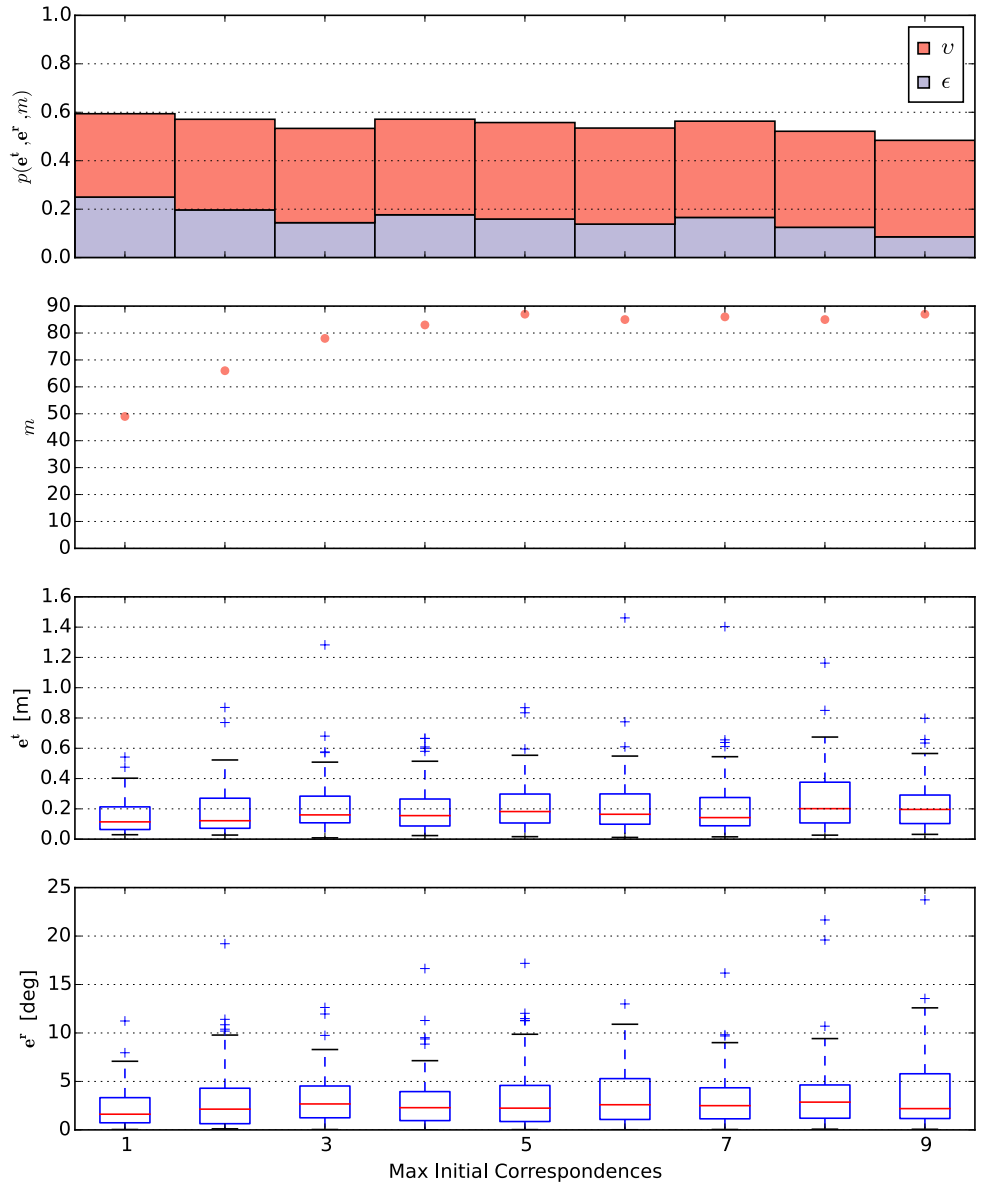


Figure A.9.: Visualization of how the Max Initial Correspondences parameter changes the map matcher performance on the complete lab dataset. The performance measure p was parameterized with $m^* = 200$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

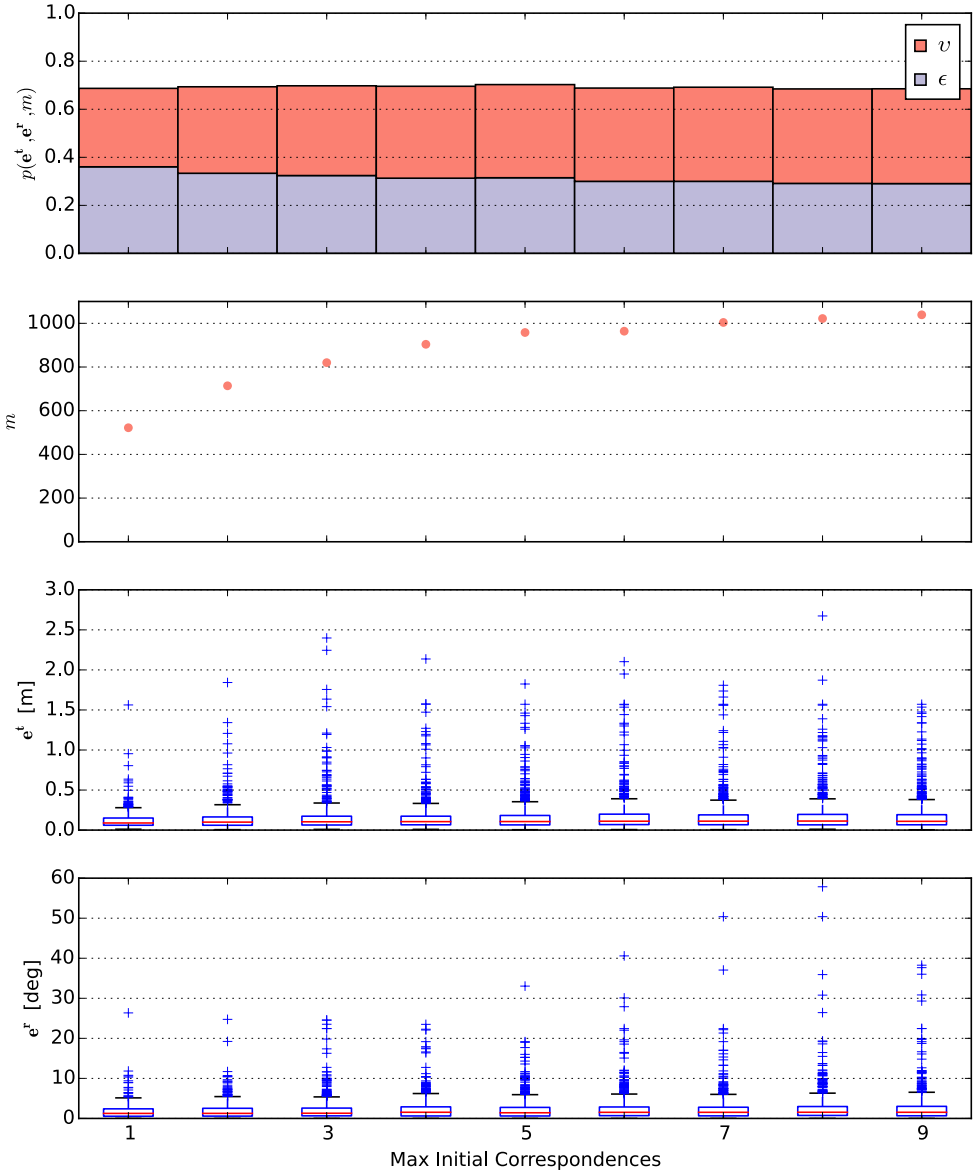


Figure A.10.: Visualization of how the Max Initial Correspondences parameter changes the map matcher performance on the complete simulation dataset. The performance measure p was parameterized with $m^* = 2500$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

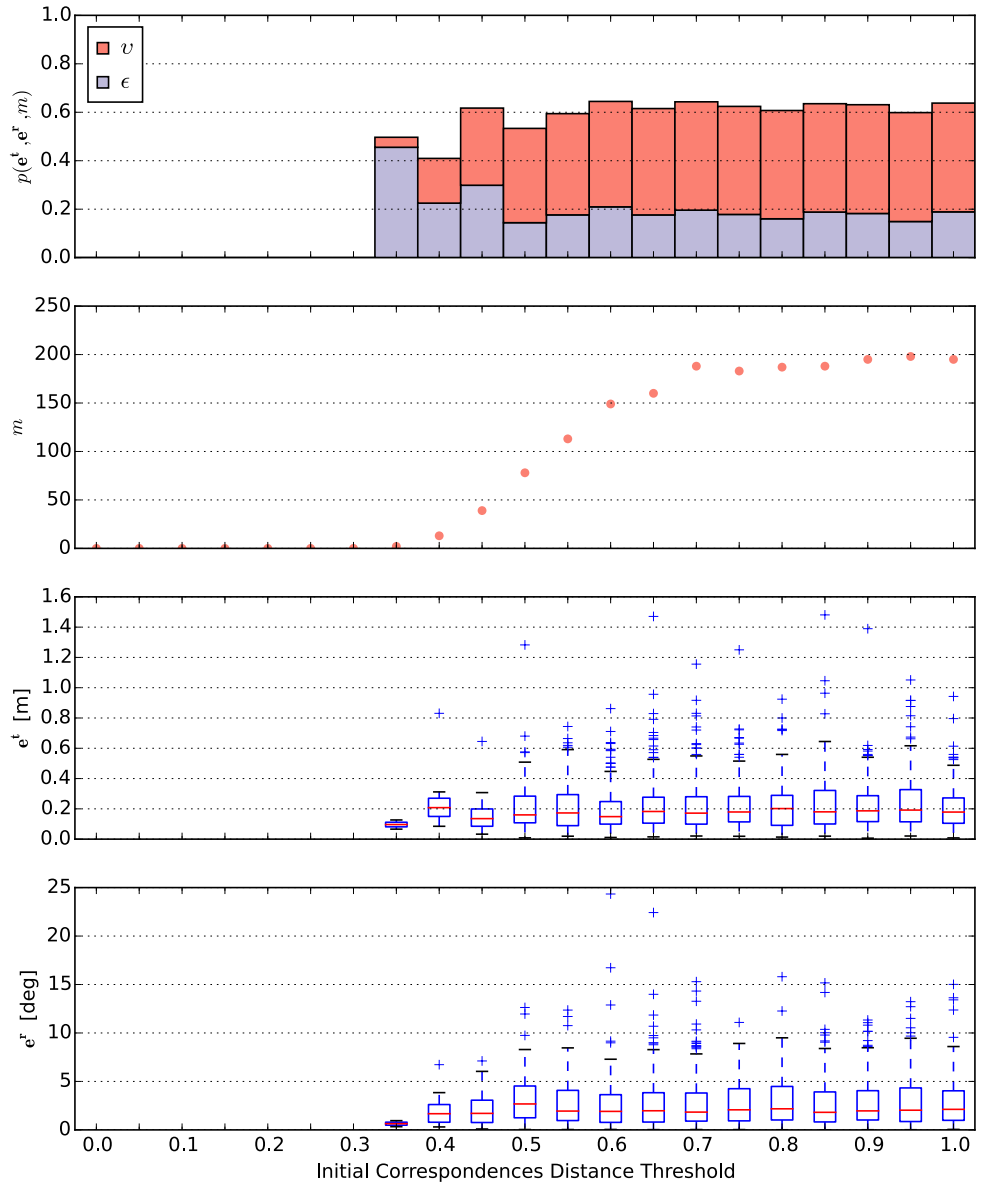


Figure A.11.: Visualization of how the Initial Correspondences Distance Threshold parameter changes the map matcher performance on the complete lab dataset. The performance measure p was parameterized with $m^* = 200$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

