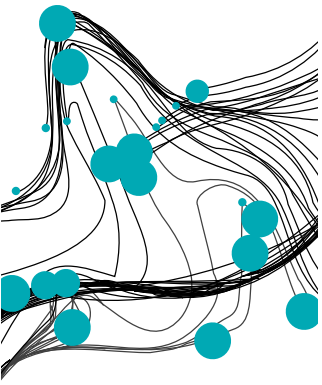
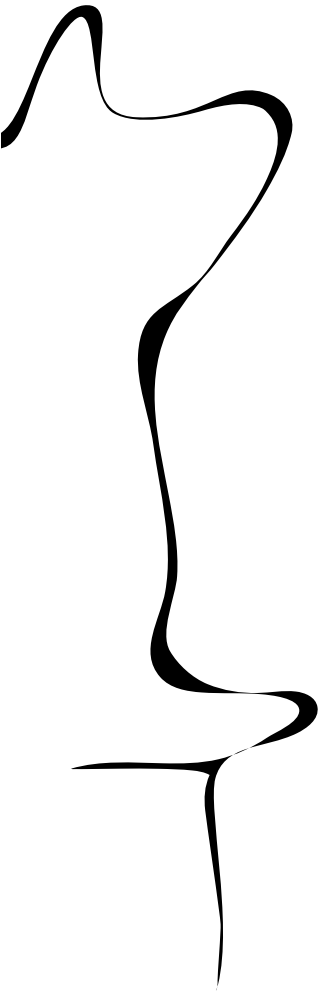


RAM ROBOTICS AND MECHATRONICS



PERFORMANCE EVALUATION OF SEVERAL SLAM ALGORITHMS IN A FEATURE-BASED VSLAM FRAMEWORK

G.J. (Gideon) Kock

MSC ASSIGNMENT

Committee:
dr. ir. F. van der Heijden
dr. F.J. Siepel, MSc
dr. M. Poel

May, 2021

031 RaM2021
Robotics and Mechatronics
EEMathCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE
UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Content

Summary

The aim of the HAPI project is to enhance a handheld device, such as the Laser Speckle Contrast perfusion imager, that scans a target area by manually pointing the device at the diseased skin area. To provide 3D geometrical information, visual SLAM (vSLAM) algorithms are involved.

In this work, several SLAM algorithms are chosen that are suitable to perform in a feature-based vSLAM framework. First, different types of Bayesian-based filters were presented. Based on these filters, four common SLAM algorithms were selected: EKF SLAM, SEIF SLAM, Graph SLAM and FAST SLAM. The working principle of the algorithms is explained in this report, including a mathematical derivation.

The performance of these algorithms is measured using an experiment in a 2D environment. Furthermore, the algorithms are tested in a feature-based vSLAM framework in a 3D environment. The input data used in these experiments consist of feature points extracted from real images and simulated feature points.

During the experimentation, it is observed that Graph SLAM provides a relatively high accuracy and consistency. However the algorithm is computationally expensive and therefore it could not be run in real-time. The results of the experiments shows that EKF and SEIF have almost the same accuracy. The SEIF algorithms is computationally more attractive, but the needed computation power was higher than expected. It is also shown that the sparsification step in SEIF ensures a low constancy of the filter. Furthermore, the FAST-SLAM algorithm is only tested on the 2D data and shows a relatively high error in the pose and the map.

The research shows that there are several SLAM-back ends that can perform in a feature-based framework. Depending on the needed accuracy, the available computation power, the available knowledge to implement and the need to perform in real-time, a SLAM back-end can be chosen. However, to measure the performance in a more realistic environment, the algorithms need to be tested in more nonlinear situation with more loop closings.

1. Introduction

The HAPI project is a TTW-funded project in which the Radboud UMC and the University of Twente collaborate. The aim of this project to develop a handheld device for blood perfusion imaging, and to deploy this system for monitoring the progression of Psoriasis. The technical development of this system is accomplished by the BMPI group. In addition, the RAM group participates in this project by providing the needed computer vision functionality.

The goal of the work package, for which RAM was responsible, is to provide methods for real-time estimation of 2D in-plane motion, e.g. optical flow. Next to this, the work package also entails an explorative study to the possibility of 3D vision to enhance the functionality of the handheld device. The current study is a continuation of this last aspect of the work package.

1.1 Motivation

The aim of the project is to enhance a handheld device, such as the Laser Speckle Contrast perfusion imager. The device scans a target area by manually pointing the device at the diseased skin area. It is desired to enlarge the field of view of the device by slowly sweeping it along the skin surface in a controlled but flexible manner. Also it has to provide 3D geometrical information of the target object in order to correct geometrical effects. In other words it has to correct non-fronto-parallel imaging. The 3D geometrical information can also be used to facilitate 3D augmented reality. Furthermore, to enable full motion artefact compensation, it must provide 3D motion information.

