A Comparison of Particle Filter and Graph-based Optimization for Localization with Landmarks in Automated Vehicles

Daniel Wilbers[1] Christian Merfels[1] Cyrill Stachniss[2]

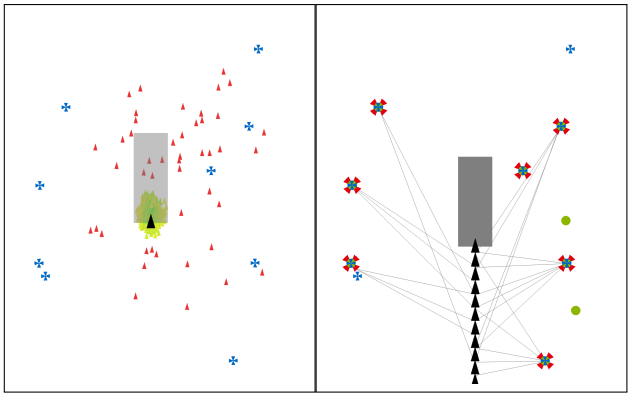
***Abstract*— It is an essential capability of mobile robots and automated vehicles to localize themselves in a map. Multiple state estimation techniques exist to this end, and it is often unclear which one to employ. In this paper we compare two prominent techniques in the context of automated vehicles: particle filters and graph-based localization. The latter is the state-of-the-art approach to simultaneous localization and mapping, and is also superseding the particle filter as the state-of-the-art for pure localization. We compare both algorithms to show why. For both state estimation algorithms, we detect pole-like objects in laser scanner data and compare them against a prebuilt landmark map. The main novelty of this work is the experimental comparison on a real prototype vehicle. Furthermore, we discuss the different advantages of both algorithms in the context of automated driving and additionally show how both approaches can adapt their computational demand to the available resources at runtime.**

I. Introduction

Localization is a fundamental task for mobile robots. Particle filter localization [**?**], [**?**] (often called Monte Carlo localization) is a highly popular approach in the mobile robotics community. For simultaneous localization and mapping (SLAM) approaches, the focus has clearly shifted in the last years from using Rao-Blackwellized particle filters [**?**] towards graph-based approaches [**?**], [**?**]. However, this cannot be said for pure localization approaches. The majority of these approaches seem to be implemented with either a Kalman filter variant or a particle filter. Only recently a few graph-based localization approaches were presented [**?**], [**?**], [**?**]. The question is therefore: is it beneficial for pure localization approaches to prefer the graph or particle filter paradigm? In this paper we provide insights and experiments to compare these two.

Localization is a prerequisite for interpreting information stored in maps. This information is beneficial for navigation, path planning, and perception. If we want to be able to make use of this map information, then we need to be able to localize with respect to the map. Thus, in this paper we focus on the problem of map-based self-localization in the context of automated vehicles.

Nowadays, Global Navigation Satellite Systems (*GNSS*) are commonly used for the localization in road vehicles. The most accurate of these systems combine Inertial Measurements Units and Real-time kinematics to achieve centimeter-level results. Besides their great costs, satellite based systems suffer from strong multipath and blocked line of sight effects especially in urban scenarios. A common approach for highly available localization in scenarios without or limited satellite reception is to use landmark-based localization techniques which we investigate in this paper.



(a) Particle filter (b) Graph-based localization

Fig. 1: Localization problem in the same situation solved with two different approaches. (a) Particle filter. The color of a particle represents its current weight. If a weight is high the color ranges from green to yellow. Recovery particles from GNSS receive a low weight (red). (b) Sliding window graph. Finding matches (red crosses) between landmark observations (green dots) and map landmarks (blue crosses) allows us to constrain the vehicle trajectory (black triangles). The connections (gray lines) illustrate from which poses a landmark was observed. The current vehicle pose is depicted as a gray rectangle.

The localization information in our maps consists of landmarks. These are static objects on the road or close to it with a semantic meaning, such as traffic signs, road markings, or other elements of the environment. In this paper, we rely on pole-like vertical objects. These are for example lampposts, round pillars, vertical trees, and similar structures. It is possible to choose a different kind of landmark type for other domains without contradicting the general findings of this paper. Landmark maps are advantageous over raw sensor data maps (e.g., point clouds) by storing semantic information, being easier to maintain and inspect for correctness, and by requiring smaller storage and memory capacities and thus being faster to process.

Using landmarks for localization has been studied under the aspects of different sensors (cameras, lidars, and radars), different types of landmarks (e.g., poles, road markings), and different state estimation methods (e.g., Kalman filters, particle filters, and sliding window filters). In this paper we compare the localization performance of a particle filter and

[1]Daniel Wilbers and Christian Merfels are with Volkswagen Group Research, Wolfsburg, and Institute of Geodesy and Geoinformation, University of Bonn, Germany.

[2]Cyrill Stachniss is with Institute of Geodesy and Geoinformation, University of Bonn, Germany.

an optimization-based sliding window filter. Fig. 1 illustrates these two concepts. For a fair comparison we use the same map, type of landmarks, vehicle, sensors, and input data (pole detections, odometry, and GNSS data). Our investigation focuses on the application in automated vehicles. Nevertheless our findings are in principle transferable to other domains.