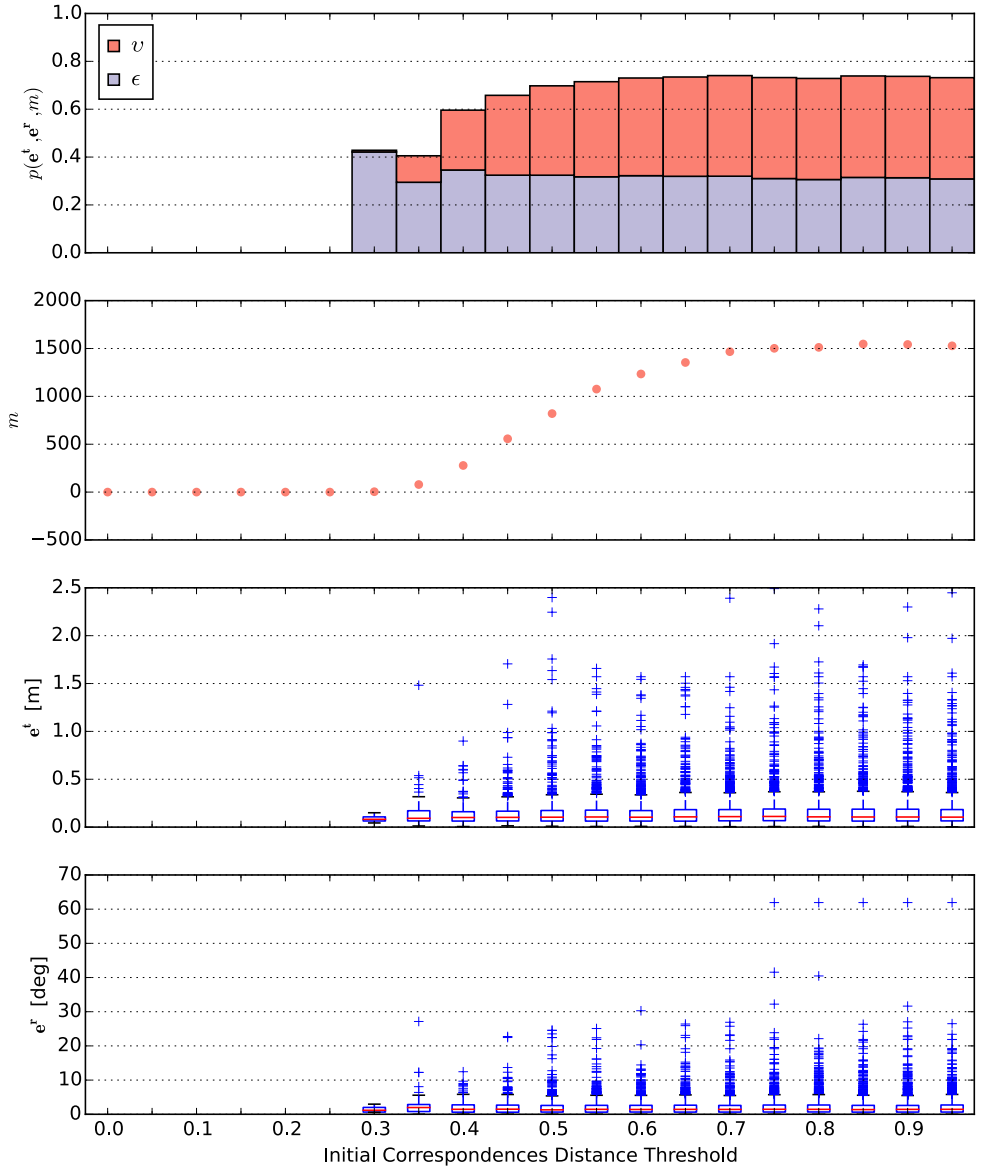


Figure A.12.: Visualization of how the Initial Correspondences Distance Threshold parameter changes the map matcher performance on the complete simulation dataset. The performance measure p was parameterized with $m^* = 2500$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

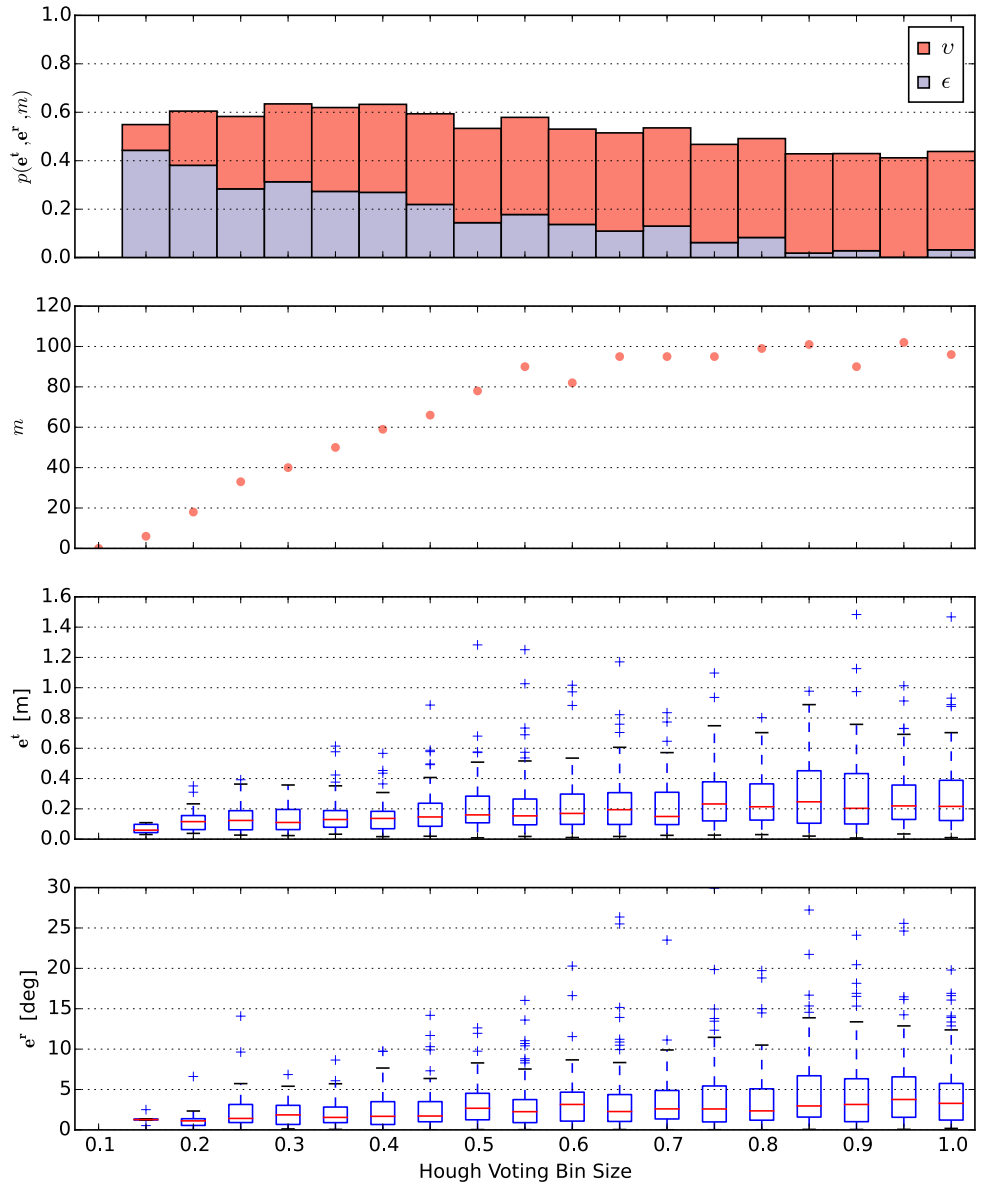


Figure A.13.: Visualization of how the Hough Voting Bin Size parameter changes the map matcher performance on the complete lab dataset. The performance measure p was parameterized with $m^* = 200$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