To provide this information, visual SLAM algorithms have to be involved. A SLAM (Simultaneous Localization And Mapping) algorithm can construct a map of an unknown environment moving around with a sensor system, and while simultaneously keeping track of the location of this device within the map. Visual SLAM (vSLAM) algorithms use camera information as input information.

Various principles of vSLAM exist [1]. In a feature-based method a set of feature points is extracted from an image. These feature points are matched with feature points extracted from other images that are taken from different perspectives. The matching process is performed by matching the so-called descriptors of the corresponding feature points. With points seen from different perspectives, 3D information can be calculated.

In direct methods, the entire images are compared to each other. This is done by matching parts of images. Instead of abstracting features points and descriptors, they use the image directly. Whereas geometric consistency is used for feature-based methods, photometric consistency is used for direct methods.

Both of the approaches might be of interest, but as a first step this study addresses only the point feature based approach leaving the other approaches for future work. Within feature-based SLAM, various implementations exist with different characteristics. There are different ways to match features points of images. Also the 3D calculation can be performed in different ways. In this calculation noise has to be taken into account. Noise induces during image acquisition and propagates into calculation of 3D points from different perspectives. To work form a probabilistic perspective noise can be taken into account. The algorithm that solves this problem is called a SLAM back-end algorithm.

If feature-based SLAM is going to be used, it is not clear which implementation is most suitable for a handheld device.

1.2 Problem statement

The goal of this research is to find the best SLAM back-end for a handheld device in a feature based vSLAM framework. This framework is realized in previous work.

The following questions will be answered in this project:

1. Which SLAM back-ends are suitable for the mapping on a handheld device?
2. What are the working principles of these algorithms?
3. What is the performance of these algorithms within the vSLAM feature-based framework?

The following hypothesis will be verified:

- A high accuracy of the estimated 3D positions will be at the cost of computational complexity of an algorithm.

1.3 Outline of the report

The structure of this report is as follows. Chapter 2 describes the background information of SLAM. Furthermore, the mathematical variables and their notations used in the next sections are explained. In Chapter 3 different filters are explained. Using this knowledge, the SLAM algorithms to be researched are chosen. Also the theoretical background and working principle of the chosen SLAM algorithms are explained. The approach to the experiments is described in Chapter 4. The results of experiments are given in Chapter 5 and discussed in Chapter 6. In the last section the conclusions are made and recommendations are provided.

2. Background: SLAM

The term SLAM was already introduced in Section 1.1. When a system does not have access to a map of the environment, nor it does know its own state, it can be seen as a typical SLAM problem. A SLAM algorithm tries to find the right path the system is taking (localization). Simultaneously, it creates a map of the environment in which the device is located (mapping) using models and measurements.

There is always uncertainty in measurements and models. For example, due to measurements noise and approximations in the model. Therefore, a SLAM algorithm is working from a probabilistic perspective, where it solves a so called optimization problem.

The core commonality mathematical framework of SLAM algorithms is Bayesian-based. A Bayesian-based filter handles different kind of measurements of the sensor device. Measurements from different sensors can be divided in two categories, namely relatives and absolute measurements. Examples are given in Table 1. In a Bayesian-based filter, a prediction of the position of a sensor device is performed using a kinematic model and/or information from relative kinematic measurements, for example speed and turning rate. The next step of a Bayesian-based filter is the measurement update. In this step, absolute measurements are used to make the predicted state more accurate. Absolute position measurements are measurements of the environment relative to the device.

Table 1 Measurement types

Measurements type	Examples
Relative measurements	Wheel encoders
	Integration of accelerators or gyroscopes
Absolute measurements	Beacons
	Laser scanner
	Vision systems

2.1 Explaining variables using an example

A graphical interpretation is provided in Figure 1, with the definitions of the variables in the legend. These variables will also be used in other sections of this report. As described in the legend, x_i is the state of the device at time i and m_j is the position of a landmark with ID j. The variable u_i defines the relative measurements at time i. The k-th absolute measurement at time i is represented by the variable z_i^k, with c_i^k the corresponding landmark ID. A variable indicated with a capital letter, signifies the maximum number of that variable. For example, $z_i^{1:K}$indicates the whole set of measurements at time i.