The main contribution of this paper is a comparison of particle filter and graph-based localization in the context of automated vehicles. We discuss the characteristics of both algorithms and investigate three key aspects in the context of automated driving: (i) accuracy, (ii) adaptive behavior in terms of computational resources, and (iii) benefit of estimating old poses.

We experimentally evaluate these questions on data gathered on a real prototype vehicle.

II. Related Work

Map-based localization is typically either carried out as a standalone approach by comparing sensor against map data or as part of a SLAM approach. The usual state estimation machinery for this encompasses extended Kalman filters (EKFs) and their variants, particle filters, and graph-based approaches. Stachniss et al. [**?**] review these three methodologies.

SLAM has been first approached with different variants of Kalman filters. This has shifted in the last one to two decades with particle filters. Their key trick is to use a method called Rao-Blackwellization which essentially leads to each particle having a separate map. Nowadays, the state-of-the-art approach to SLAM is graph-based SLAM. It represents measurements in the form of soft constraints and optimizes the resulting problem with nonlinear least squares solvers. We refer to the work of Cadena et al. [**?**] for a recent and to Durrant-Whyte and Bailey [**?**] for a general overview.

Pure localization has also started off by being approached with Kalman filters, and continues so successfully for certain applications. Similarly to SLAM, particle filters have been introduced and used in the last two decades. However, few approaches propose map-based localization with graph-based systems.

Lundgren et al. [**?**] use camera and radar detections of landmarks for map-based self-localization with a particle filter. One of their main findings is that both sensing modalities are needed for a robust and precise result. Deusch et al. [**?**] focus on detecting lane markings and road paintings in grayscale camera images and laser scanner data. Their particle filter compares the detections to a self-built map and computes a global pose estimate. Similarly to Lundgren et al., they find that using both sensing modalities increases robustness and reliability at the expense of a higher computational cost. Spangenberg et al. [**?**] detect poles in stereo camera images. These are used in a coupled particle and Kalman filter setup to estimate the current vehicle pose by comparing them against a prebuilt map. They favor poles as type of landmark because they are distinct and long-term stable. Moreover, they can be reliably detected and efficiently stored. In a similar line of thinking, Brenner [**?**] proposes to use pole landmarks for global localization. Schlichting and Brenner [**?**] apply this system on a vehicle with an automotive-grade lidar. Schindler [**?**] approaches the localization problem by fusing camera-based lane marking and lidar-based pole detections with a particle filter. The measurement model employs prototype fitting to estimate the likelihood of the detections given the map. Interestingly, the approach by Levinson et al. [**?**] generates maps with a graph-based SLAM system, while a particle filter is employed for localization. Wu et al. [**?**] presented a localization approach based on graph optimization using camera and lidar data. In comparison to their work, our graph-based approach benefits from a different association strategy as well as grid independent landmark detections [**?**]. Recently, Harr et al. [**?**] presented a pose graph approach using grayscale cameras. Our graph-based approach uses a different association strategy and relies on different sensors resulting in a more accurate system.

An initial version of our particle filter is described by Stess [**?**]. Our graph-based localization approach is built on the foundation of a pose fusion system [**?**], [**?**] and is described in more detail by Wilbers et al. [**?**]. We summarize both approaches in Sec. III.

III. Localization

Map-based localization aims to estimate the current pose of the robot with respect to the map. The pose is defined as the two-dimensional location and orientation $\boldsymbol{x}_t = [x, y, \theta]^\top$ within the coordinate frame of the map. To this end, we take the measurements $\boldsymbol{z}_t = [\boldsymbol{z}^{\mathrm{o}}, \boldsymbol{z}^{\mathrm{g}}, \boldsymbol{z}^{\mathrm{l}}]^\top$ into account. Here, $\boldsymbol{z}^{\mathrm{o}}$ denotes the measured odometry. We assume that the velocity and yaw rate measurements have already been preprocessed with an appropriate motion model into a relative movement vector $\boldsymbol{z}^{\mathrm{o}} = [\Delta x, \Delta y, \Delta\theta]^\top$. The vector $\boldsymbol{z}^{\mathrm{g}} = [x^{\mathrm{g}}, y^{\mathrm{g}}, \theta^{\mathrm{g}}]^\top$ denotes the global pose estimate from the GNSS receiver. The landmark detections are stored in the vector $\boldsymbol{z}^{\mathrm{l}} = \{[x_k^{\mathrm{l}}, y_k^{\mathrm{l}}]^\top\}_{k=1}^{K}$. We compare them against the landmark map $\boldsymbol{m} = \{[x_i^{\mathrm{m}}, y_i^{\mathrm{m}}]^\top\}$.

In this paper, landmarks are objects of cylindrical shape such as poles, traffic lights, or traffic signs. We detect them in lidar data of Velodyne VLP16 that are mounted on the roof of our prototype vehicle. The detection step is a straightforward geometric pattern matching similar to the approach by Brenner [**?**].

The landmark map contains a list of these poles in the world coordinate frame. As each landmark is reduced to two coordinates, the map can be stored efficiently. Its size remains small even for large environments compared to typical feature or point cloud methods.