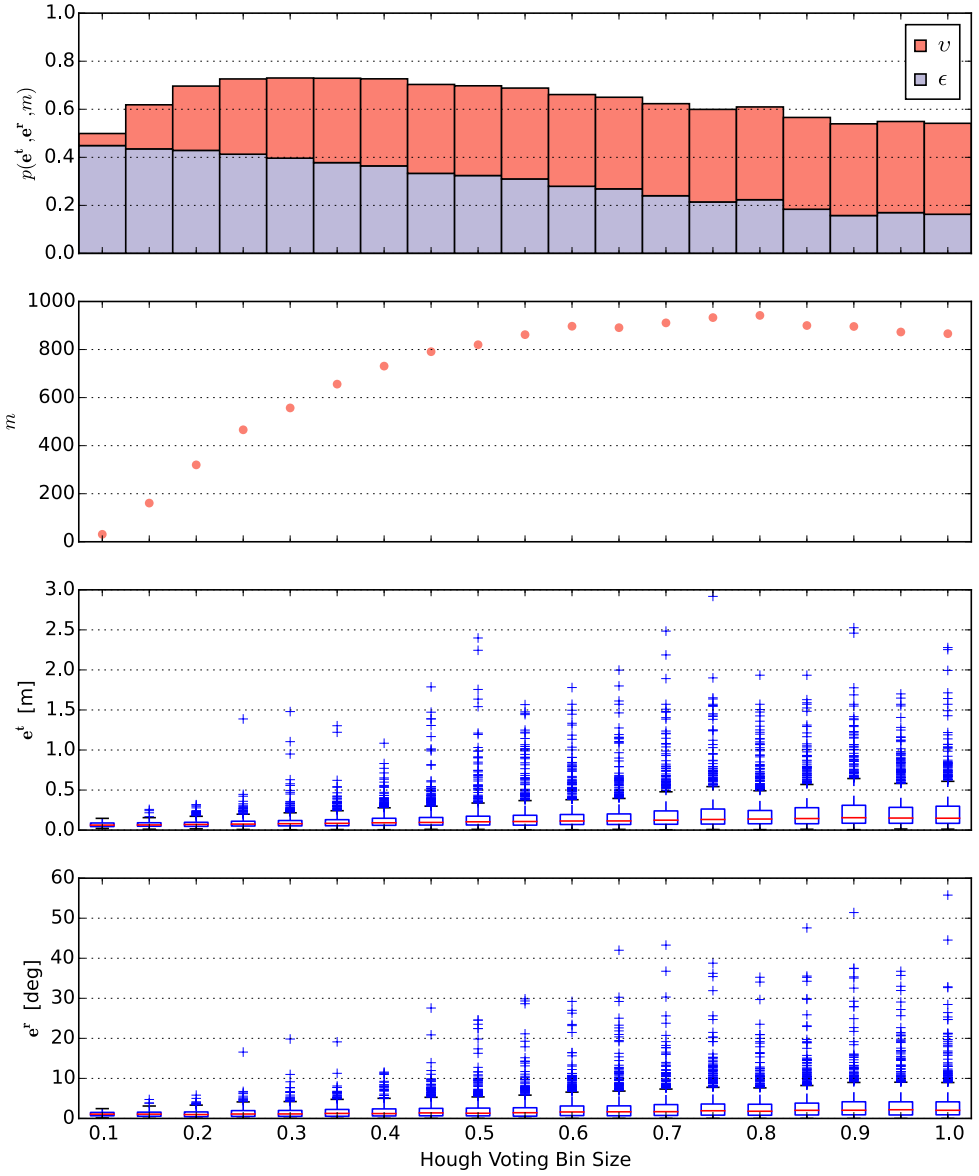


Figure A.14.: Visualization of how the Hough Voting Bin Size parameter changes the map matcher performance on the complete simulation dataset. The performance measure p was parameterized with $m^* = 2500$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

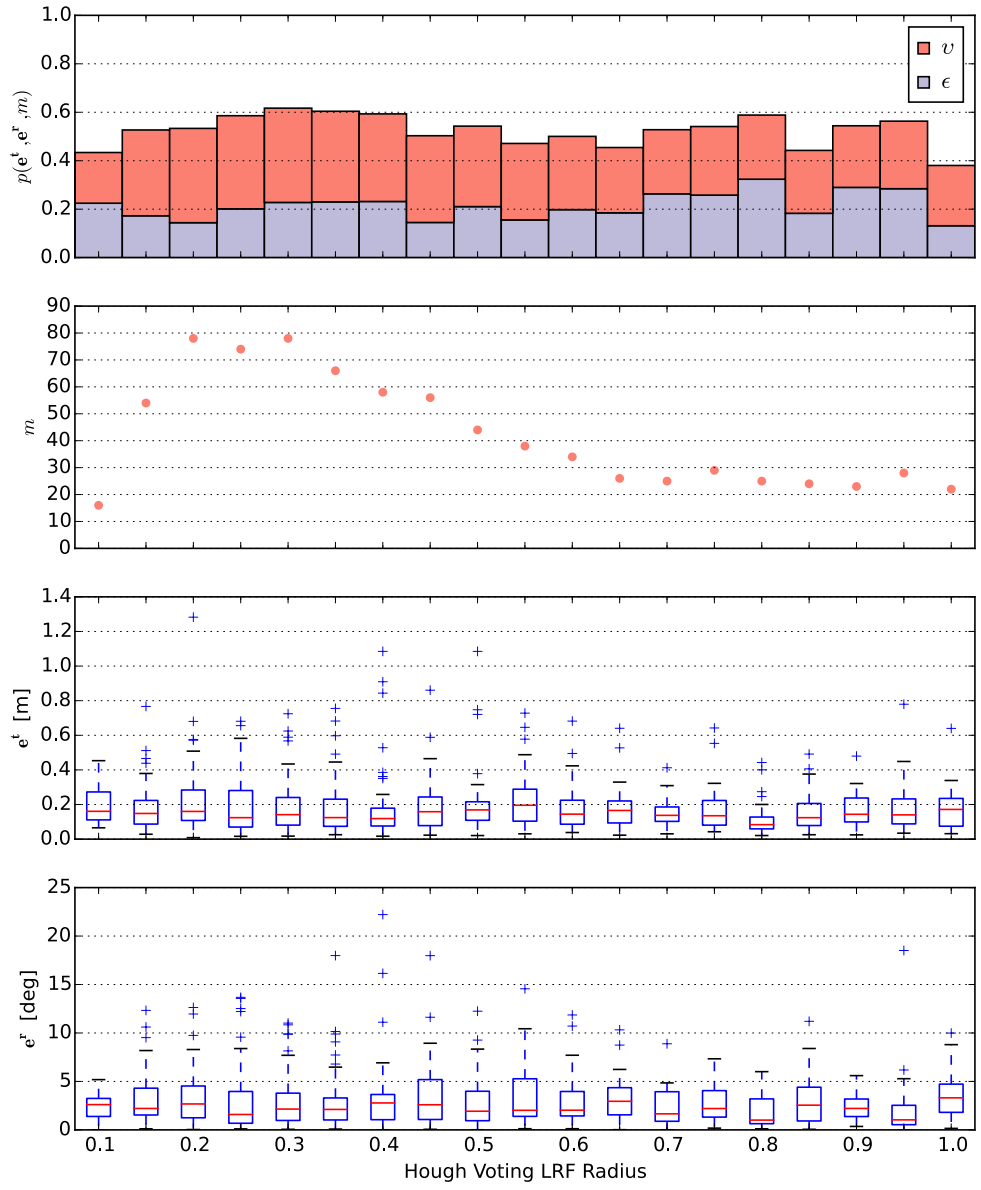


Figure A.15.: Visualization of how the Hough Voting LRF Radius parameter changes the map matcher performance on the complete lab dataset. The performance measure p was parameterized with $\mathrm{m}^* = 200$ and $\mathrm{e}^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

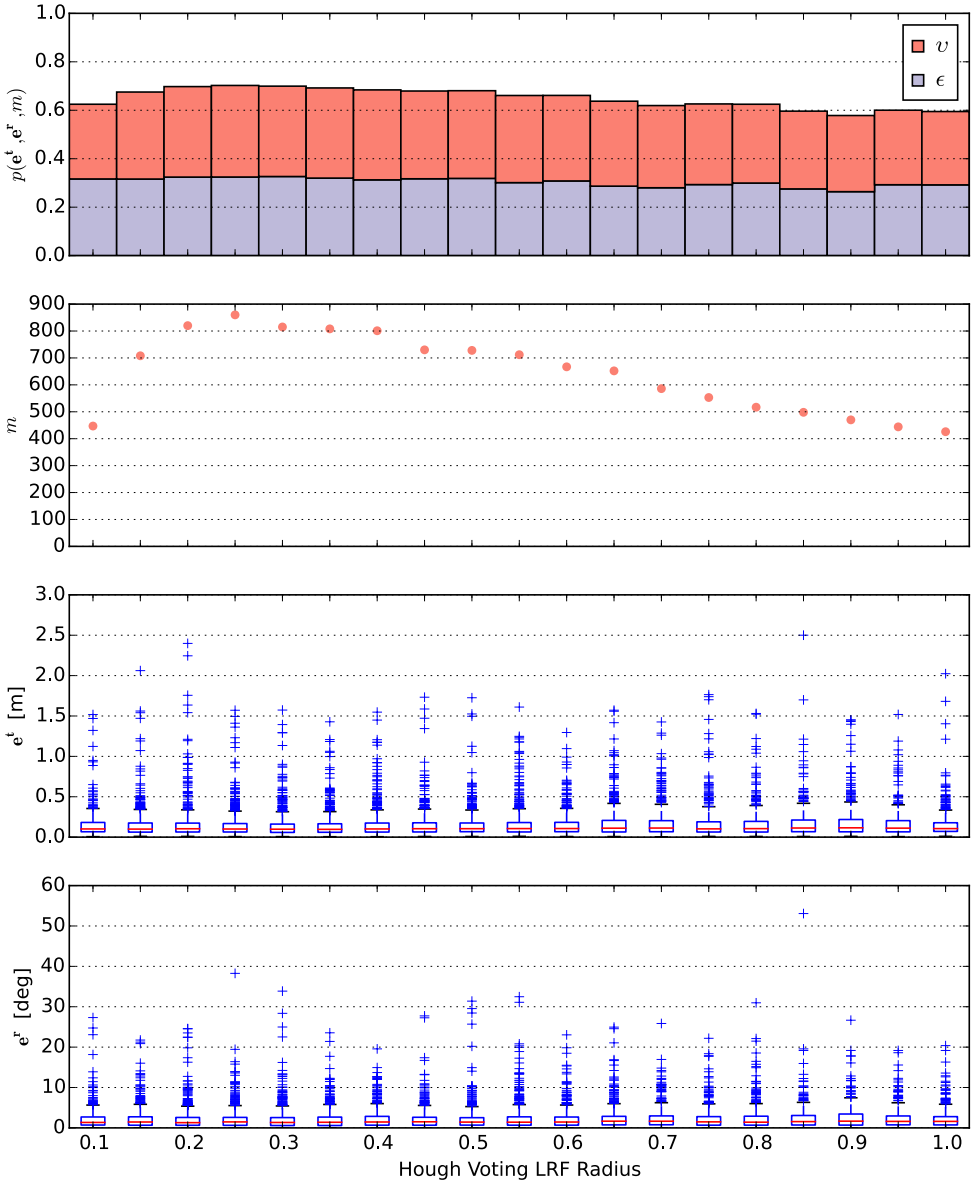


Figure A.16.: Visualization of how the Hough Voting LRF Radius parameter changes the map matcher performance on the complete simulation dataset. The performance measure p was parameterized with $m^* = 2500$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