Figure 1 shows that the estimated path deviates from the real path. At time i the sensor device, has a state x_i. Using the previous state $x_{i=1}$ and relative measurements $u_{i=2}$, a new state $\bar{x}_{i=2}$ is predicted. Note that the upper bar indicates an predicted state. At i=2 it observes two landmarks $m_{j=1}$ and $m_{j=2}$. Two measurements $z_{i=2}^{k=1}$ and $z_{i=2}^{k=2}$ are obtained, with ID numbers 1 and 2 stored in $c_{i=2}^{k=1}$ and $c_{i=2}^{k=2}$, respectively. Using this information the state $\bar{x}_{i=2}$ is updated to $x_{i=2}$. This process iterates at each time stamp. A prediction is performed to calculate $\bar{x}_i$ using the previous state $x_{i-1}$and relative measurements u_i. The measurements update is performed using absolute measurements $z_i^{1:K}$ with correspondences $c_i^{1:K}$ to calculate x_i.

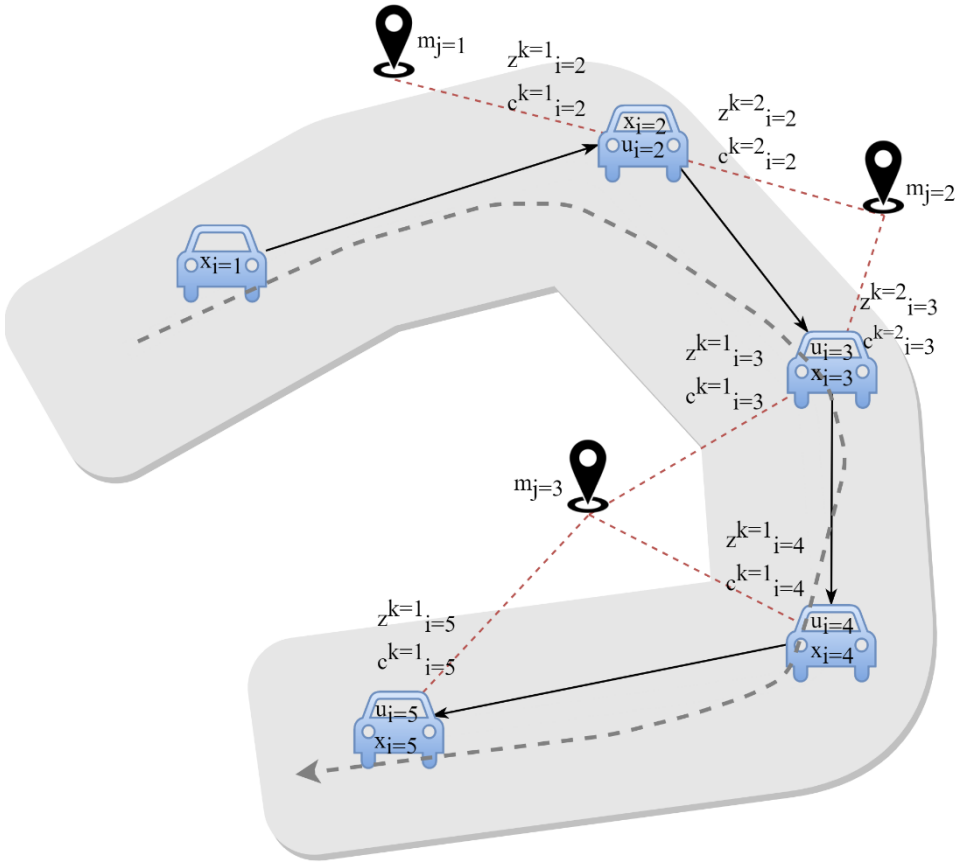


Figure 1 Graphical interpretation of solving a SLAM problem using a bayes filter

Legend

Symbol	Label	Description
	x_i	Estimated state at time i
	u_i	Relative measurements at time i
	m_j	Estimated state of landmark with id 'j'
	z_i^k	k^{th} measurement at time i
	c_i^k	Correspondence (landmark id) of k^{th} measurement at time i.
	-	Estimated path
	-	Real path

2.2 Online vs Full SLAM

From a probabilistic point of view there are two types of SLAM problems: online and full SLAM. Both calculate the posterior probability using measurements and controls, but online SLAM involves estimating the momentary pose, while in full SLAM the entire path is estimated. The information about the state of the system x (state vector) and the map m containing the landmarks are stored in a so called combined state vector y. An online SLAM algorithm only stores the current state x_i in the combined state vector, while in full SLAM all previous poses $x_{1:I}$ are also stored.

The corresponding probability functions and the layout of the combined state factor are given in Table 2.

Table 2 SLAM types

	ONLINE SLAM	**FULL SLAM**
Probability function	$p(x_i, m_{1:J} \mid u_{1:I}, z_{1:I}^{1:K}, c_{1:I}^{1:K})$	$p(x_{0:I}, m_{1:J} \mid u_{1:I}, z_{1:I}^{1:K}, c_{1:I}^{1:K})$
Combined state vector	$y_i = \begin{pmatrix} x_i \\ m_{1:J} \end{pmatrix}$	$y_i = \begin{pmatrix} x_{1:I} \\ m_{1:J} \end{pmatrix}$

3. The family of feature-based SLAM

As described earlier, the core commonality mathematical framework of SLAM algorithms is Bayesian-based. Bayes filters can be implemented in different ways. The algorithms can be divided in two main categories: Gaussian Filters and nonparametric filters.