One common requirement for both our localization approaches is to deliver a pose estimate at a fixed frequency of $20\,\mathrm{Hz}$.

The goal in vehicle localization is to estimate the most likely vehicle pose $\boldsymbol{x}_t^*$ given the measurements and the map. Formally, we seek to compute

$$\boldsymbol{x}_t^* = \arg\max_{\boldsymbol{x}} p(\boldsymbol{x}_t | \boldsymbol{z}_t, \boldsymbol{m}). \tag{1}$$

While both estimation algorithms, the particle filter and the graph-based localization, contribute to this goal, they employ different methods to achieve it.

A. Particle filter

The particle filter is a nonparametric Bayesian filter that represents its posterior distribution via a set of hypotheses. Each of these hypotheses is called a particle and consists of a weighted estimate of the vehicle's location and orientation. Therefore, the state is represented as a set of $\mathcal{M}$ particles $\mathcal{S} = \{\langle \boldsymbol{x}^i, \omega^i \rangle\}_{i=1}^{\mathcal{M}}$, where each particle has a weight ω^i. The sum of all weights is equal to one. This set of weighted samples can represent arbitrary probability distributions given enough particles.

Initially, $\mathcal{S}_0$ is sampled from a large Gaussian distribution around the pose estimate $\boldsymbol{z}^{\mathrm{g}}$ from a low-cost GNSS receiver. Subsequently, the particle filter algorithm estimates $\mathcal{S}_t$ at time t based on its previous estimate $\mathcal{S}_{t-1}$. Its three main steps are:

1) motion update: propagate each particle in $\mathcal{S}_{t-1}$ forward according to the odometry measurement $\boldsymbol{z}^{\mathrm{o}}$ plus some artificial noise.
2) importance weighting: compute $\omega_t^i = p(\boldsymbol{z}_t^{\mathrm{l}} | \boldsymbol{x}_t^i, \boldsymbol{m})$, i.e., evaluate the sensor model. Each weight is computed as the probability that the landmark measurements $\boldsymbol{z}_t^{\mathrm{l}}$ were observed from the corresponding pose of the particle $\boldsymbol{x}_t^i$.
3) resampling: randomly draw $\mathcal{M}$ samples from the set of particles. The likelihood to draw a particle is proportional to its weight. We employ *low variance resampling* [**?**] to reduce the risk of particle depletion.

The sensor model is key for a good filter performance. Assuming statistical independence and computing the log-likelihood for numerical stability we evaluate it to

$$\ln \omega_t^i = \ln p(\boldsymbol{z}_t^{\mathrm{l}} | \boldsymbol{x}_t^i, \boldsymbol{m}) = \sum_{k=1}^{\mathcal{K}} \ln p(\boldsymbol{z}_{t,k}^{\mathrm{l}} | \boldsymbol{x}_t^i, \boldsymbol{m}). \quad (2)$$

We solve data association via a simple but sufficient nearest neighbor strategy. This is possible because our landmarks are chosen as objects that are distinctively placed from each other. Suppose that $\boldsymbol{m}_p$ is the map landmark that we associate to $\boldsymbol{z}_{t,k}^{\mathrm{l}}$. If we assume a Gaussian distribution and discard $\boldsymbol{x}_t^i$ after having used it to bring the map landmarks into the local vehicle reference frame of the particle, we obtain

$$\ln \omega_t^i = \sum_{k=1}^{\mathcal{K}} \ln p(\boldsymbol{z}_{t,k}^{\mathrm{l}} | \boldsymbol{x}_t^i, \boldsymbol{m}) = \sum_{k=1}^{\mathcal{K}} \ln \mathcal{N}(\boldsymbol{z}_{t,k}^{\mathrm{l}} | \boldsymbol{m}_p). \quad (3)$$

This term can become a large negative number depending on the Mahalanobis distances between the observed and the map landmarks. As a consequence, naively recovering $\omega_t^i = \exp(\ln \omega_t^i)$ is not numerically stable. However, we are not directly interested in ω_t^i but rather in its normalized value $\hat{\omega}_t^i$ such that $\sum_i \hat{\omega}_t^i = 1$. To circumvent the numerical stability issues, we therefore regularize the weights as

$$\ln \tilde{\omega}_t^i = \ln \omega_t^i - \ln \omega_t^{\max} = \ln \frac{\omega_t^i}{\omega_t^{\max}} \quad (4)$$

with $\omega_t^{\max}$ being the maximum weight at the current time step t. This trick allows us to recover

$$\hat{\omega}_t^i = \frac{\exp(\ln \tilde{\omega}_t^i)}{\sum \exp(\ln \tilde{\omega}_t^k)} = \frac{\frac{\omega_t^i}{\omega_t^{\max}}}{\sum \frac{\omega_t^k}{\omega_t^{\max}}} = \frac{\omega_t^i}{\sum \omega_t^k} \quad (5)$$

without running into numerical issues due to explicitly computing $\exp(\ln \omega_t^i)$.