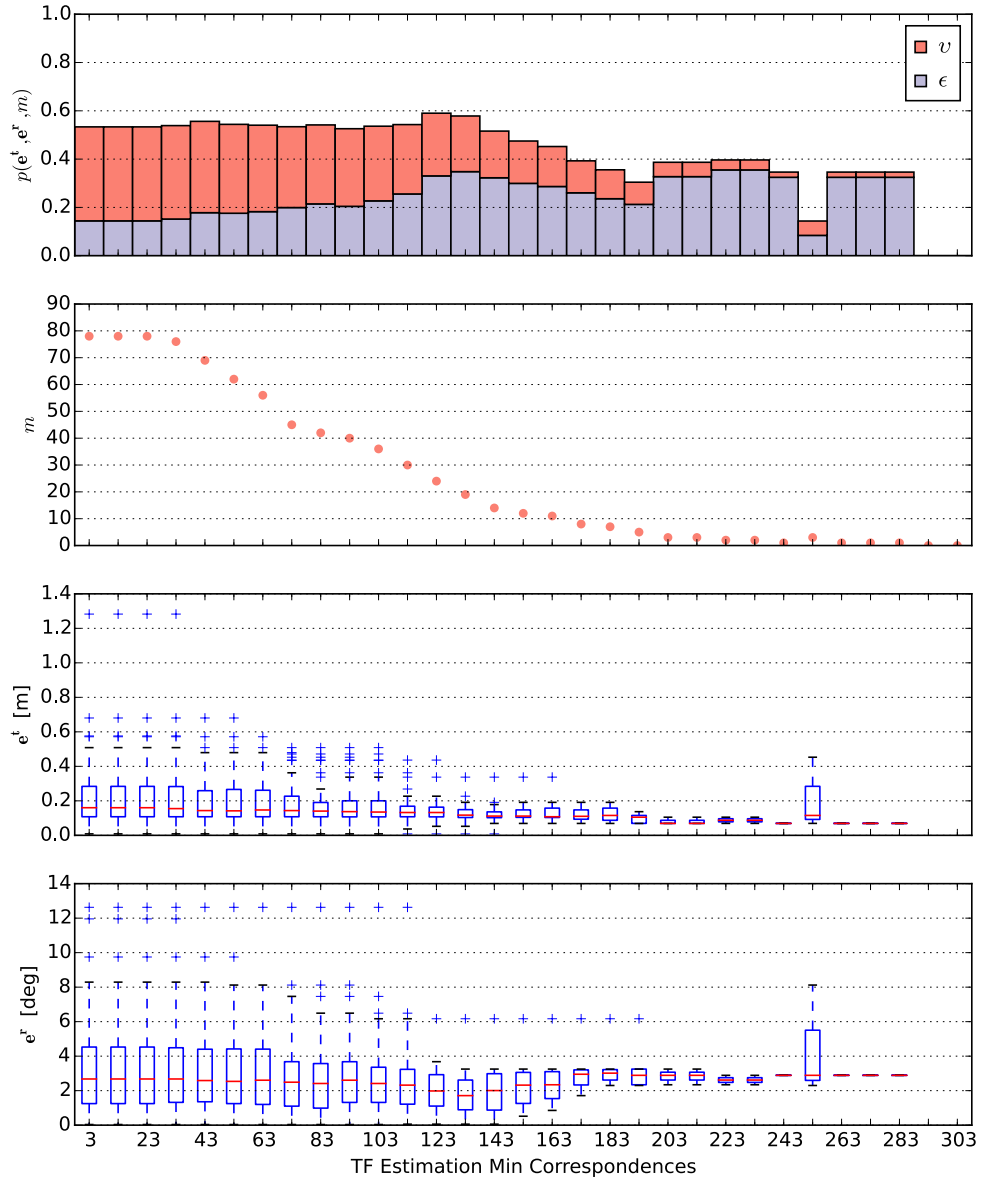


Figure A.17.: Visualization of how the TF Estimation Min Correspondences parameter changes the map matcher performance on the complete lab dataset. The performance measure p was parameterized with $m^* = 200$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

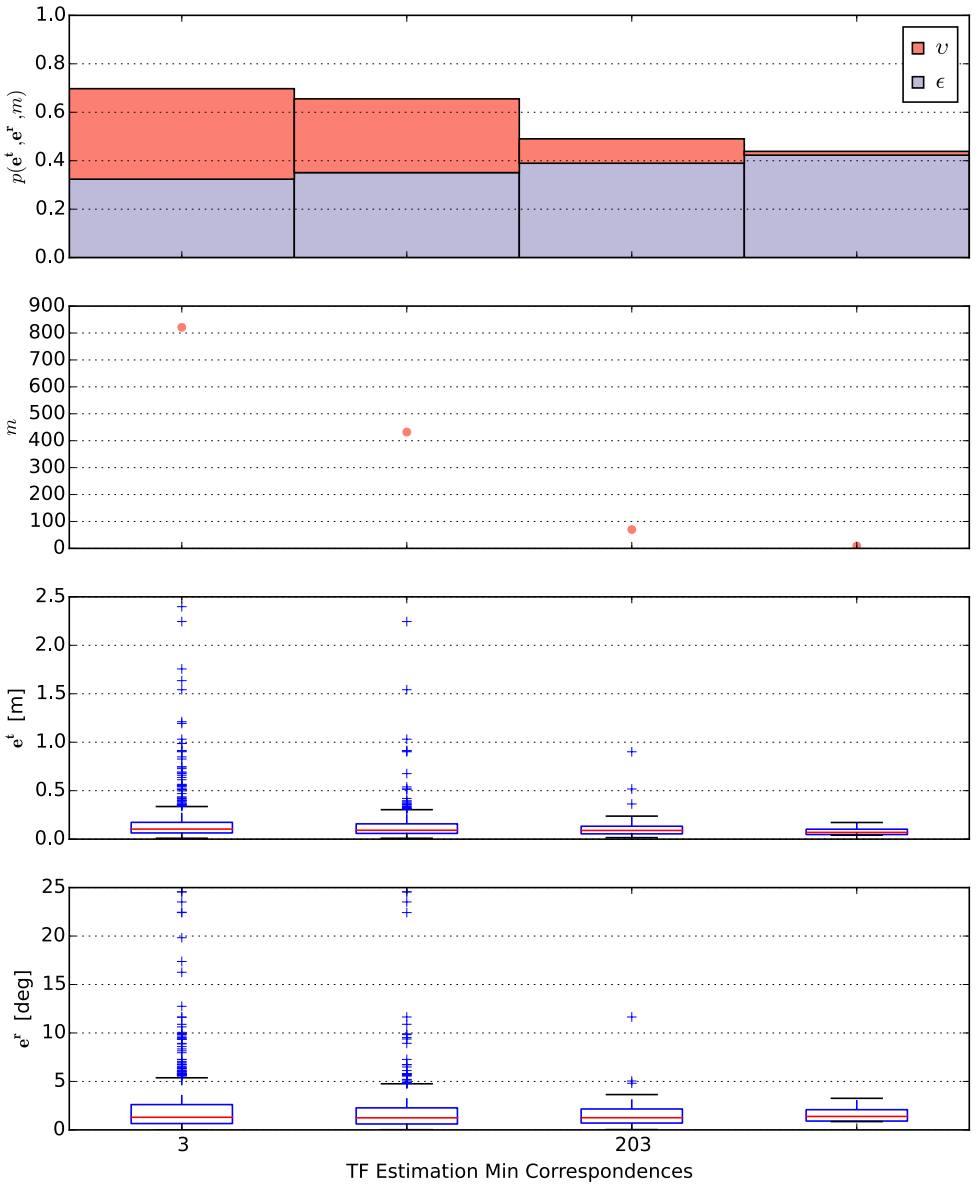


Figure A.18.: Visualization of how the TF Estimation Min Correspondences parameter changes the map matcher performance on the complete simulation dataset. The performance measure p was parameterized with $m^* = 2500$ and $e^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

B. Parameter Pair Experiments

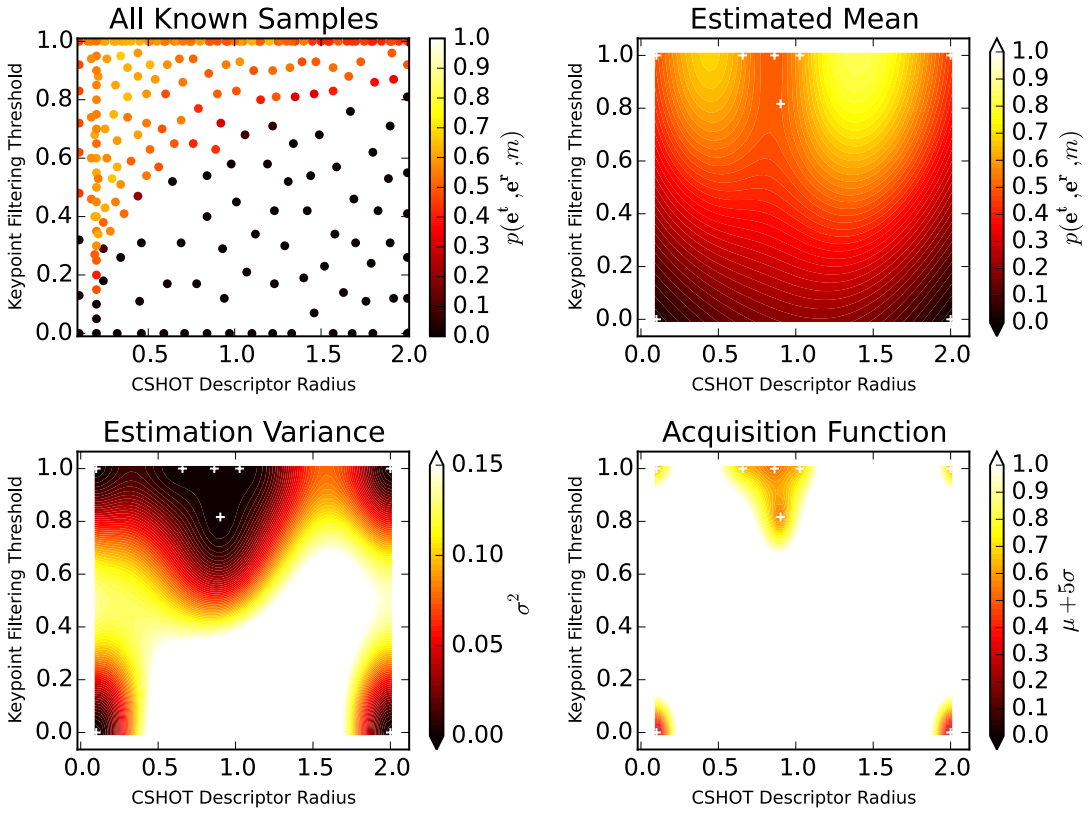


Figure B.1.: Visualization of the optimization process on the two parameters Keypoint Filtering Threshold and CSHOT Descriptor Radius on the lab dataset at iteration 8. The plots of the estimated mean, its variance and the acquisition function induced by them include white crosses to visualize the locations of observations in D_8. The performance measure p was parameterized with $\mathrm{m}^* = 200$ and $\mathrm{e}^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

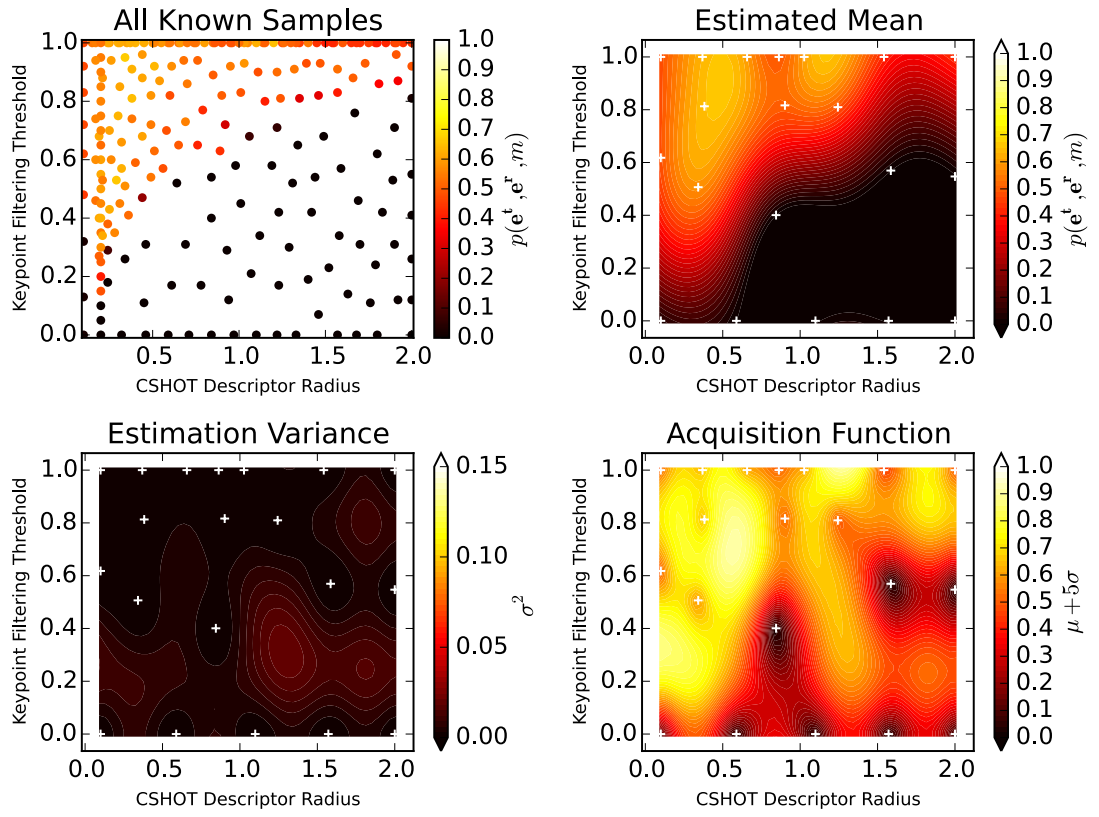


Figure B.2.: Visualization of the optimization process on the two parameters Keypoint Filtering Threshold and CSHOT Descriptor Radius on the lab dataset at iteration 20. The plots of the estimated mean, its variance and the acquisition function induced by them include white crosses to visualize the locations of observations in D_{20}. The performance measure p was parameterized with $\mathrm{m}^* = 200$ and $\mathrm{e}^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

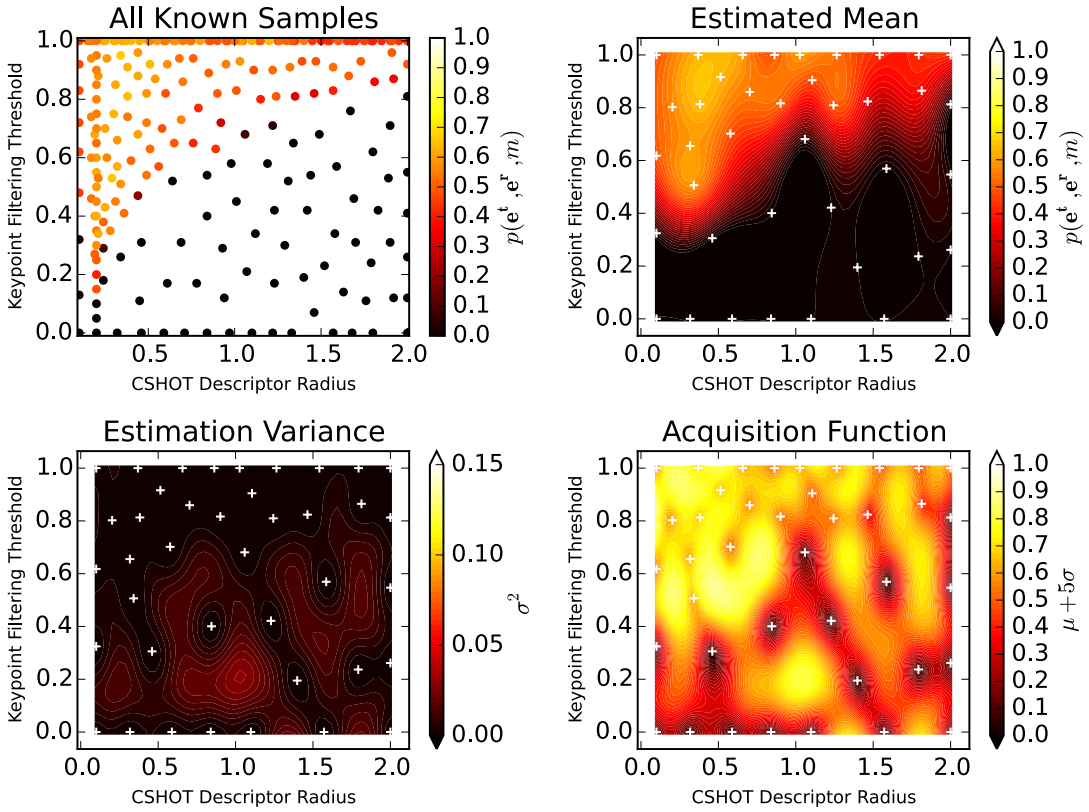


Figure B.3.: Visualization of the optimization process on the two parameters Keypoint Filtering Threshold and CSHOT Descriptor Radius on the lab dataset at iteration 40. The plots of the estimated mean, its variance and the acquisition function induced by them include white crosses to visualize the locations of observations in D_{40}. The performance measure p was parameterized with $\mathrm{m}^* = 200$ and $\mathrm{e}^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

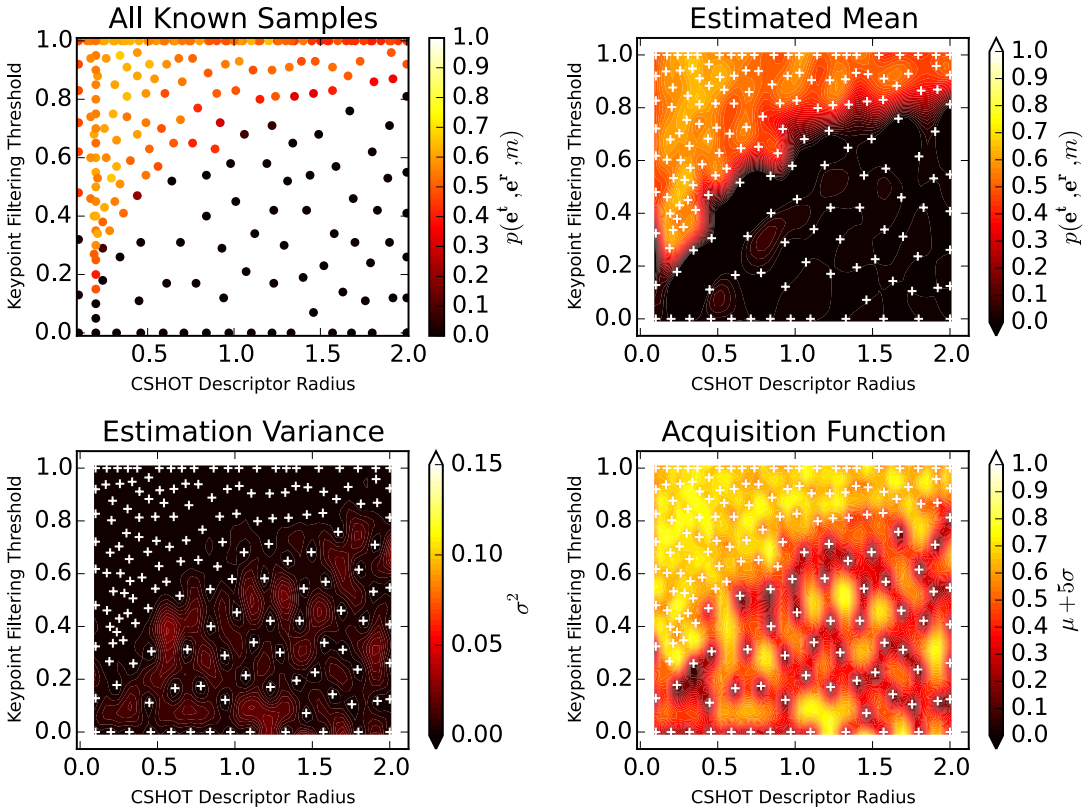


Figure B.4.: Visualization of the optimization process on the two parameters Keypoint Filtering Threshold and CSHOT Descriptor Radius on the lab dataset at iteration 160. The plots of the estimated mean, its variance and the acquisition function induced by them include white crosses to visualize the locations of observations in D_{160}. The performance measure p was parameterized with $\mathrm{m}^* = 200$ and $\mathrm{e}^* = 0.4$. All other map matcher parameters were set to the values defined in $x_{baseline}$ (see Table 5.2).