The number of particles can be dynamically adapted in the resampling step. This can either be done using Kullback-Leibler divergence sampling [**?**], or by using a resource-adaptive approach to maximize the number of particles. We opt for the latter for better comparability with our optimization-based approach. It consists in measuring the needed computation time for the current cycle and by linearly adapting $\mathcal{M}_{t+1}$ such that we aim to make exact use of the available cycle time. This is based on the fact that the computational complexity of a particle filter is in $\mathcal{O}(\mathcal{M})$.

We use $1\,\%$ of the available particles to add recovery particles sampled around the GNSS pose estimate. This is, for example, useful when we drive for some time in unmapped areas and reenter mapped areas.

An important design question is when to resample. The naive approach to resample in every step can lead to issues regarding state diversity and accidentally dropping good hypotheses. One common idea to counteract this is to decide when to resample based on the criterion of *effective number of particles* [**?**]

$$N_{\mathrm{eff}} = \frac{1}{\sum_{i=1}^{\mathcal{M}} (\omega_t^i)^2} < \frac{\mathcal{M}}{2}. \quad (6)$$

While this yields reasonable results, we have observed even better results with taking into account the history of the particle weights. For this, we compute with

$$\omega_t^i = \alpha \cdot p(\boldsymbol{z}_t^{\mathrm{l}} | \boldsymbol{x}_t^i, \boldsymbol{m}) + (1 - \alpha) \cdot \omega_{t-1}^i \quad (7)$$

the exponential moving average of the particle weight. The idea is to prevent low weights for good particles at the current time step due to false positive detections, if they have obtained high weights in the past.

To obtain a single vehicle pose as output of our particle filter, we fit a Gaussian distribution to the particle cloud, i.e., we take the weighted mean of the particle poses. Other choices, such as taking the particle with the maximum weight or clustering the particle set, are also possible.

B. Sliding window graph-based localization

The optimization-based localization can be represented as a factor graph. While measurements z_i and their corresponding covariance matrices $\boldsymbol{\Sigma}_i$ can be seen as factors, state variables are denoted as nodes. See Fig. 2 for an illustration. The graphical representation helps to identify the required error functions and describes the structure of

the underlying optimization problem. In our approach the state vector $\tilde{\boldsymbol{x}} = \begin{bmatrix}\boldsymbol{x}^{\mathrm{p}} & \boldsymbol{x}^{\mathrm{l}}\end{bmatrix}$ is a sliding window with length n of vehicle poses $\boldsymbol{x}^{\mathrm{p}}_{t-n+1:t}$. Additionally it contains all landmark estimates $\boldsymbol{x}^{\mathrm{l}}$. Although including landmarks into the estimation is unusual in common pose graph localization approaches, the benefits of our approach are easy visual inspection and predominantly the ability to refine landmark maps. As updating a map in the context of automated driving may affect safety requirements, we store the updates separately for later inspection. The update process itself is not part of this paper as we focus on pure online localization.

Assuming Gaussian distributions and i.i.d. measurements, we represent (1) as a weighted least squares problem

$$\begin{aligned}\tilde{\boldsymbol{x}}^* = \arg\min_{\tilde{\boldsymbol{x}}} &\sum_i e^{\mathrm{o}}(\boldsymbol{x}^{\mathrm{p}}, \boldsymbol{z}^{\mathrm{o}}_i)^\top \boldsymbol{\Omega}^{\mathrm{o}}_i e^{\mathrm{o}}(\boldsymbol{x}^{\mathrm{p}}, \boldsymbol{z}^{\mathrm{o}}_i) \\ &+ \sum_i e^{\mathrm{g}}(\boldsymbol{x}^{\mathrm{p}}, \boldsymbol{z}^{\mathrm{g}}_i)^\top \boldsymbol{\Omega}^{\mathrm{g}}_i e^{\mathrm{g}}(\boldsymbol{x}^{\mathrm{p}}, \boldsymbol{z}^{\mathrm{g}}_i) \\ &+ \sum_i e^{\mathrm{l}}(\boldsymbol{x}^{\mathrm{p}}, \boldsymbol{x}^{\mathrm{l}}, \boldsymbol{z}^{\mathrm{l}}_i)^\top \boldsymbol{\Omega}^{\mathrm{l}}_i e^{\mathrm{l}}(\boldsymbol{x}^{\mathrm{p}}, \boldsymbol{x}^{\mathrm{l}}, \boldsymbol{z}^{\mathrm{l}}_i) \\ &+ \sum_k \mathcal{F}^{\mathrm{map}}_k(\boldsymbol{x}^{\mathrm{l}}, m_k), \end{aligned} \tag{8}$$

with error functions for GNSS e^{g}, odometry e^{o}, landmark measurements e^{l}, landmark observations to map associations $\mathcal{F}^{\mathrm{map}}_k$ and the information matrices $\boldsymbol{\Omega}_i$ for each measurement. The error functions are described in detail by Grisetti et al. [**?**], for example. We now interpret finding the most likely vehicle pose as denoted in (1) as minimizing a sum of weighted errors. In contrast to the particle filter, the optimization approach searches for a single optimized estimate rather than maintaining a set of weighted hypotheses.