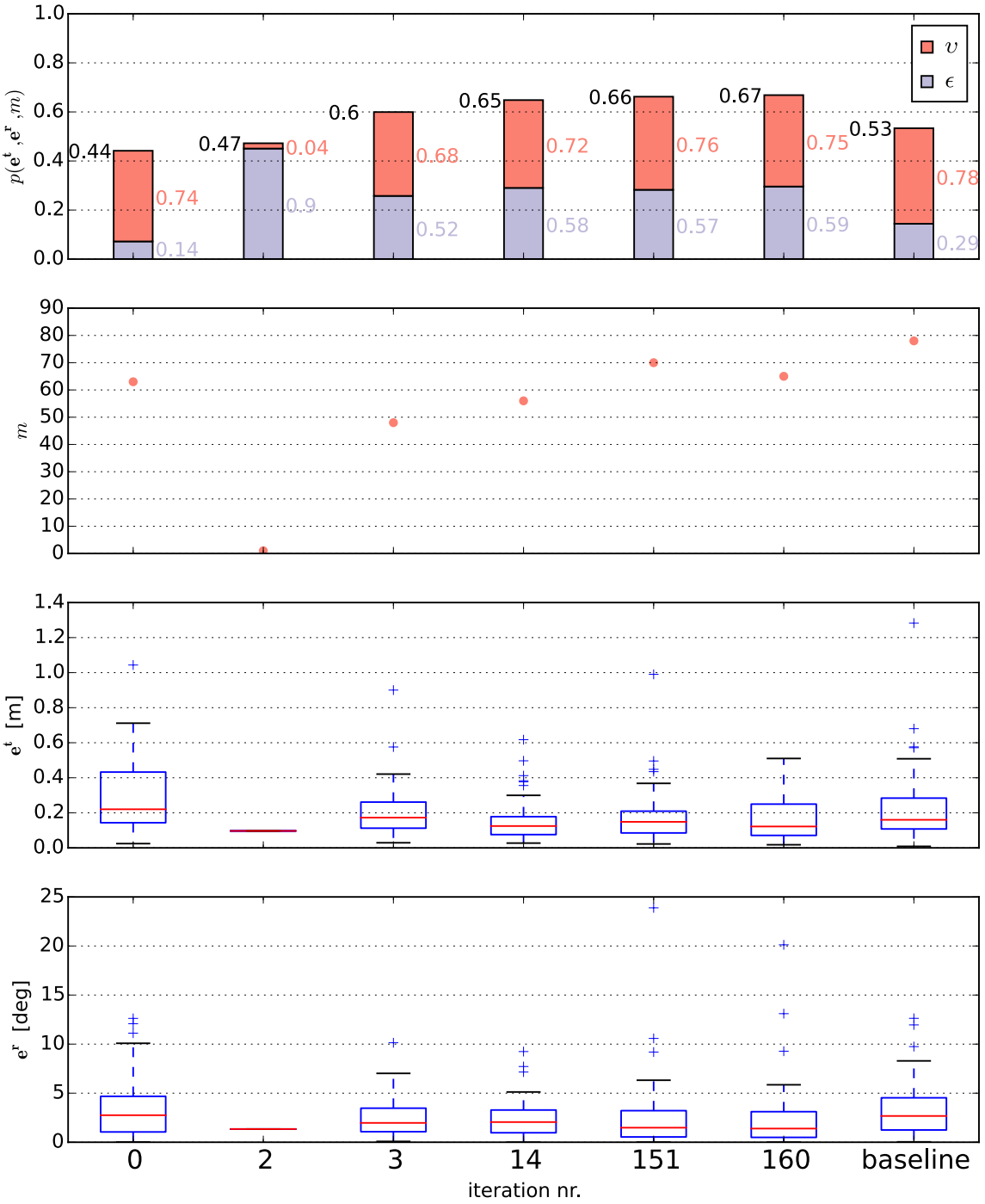


Figure B.5.: Visualization of the best sets of parameters found during joint optimization of the parameter pair Keypoint Filtering Threshold and CSHOT Descriptor Radius on the lab dataset. A new parameter set is only visualized if it improves upon the previously known best set of parameters. Additionally, the parameter set $x_{baseline}$ is displayed on the right, to compare the optimizer results against. The visualization includes the errors e^r and e^t at the bottom, the number of matches m and the resulting performance measure at the top. Note that only the sum of the two parts ε, υ of the performance measures is considered by the Optimizer, all other visualized properties are solely available to the human reader. The performance measure p was parameterized with $m^* = 60$ and $e^* = 0.4$.

C. Joint Optimization

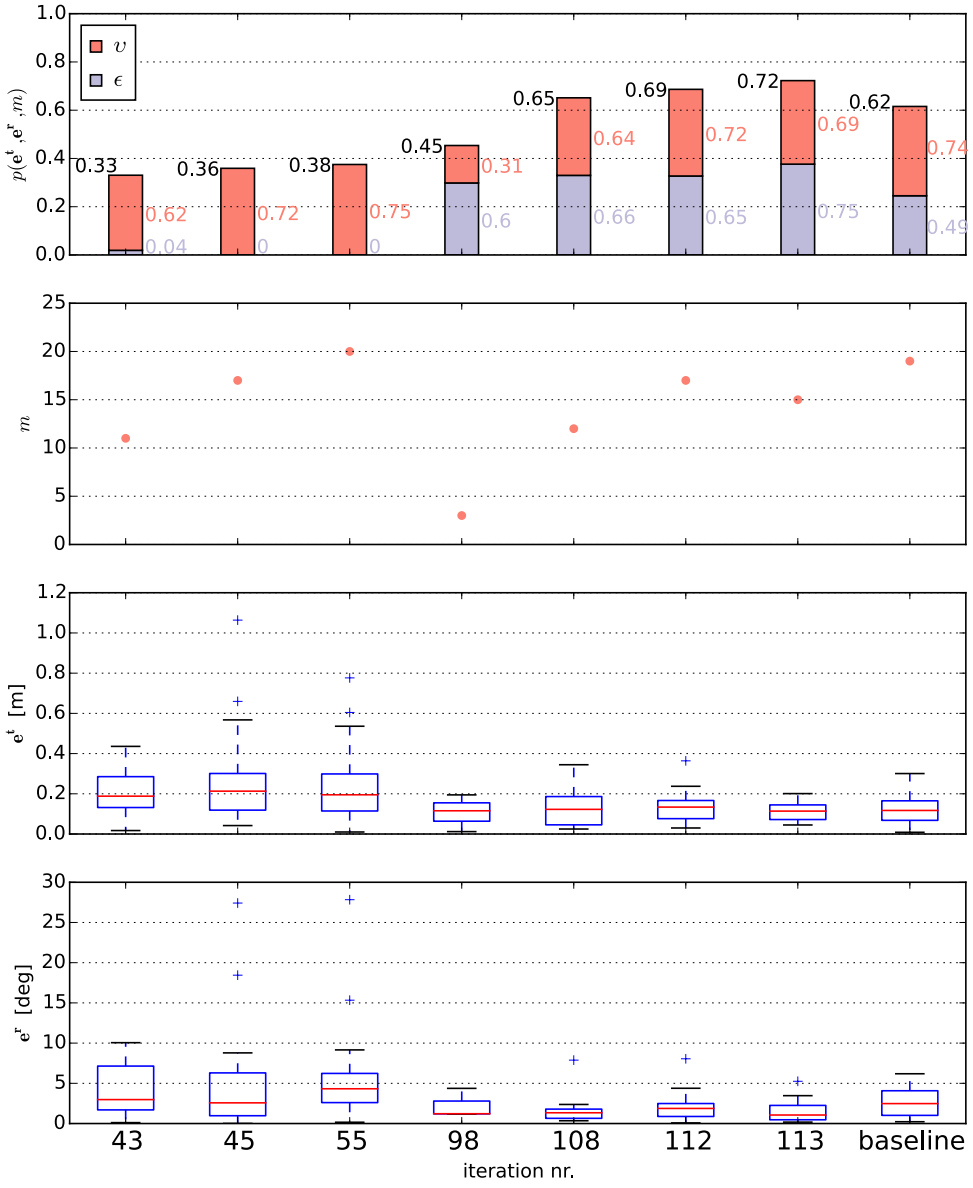


Figure C.1.: Visualization of the best sets of parameters found during the experiment that yielded $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssgt}}$. Note that the displayed data is based on the training dataset of the experiment. The performance of $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssgt}}$ on the both evaluation datasets is visualized in Figures 5.12 and 5.13.

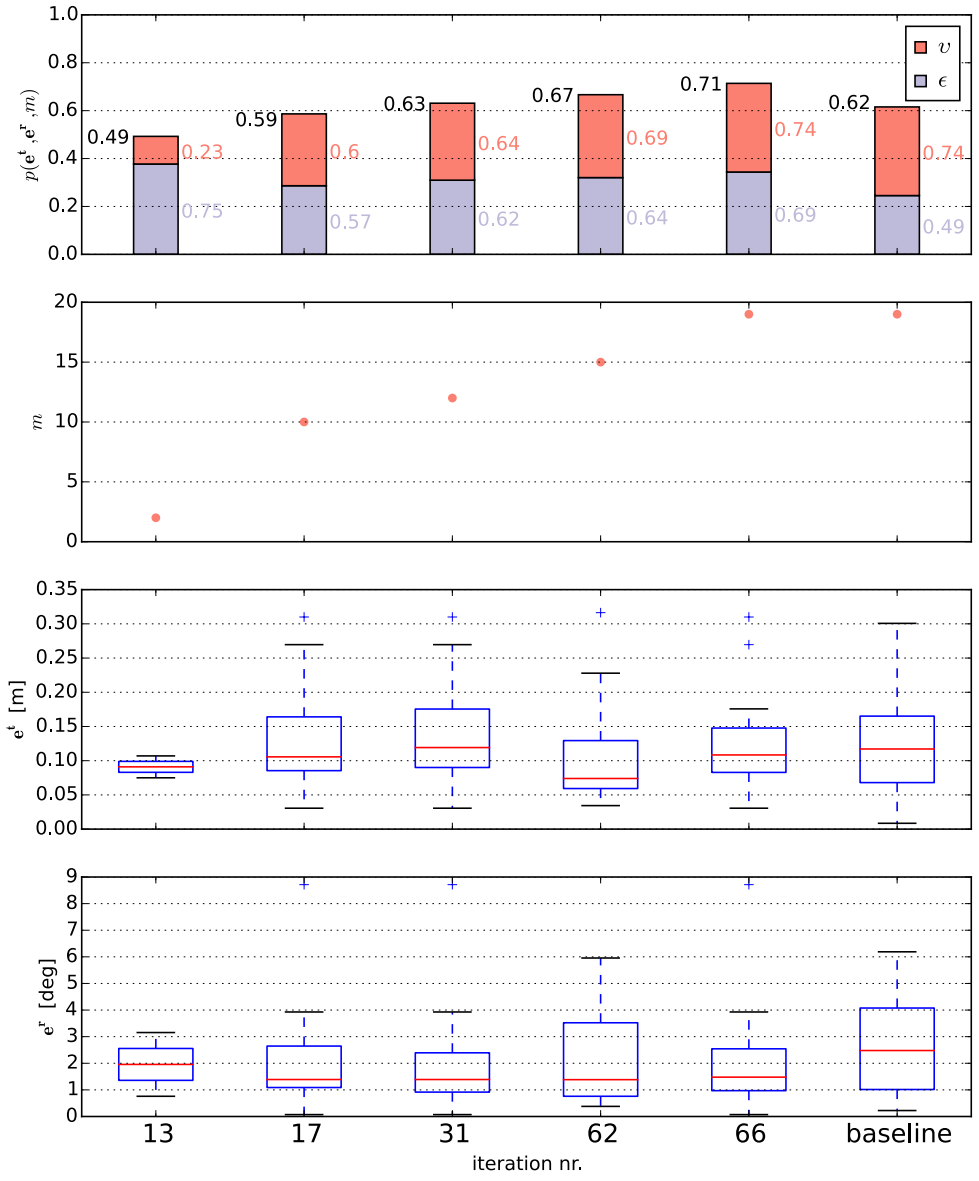


Figure C.2.: Visualization of the best sets of parameters found during the experiment that yielded $\hat{\mathbf{x}}^2_{\mathbf{9Dssgt}}$. An iteration is only visualized if it improves upon the previously known best set of parameters. Additionally, the parameter set $x_{baseline}$ is displayed on the right, to compare the optimizer results against. Note that the displayed data is based on the training dataset of the experiment. The performance of $\hat{\mathbf{x}}^2_{\mathbf{9Dssgt}}$ on the both evaluation datasets is visualized in Figures 5.12 and 5.13.