We include knowledge about our map by matching landmark observations to the map upfront. See Wilbers et al. [**?**] for a description of the data association process. The state of every associated landmark is globally constrained by

$$\mathcal{F}^{\mathrm{map}}_k(\boldsymbol{x}^{\mathrm{l}}, m_k) = e^{\mathrm{m}}(\boldsymbol{x}^{\mathrm{l}}, m_k)^\top \boldsymbol{\Omega}^{\mathrm{map}}_k e^{\mathrm{m}}(\boldsymbol{x}^{\mathrm{l}}, m_k), \tag{9}$$

$$e^{\mathrm{m}}(\boldsymbol{x}^{\mathrm{l}}, m_k) = \boldsymbol{x}^{\mathrm{l}} - m_k, \tag{10}$$

with map information matrices $\boldsymbol{\Omega}^{\mathrm{map}}_k$. The map information matrix represents the quality of each map landmark. In case it is unknown, an educated guess must be made. By globally constraining the landmarks we implicitly constrain the vehicle poses inside the sliding window. This allows us to globally localize the vehicle even if no GNSS is available.

The optimization problem (8) is nonlinear and is solved iteratively. A common technique is to use the Gauss-Newton or Levenberg-Marquardt algorithms. In both the error function (8) is linearized around the current estimate $\breve{\boldsymbol{x}}$ and transformed into the linear system

$$\boldsymbol{H}(\breve{\boldsymbol{x}})\Delta\tilde{\boldsymbol{x}}^* = -\boldsymbol{b}(\breve{\boldsymbol{x}}), \tag{11}$$

with system matrix $\boldsymbol{H}$ and state vector $\boldsymbol{b}$ from which we compute the optimal iterative update $\Delta\tilde{\boldsymbol{x}}^*$. For a detailed derivation of $\boldsymbol{H}$ and $\boldsymbol{b}$ we refer to Grisetti et al. [**?**].

A conceptual difference of our approach to common implementations is to neglect any marginalization. On the one hand, this allows us to avoid undesired fill-in in the system matrix $\boldsymbol{H}$, and on the other hand we prevent the accumulation of errors induced by falsely associated landmarks.

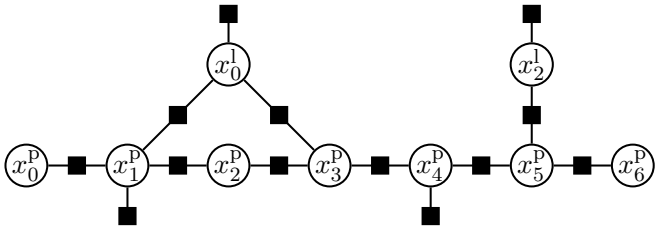


Fig. 2: Factor graph representation of the localization problem.

C. Resource-adaptive localization

We turn our attention to the question what size of state vector we can allow ourselves to choose. For both the particle filter and the graph-based localization, it is generally beneficial for the localization accuracy to augment the state vector if possible. In case of the particle filter, this is the number of particles that we choose to represent the posterior. For the graph-based localization, it is the length of the sliding window. We cannot set these values arbitrarily large as we have finite computational resources and need to deliver pose estimates at a fixed frequency f. As we deploy our localization algorithms in multiple vehicles with different hardware and software configurations, it is cumbersome to determine beforehand for each vehicle separately how many computational resources will be available at runtime. Even if we did this, it would only be an estimate of the actual available resources because they change over time: other software runs on the same hardware and will temporarily demand more or less CPU time, leaving our algorithms with a time-varying amount.

These resource constraints translate to the challenge that it takes different amounts of time to solve the same problem on different hardware or when the available CPU time fluctuates over time. However, we aim to exploit all available resources if possible to achieve the best localization accuracy. Thus, our approach is to adapt the size of the state vector at runtime by measuring how long the computation took in the current cycle and derive from that the size of the state vector for the next cycle [**?**]. We simplify this problem by assuming linear runtime complexity in the size of the state vector for both our algorithms. For the particle filter, this is exactly true, and for the graph-based localization, this is approximately true which we show empirically in our experiments.

Our solution consists in using the proportional-integral-derivative (PID) controller concept from linear control theory as e.g., applied by Li and Nahrstedt [**?**] in a more general domain. Let us explain this for the example of the particle filter, which can be directly applied to the graph-based localization, too. We create a feedback loop and measure the computation time c_t for the current time step and a given number of particles $\mathcal{M}_t$. Let $\Delta c_t = \frac{1}{f} - c_t$ be the difference between the nominally available and the measured

computation time. The PID controller sets

$$\mathcal{M}_{t+1} = \mathcal{M}_t + K_\mathrm{p}\Delta c_t + K_\mathrm{i}\sum_{i=1}^{t}\frac{\Delta c_i}{f} + K_\mathrm{d}(\Delta c_t - \Delta c_{t-1})f \tag{12}$$

where K_p, K_i, and K_d are proportional, integral, and derivative gain constants. These are empirically determined and left constant during the runtime. The goal is to obtain $\Delta c_{t+1} \approx 0$.

D. Deterministic vs. stochastic algorithm

Determinism is the property of an algorithm to always produce the same output given a fixed input. In the context of localization, this means that deterministic algorithms compute the same trajectories if they are fed with the same stream of input data. Stochastic algorithms are non-deterministic as they involve some form of randomness.

Particle filter localization is a stochastic algorithm as it is fundamentally based on randomly drawing samples from a proposal distribution. The two steps motion update and importance weighting are usually deterministic, but the resampling step introduces randomness.

Graph-based localization relies on optimizing a set of constraints that are derived from the input data. The conventional probabilistic interpretation of this step is that the maximum likelihood of the set of robot poses is determined. However, this probabilistic notion does not imply that it is a stochastic algorithm. In fact, it is deterministic and produces the same results if it is run repeatedly on the same data.

In the context of automated driving, localization plays a safety-critical role as driving maneuvers are based on the pose of the vehicle relative to the map. Therefore, functional safety requirements need to be fulfilled. This is usually significantly harder—if not impossible—for stochastic than for deterministic algorithms which is why the graph-based approach is preferable.

E. Unimodal vs. multimodal

Particle filter and graph-based localization differ in how they represent their belief about their estimated poses. The particle filter tracks a nonparametric distribution over time. It can represent multimodal beliefs that are of non-Gaussian nature. This is useful in situations of high ambiguity due to perceptual aliasing or self-similar environments, for example. The algorithm relies on having sufficient and well-distributed samples to represent the belief about the current pose.

In graph-based localization under the conventional probabilistic interpretation, the graph infers the set of most likely poses given the measurements. As a graph is usually built up by multiple poses, this is technically a multimodal Gaussian distribution. However, each pose corresponds to a single Gaussian that is being estimated. Therefore, the expressiveness about the belief of the pose at a given point in time is higher for particle filters, as they are not restricted to parametric or Gaussian distributions. Depending on the use case and subsequent software modules for which localization is required both approaches are feasible.

F. Integration of outdated measurements

The integration of out-of-sequence or delayed measurements is of practical importance for most state estimation techniques. Out-of-sequence measurements describe data that comes in the wrong order, i.e., newer data from that input source has already been processed. Delayed measurements refers to data that arrives later than usual and after the state estimation has processed the data for the corresponding time frame. Both types of behavior can be caused by sensors, faulty network transmission, or sensor preprocessing and have a similar effect on the state estimation process.

Particle filters struggle to integrate data from timestamps that have already been processed. They would have to propagate their state back in time with some form of inverse motion update (retrodiction), apply the measurement by performing the importance weighting, and subsequently resample and perform all other steps up to the current point in time. Since the resampling step is non-deterministic, the particle distribution at the point in time of the delayed measurement is different to what it was at that point in time. Also, the retrodiction steps are computationally expensive as repeated importance weighting and resampling steps have to be performed. An alternative implementation could make use of a history of the particle sets in the past, thus sacrificing memory for computational speed. Usually, delayed and out-of-sequence measurements are simply discarded due to these complications.

Graph-based approaches intrinsically keep some outdated poses as nodes within their graph. As long as the delayed or out-of-sequence measurements fall within the sliding window of the graph, it is therefore straightforward how to integrate this information. It is simply a matter of adding new nodes and edges that reflect these measurements, and the remaining graph structure stays untouched. Therefore graph-based methods are preferable over particle filters when integrating out-of-sequence or delayed measurements.

G. Estimating old poses

Graph-based approaches maintain a recent history of poses within the state estimation problem while particle filters only maintain the current pose. However, as older and newer poses are strongly linked to each other, we argue that it can be beneficial to finetune the past poses in order to estimate the more recent poses with higher accuracy.

The sliding window graph contains information about the last estimated poses and their relations with the map and the measurements. They are still being optimized for and thus influence the estimate of the current pose. If newer information indicates that, for example, previously unlikely seeming data associations now seem likely, they can simply be added; a method known as *delayed data association*. Moreover, frequent relinearization seeks to minimize the relinearization error. This thus improves our estimate about the past and of the present simultaneously. In fact, we can even make use of the history within the graph to optionally output a *lagged pose*, i.e., a pose with a certain delay compared to the current timestamp. This pose is obtained

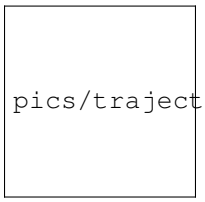


Fig. 3: Map and trajectory of our real-world dataset. All pole-like landmarks in the test area are denoted in blue. The 16 km trajectory is denoted in red. It contains velocities between 0 km/h and 70 km/h and covers typical urban scenarios like heavy and light traffic, different street types, and different junction sizes.

from one of the older nodes in the graph. It represents our current best estimate of the past. Because we do not use marginalization, nodes at the center of the sliding window are constrained the most and their estimated pose is thus of the highest accuracy. Outputting a lagged pose can therefore be beneficial for applications that can cope with older pose estimates in favor of even higher estimation accuracy.

Particle filters never change their past beliefs about the robot poses. They only track the current set of particles. It could be beneficial for them to finetune their past beliefs to achieve a better representation of the pose at the current time step. This would be useful after the set of particles deviated strongly in subsequent importance weighting steps from the trajectory that pure motion updates would have dictated. Such behavior indicates that either the odometry data was off or that the past beliefs did not well represent the actual pose. In the latter case, adapting the belief in the past could lead to a more reasonable distribution of particles in the present. This would allow for previously unlikely data associations, for example. However, it is unclear how to exactly achieve this kind of retrodiction in an optimal fashion.