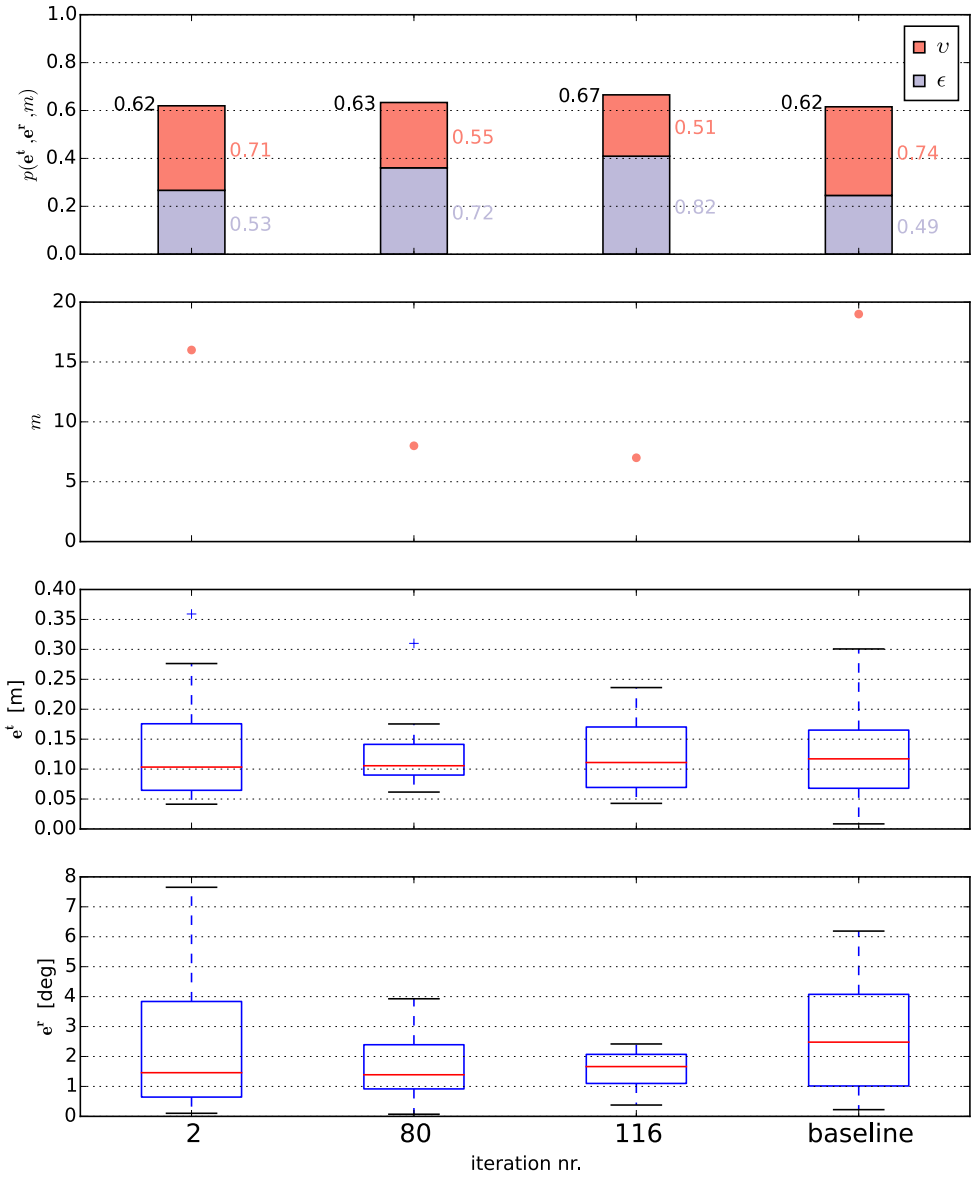


Figure C.3.: Visualization of the best sets of parameters found during the experiment that yielded $\hat{\mathbf{x}}^3_{\mathbf{9Dssgt}}$. An iteration is only visualized if it improves upon the previously known best set of parameters. Additionally, the parameter set $x_{baseline}$ is displayed on the right, to compare the optimizer results against. Note that the displayed data is based on the training dataset of the experiment. The performance of $\hat{\mathbf{x}}^3_{\mathbf{9Dssgt}}$ on the both evaluation datasets is visualized in Figures 5.12 and 5.13.

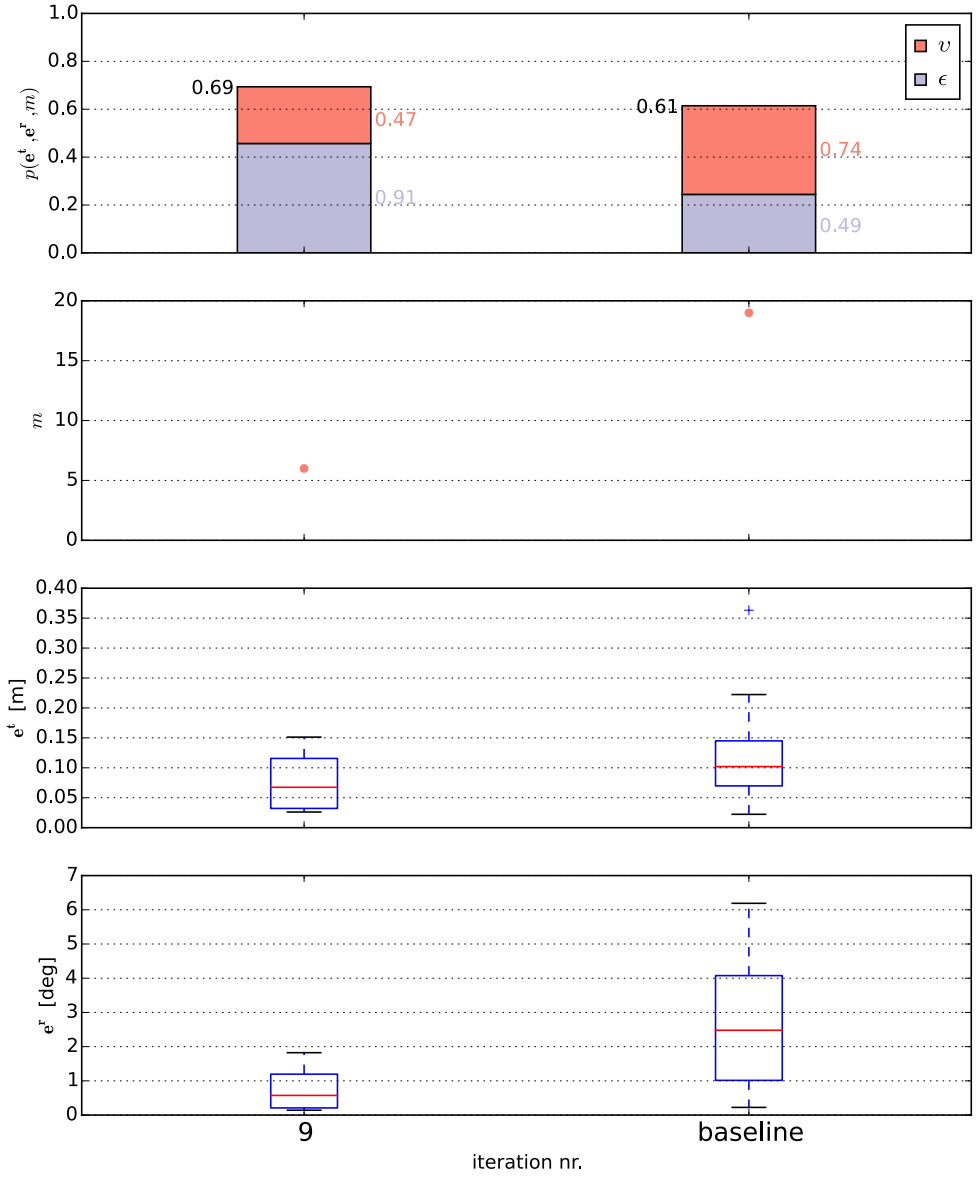


Figure C.4.: Visualization of the best sets of parameters found during the experiment that yielded $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssi}}$. An iteration is only visualized if it improves upon the previously known best set of parameters. Additionally, the parameter set $x_{baseline}$ is displayed on the right, to compare the optimizer results against. Note that the displayed data is based on the training dataset of the experiment. The performance of $\hat{\mathbf{x}}^{\mathbf{1}}_{\mathbf{9Dssi}}$ on the both evaluation datasets is visualized in Figures 5.12 and 5.13.