H. Ease of implementation

For the practictioner topics like ease of implementation, software maintainability, ease of debugging and inspection etc. play an important role. Though, these points are to a certain degree subjective which makes it hard to judge them objectively. Still, these are important topics and we think it is safe to say that it is generally less complex to implement a particle filter than a graph-based system. However, note that many libraries ease the implementation of graph-based systems substantially, e.g., g2o [**?**] or GTSAM [**?**].

IV. Experimental Evaluation

Our experiments are designed to serve as a comparison between the particle filter and graph-based localization approaches. Apart from our theoretical discussion in Sec. III, we investigate three key aspects in the context of automated driving: (i) accuracy, (ii) adaptive behavior in terms of computational resources, and (iii) benefit of estimating old poses.

We conduct a set of experiments in which we vary the size of the state vector. In case of the particle filter we directly control the state vector size by setting the number of particles, whereas for the graph-based localization the state vector size is only partly influenced by controlling the

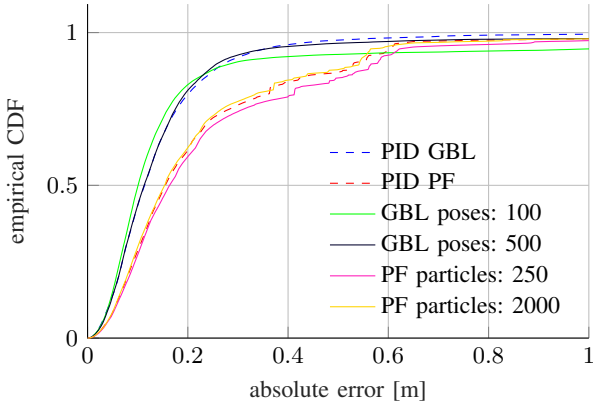


Fig. 4: Empirical cumulative distribution function (CDF) for a set of experiments, comparing particle filter (PF) vs. graph-based localization (GBL).

number of poses. In the following, we study the effects and focus on accuracy and computation time. The experiments are based on a 16 km long dataset from a real prototype vehicle. The dataset of this evaluation is based on Velodyne VLP-16 to detect pole-like landmarks, wheel-tick and IMU based odometry, and a low-cost GNSS receiver. The trajectory of our dataset is depicted in Fig. 3. Our reference system is a high-frequency and post-processed Real Time Kinematic (RTK) system with a location accuracy specification of 2 cm. We interpolate the reference trajectory to the timestamps of the resulting trajectories to compute the errors.

A. Accuracy

Our first set of experiments compares the accuracy of both localization approaches. For clarity, Fig. 4 shows the empirical cumulative error distributions (CDF) only for a selected number of experiments. The chosen experiments represent the lower and upper bound of the CDFs for each method. Our experiments show that in terms of accuracy the graph-based localization (GBL) is superior to the particle filter (PF) as the CDF accumulates faster. In both localization approaches it is necessary to maintain at least a minimum number of the states. This can be seen in the experiment with 250 particles and respectively for the graph-based approach with a sliding-window of 100 poses. Both are less accurate than the experiments with the greater state vector size. Additionally, Fig. 4 shows that our resource-adaptive PID controller with a variable state vector size delivers comparable accuracy results.

Tab. I shows the accuracy in terms of average trajectory errors. The errors reflect that the PID controller works in terms of accuracy for both approaches and achieves comparable accuracy.

B. Runtime behavior and PID controller

Our second experiment is designed to show the relation between computation time and the size of the state vectors. Fig. 5 compares the runtime behavior between experiments with a fixed number of particles and poses. The average computation time for each experiment is denoted as a red cross, with the single standard deviation in black lines. Both

	Experiment	abs.	long.	lat.	heading
GBL	100 poses	0.221 m	0.136 m	0.143 m	0.49°
	500 poses	0.168 m	0.116 m	0.096 m	0.39°
	PID (1104)	0.150 m	0.111 m	0.078 m	0.34°
PF	250 particles	0.270 m	0.196 m	0.139 m	1.46°
	2000 particles	0.237 m	0.183 m	0.105 m	1.38°
	PID (2160)	0.240 m	0.176 m	0.119 m	1.42°

TABLE I: Absolute, longitudinal, lateral, and heading errors for different configurations during a 16 km urban drive. For the PID experiments we denote the average number of particles and poses in the graph in brackets.

approaches show a linear relation between computation time and control variable. For the graph-based approach we rely on the highly tuned g2o [**?**] framework for optimization and apply sparse block matrices together with Cholesky decomposition, which improves the runtime performance.

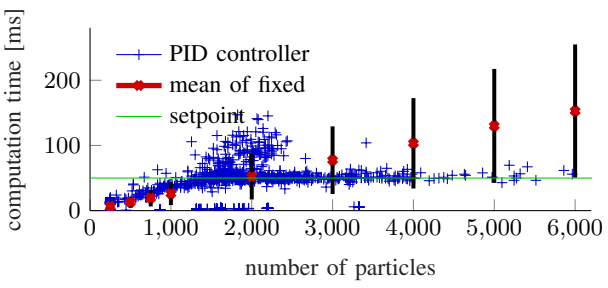
Additionally, Fig. 5 shows that our PID controller manages to satisfy timing requirements by adjusting the number of particles and number of poses in the graph. In both cases our PID approach successfully provides pose estimates around the set target frequency (setpoint). It exploits the complexity of the problem, which depends on the number of visible landmarks in each timestamp. When only a small number of landmarks is visible, the complexity decreases and the size of the state vector is increased. We show this behavior in Fig. 6 for the graph-based localization. It shows that the computation time for the experiments with a fixed number of poses depends roughly linearly on the number of edges in the graph. As the number of measurements itself is not constrained, the number of measurements in the graph depends on the environment structure. Due to our sliding window approach, in which all poses have the same temporal distance, the number of poses implicitly controls the number of measurements in the graph. Our PID controller exploits this fact and successfully provides pose estimates with the set target frequency.

C. Estimating old poses

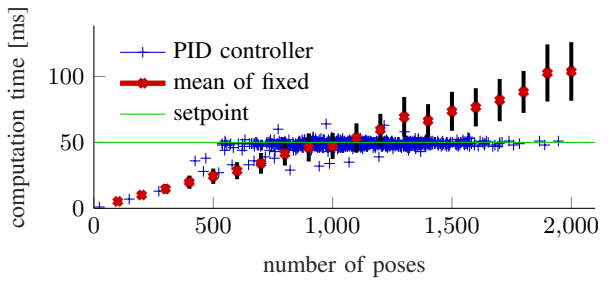
Our third experiment demonstrates the benefit of optimizing past poses in our graph-based approach. In contrast to the particle filter, estimating the trajectory within the sliding window is part of the graph-based optimization. Fig. 7 shows box plots for the Euclidean errors within the trajectory of the sliding-window graph. The accuracy in the middle of the graph is better than at the head and tail. The reason for this effect is that the poses in the middle of the graph are more constrained than the poses at the ends of the graph. The effect at the tail of the graph is caused by using truncation instead of marginalization. Using lagged poses is especially beneficial for non-timing-critical applications, for which a more accurate but lagged pose is helpful.

V. Conclusion

In this paper we provided an argumentative and experimental comparison between particle filter and graph-based localization for automated vehicles. On the one hand, we argued that particle filters have the advantage of being able to represent multimodal distributions and are generally easier



(a)



(b)

Fig. 5: Relation between computation time and control variable. (a) For the particle filter, we control the number of particles. (b) For graph-based localization, we control the number of poses.

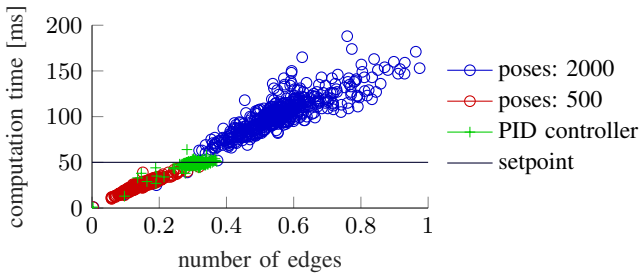


Fig. 6: Relation between computation time and the number of edges in the graph-based localization. The number of edges represents the number of measurements used in the graph.

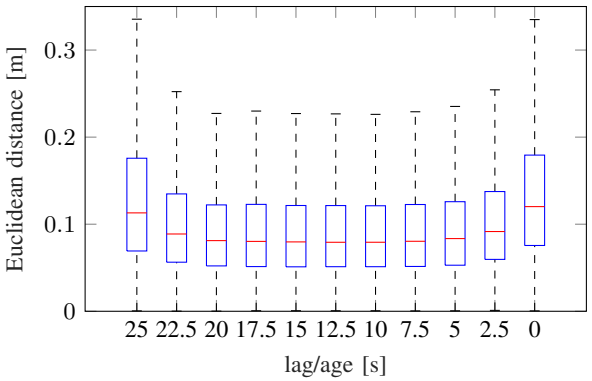


Fig. 7: Accuracy within the trajectory of a graph. A lag of 0 s represents the most recent pose at the head of the graph, whereas a lag of 25 s denotes the oldest pose at the tail of the graph. This plot corresponds to a graph with 500 poses and a frequency of 20 Hz.The figure shows that the most accurate poses are in the middle of the graph.

to implement. On the other hand, graph-based approaches as non-stochastic algorithms can fulfill functional safety requirements more easily. Moreover, their sliding window of the past allows them to easily integrate outdated measurements, perform delayed data association, and optionally output a lagged pose. We have experimentally shown that the graph-based localization achieves a higher accuracy than the particle filter, that outputting a lagged pose allows us to trade estimation age versus accuracy, and that a PID controller is able for both approaches to regulate the computation time to not exceed a predefined setpoint.

Depending on the importance of these criteria, practitioners can choose the best algorithm for their application. In general we recommend using graph-based approaches as they have proven to be of higher accuracy, which is usually the most important property for localization. Most SLAM approaches have switched from adopting the particle filter to the optimization paradigm, and we believe that this is similarly beneficial for many localization systems.