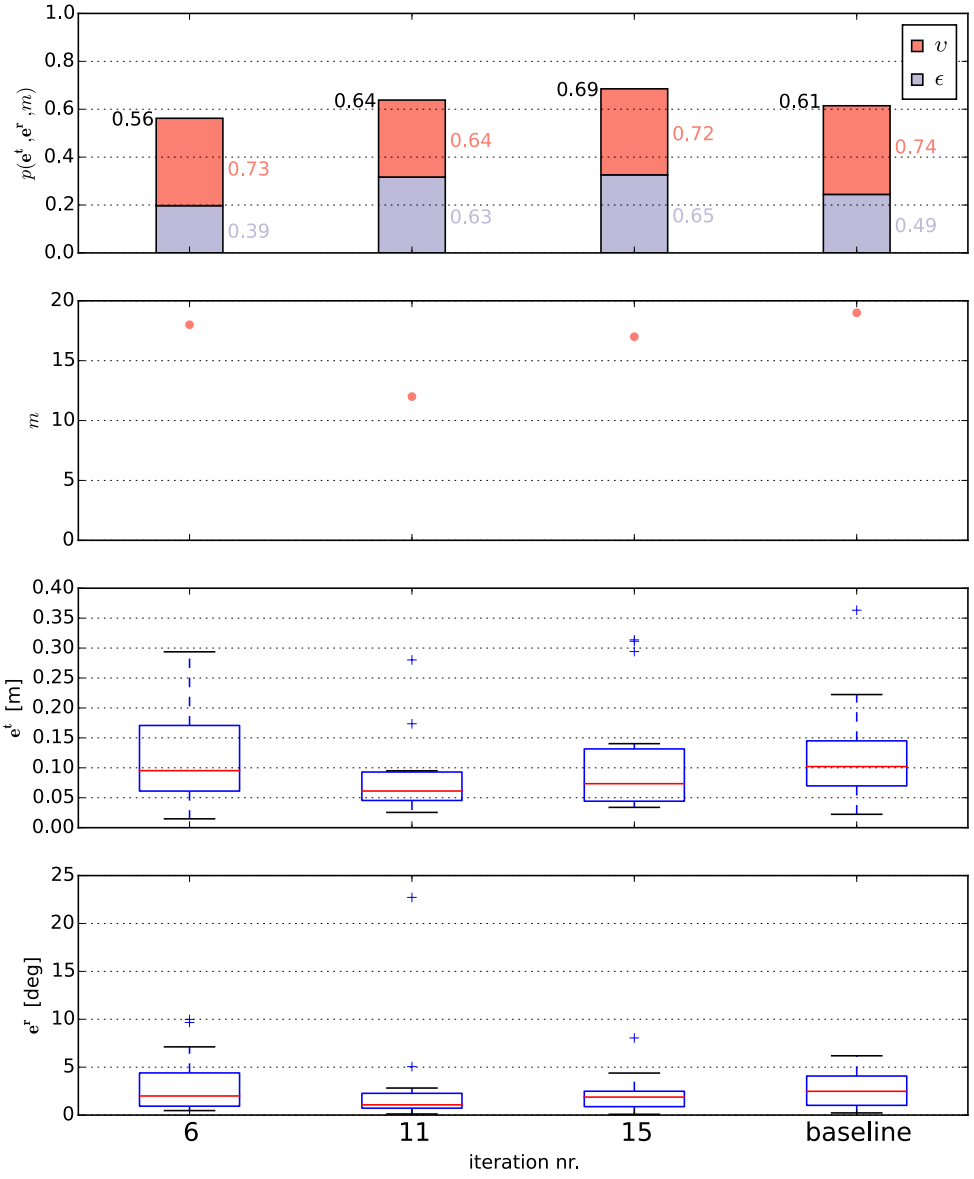


Figure C.5.: Visualization of the best sets of parameters found during the experiment that yielded $\hat{\mathbf{x}}^2_{\mathbf{9Dssi}}$. An iteration is only visualized if it improves upon the previously known best set of parameters. Additionally, the parameter set $x_{baseline}$ is displayed on the right, to compare the optimizer results against. Note that the displayed data is based on the training dataset of the experiment. The performance of $\hat{\mathbf{x}}^2_{\mathbf{9Dssi}}$ on the both evaluation datasets is visualized in Figures 5.12 and 5.13.

D. List of Figures

E. List of Tables

F. Bibliography

[1] IROS 2017 workshop: Introspective methods for reliable autonomy. `http://aass.oru.se/Agora/IROS2017_Introspection`. Last accessed 07/01/2018.

[2] The stanford 3D scanning repository. `http://www.graphics.stanford.edu/data/3Dscanrep/`. Last accessed 23/11/2017.

[3] P. Abrahamsen. A review of Gaussian random fields and correlation functions, 1997.

[4] P. Agrawal, J. Carreira, and J. Malik. Learning to see by moving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 37–45, 2015.

[5] P. J. Besl and N. D. McKay. Method for registration of 3-D shapes, 1992.

[6] C. Brand, M. J. Schuster, H. Hirschmüller, and M. Suppa. Stereo-vision based obstacle mapping for indoor/outdoor SLAM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1846–1853, 2014.

[7] C. Brand, M. J. Schuster, H. Hirschmüller, and M. Suppa. Submap matching for stereo-vision based indoor/outdoor SLAM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5670–5677, 2015.

[8] E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[9] M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1):59–73, 2007.

[10] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[11] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard. Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[12] H. Chen and B. Bhanu. 3D free-form object recognition in range images using local surface patches. *Pattern Recognition Letters*, 28(10):1252–1262, 2007.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[14] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[15] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[16] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, J. Wan, and N. M. Kwok. A comprehensive performance evaluation of 3D local feature descriptors. *International Journal of Computer Vision*, 116(1):66–89, 2016.

[17] R. Held and A. Hein. Movement-produced stimulation in the development of visually guided behavior. *Journal of comparative and physiological psychology*, 56(5):872, 1963.

[18] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2008.

[19] H. Hirschmuller, P. R. Innocent, and J. M. Garibaldi. Fast, unconstrained camera motion estimation from stereo without tracking and robust statistics. In *7th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, volume 2, pages 1099–1104. IEEE, 2002.

[20] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at `http://octomap.github.com`.

[21] R. Horst and P. M. Pardalos. *Handbook of global optimization*, volume 2. Springer Science & Business Media, 2013.

[22] H. Hu and G. Kantor. Efficient automatic perception system parameter tuning on site without expert supervision. In S. Levine, V. Vanhoucke, and K. Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 57–66. PMLR, 13–15 Nov 2017.

[23] H. Hu and G. Kantor. Introspective evaluation of perception performance for parameter tuning without ground truth. *Proceedings of Robotics: Science and Systems, Boston, USA*, 2017.

[24] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. *LION*, 5:507–523, 2011.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[26] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[27] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g 2 o: A general framework for graph optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3607–3613, 2011.

[28] H. Lehner, M. J. Schuster, T. Bodenmüller, and S. Kriegel. Exploration with active loop closing: A trade-off between exploration efficiency and map quality. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[29] J. Levinson and S. Thrun. Unsupervised calibration for multi-beam lasers. In *Experimental Robotics*, pages 179–193. Springer, 2014.

[30] J. L. Morales and J. Nocedal. Remark on "algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization". *ACM Transactions on Mathematical Software (TOMS)*, 38(1):7, 2011.

[31] R. M. Neal. *Bayesian Learning for Neural Networks*, volume 118. Springer Science & Business Media, 2012.

[32] J. Oberländer, S. Klemm, G. Heppner, A. Roennau, and R. Dillmann. A multi-resolution 3-D environment model for autonomous planetary exploration. In *IEEE International Conference on Automation Science and Engineering (CASE)*, pages 229–235, 2014.

[33] J. Oberländer, L. Pfotzer, A. Roennau, and R. Dillmann. Fast calibration of rotating and swivelling 3-D laser scanners exploiting measurement redundancies. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3038–3044, 2015.

[34] M. Pauly, M. Gross, and L. P. Kobbelt. Efficient simplification of point-sampled surfaces. In *Proceedings of the conference on Visualization'02*, pages 163–170. IEEE Computer Society, 2002.

[35] J. Pintér. Global optimization. From MathWorld—A Wolfram Web Resource. `http://mathworld.wolfram.com/GlobalOptimization.html`. Last accessed 15/11/2017.

[36] C. E. Rasmussen and C. K. Williams. *Gaussian processes for machine learning*, volume 1. MIT press Cambridge, 2006.

[37] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz. Towards 3D point cloud based object maps for household environments. volume 56, chapter 4.1. Sparse outlier removal, pages 927–941. Elsevier, 2008.

[38] K. Schmid, F. Ruess, M. Suppa, and D. Burschka. State estimation for highly dynamic flying systems using key frame odometry with varying time delays. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2997–3004, 2012.

[39] M. J. Schuster, C. Brand, S. G. Brunner, P. Lehner, J. Reill, S. Riedel, T. Bodenmüller, K. Bussmann, S. Büttner, A. Dömel, et al. The lru rover for autonomous planetary exploration

and its success in the spacebotcamp challenge. In *International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 7–14. IEEE, 2016.

[40] M. J. Schuster, C. Brand, H. Hirschmüller, M. Suppa, and M. Beetz. Multi-robot 6D graph SLAM connecting decoupled local reference filters. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5093–5100, 2015.

[41] M. J. Schuster, S. G. Brunner, K. Bussmann, S. Büttner, A. Dömel, M. Hellerer, H. Lehner, P. Lehner, O. Porges, J. Reill, S. Riedel, M. Vayugundla, B. Vodermayer, T. Bodenmüller, C. Brand, W. Friedl, I. Grixa, H. Hirschmüller, M. Kaßecker, Z.-C. Márton, C. Nissler, F. Ruess, M. Suppa, and A. Wedler. Towards Autonomous Planetary Exploration: The Lightweight Rover Unit (LRU), its Success in the SpaceBotCamp Challenge, and Beyond. *Journal of Intelligent & Robotic Systems (JINT)*, Nov. 2017.

[42] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[43] N. Srinivas, A. Krause, M. Seeger, and S. M. Kakade. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1015–1022, 2010.

[44] K. H. Strobl and G. Hirzinger. Optimal hand-eye calibration. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 4647–4653, 2006.

[45] F. Tombari and L. Di Stefano. Object recognition in 3D scenes with occlusions and clutter by hough voting. In *Fourth Pacific-Rim Symposium on Image and Video Technology (PSIVT)*, pages 349–355. IEEE, 2010.

[46] F. Tombari, S. Salti, and L. Di Stefano. Unique signatures of histograms for local surface description. In *European conference on computer vision*, pages 356–369. Springer, 2010.

[47] F. Tombari, S. Salti, and L. Di Stefano. A combined texture-shape descriptor for enhanced 3D feature matching. In *18th IEEE International Conference on Image Processing (ICIP)*, pages 809–812, 2011.

[48] F. Tombari, S. Salti, and L. Di Stefano. Performance evaluation of 3D keypoint detectors. *International Journal of Computer Vision*, 102(1-3):198–220, 2013.

[49] A. Wedler, B. Rebele, J. Reill, M. Suppa, H. Hirschmüller, C. Brand, M. Schuster, B. Vodermayer, H. Gmeiner, A. Maier, et al. Lru-lightweight rover unit. In *Proceedings of the 13th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2015.

[50] C. K. Williams and C. E. Rasmussen. Gaussian processes for regression. In *Advances in neural information processing systems*, pages 514–520, 1996.

[51] A. Zeng, S. Song, M. Nießner, M. Fisher, and J. Xiao. 3DMatch: Learning the matching of local 3D geometry in range scans. *arXiv preprint arXiv:1603.08182*, 2016.

[52] Y. Zhong. Intrinsic shape signatures: A shape descriptor for 3D object recognition. In *IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–696, 2009.