3.1 Gaussian filters

Gaussian techniques all share the basic idea that beliefs are represented by multivariate normal distributions (Figure 2). There are two common Gaussian filters [2].

3.1.1 Kalman filter

The most common Gaussian filter is the Kalman filter, where a Gaussian is defined with moments parameterization: a mean vector μ and covariance matrix Σ.

The general form of a Gaussian function using moments parameterization is given in (1).

$$p(x) = \frac{1}{\sqrt{(2\pi)^D|\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right) \quad (1)$$

where x is a random vector with dimension D.

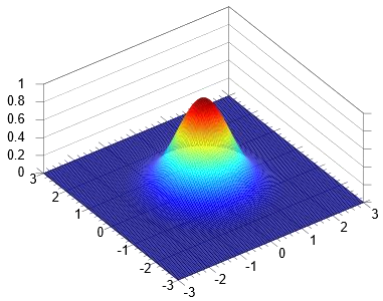


Figure 2 Two dimensional Gaussian distribution (D=2) [3]

3.1.2 Information filter

Another Gaussian filter is the information filter, the dual of the Kalman filter. Instead of moments parameterization, a Gaussian is defined in canonical parameterization. The canonical form consists of a information matrix Ω and an information vector ξ. The relation between moments and canonical parameterization is the following:

$$\Omega = \Sigma^{-1} \quad (2)$$

$$\xi = \Sigma^{-1}\mu \quad (3)$$

The Gaussian in canonical parameterization is shown in (4), where the parts that are not dependent on x are subsumed to η.

$$p(x) = \eta \exp\left(-\frac{1}{2}x^T\Omega x + x^T\xi\right) \quad (4)$$

In information form, the probabilities are often represented by their negative logarithmic. The Gaussian in this form has some advantages. In particular, Bayes rule becomes an addition. Another advantage is that p(x) now has a quadratic term in x parameterized by Ω and the linear term in x is parameterized by ξ.

The negative logarithmic form becomes:

$$-\log p(x) = c + \frac{1}{2}x^T\Omega x - x^T\xi \quad (5)$$

Where the log of η becomes a constant value, labeled with 'c'.

3.1.3 Extended versions of the Information- and Kalman filter

In Bayes filters, a motion model is used to predict the next state and a measurement model is used for the update step. The filter explained above can be worked out straightforwardly if the measurement function and the kinematic model are linear. Unfortunately, in practice these functions are nonlinear. To deal with nonlinearities, the two filters are both extended with linearizing nonlinear functions using Taylor expansion. These filters are called the Extended Kalman filter and the Extended information filter.

The nonlinear functions of the motion model and measurement model are given in (6) and (7), respectively.

$$\bar{x}_i = g(u_i, x_{i-1}) + \varepsilon_i \tag{6}$$

$$z_i^k = h(y_i, k) + \delta_i \tag{7}$$

Symbols ε_i and δ_i are variables relating to motion and measurement noise, respectively.

To deal with this nonlinearity, the extended versions of the information- and Kalman filter include a linearization via Taylor series expansion. The Taylor series expansion of the motion and measurement functions are given in (8) and (9), respectively.

$$\bar{x}_i \approx \underbrace{g(u_i, x_{i-1})}_{\tilde{x}_i} + G_i(x_{real,i-1} - x_{i-1}) + \varepsilon_i \tag{8}$$

$$z_i^k \approx \underbrace{h(y_i, c_i^k)}_{\tilde{z}_i^k} + H_i^k (y_{real,i-1} - y_{i-1}) + \delta_i \tag{9}$$

where the Jacobian G_i is the derivative of $g(u_i, x_{i-1})$ with respect to x_{i-1}. In (9) the Jacobian H_i^k is the derivative of $h(y_i, k)$ with respect to y_i. The curly embraced part by $\tilde{x}_i$ in (8) and the curly embraced part by $\tilde{z}_i^k$ in (9) are the noise free components and can be calculated. The parts between brackets after the Jacobians of both functions are the estimation errors in x_i and y_i. In practice the real values $x_{real,i}$ and $y_{real,i}$ are not known. However both filters are using the Jacobians to approximate the noise.

3.2 Nonparametric filters

The other category, nonparametric filters, are not based on parameters like the mean and the covariance matrix to describe uncertainty. Instead, they approximate posteriors by a finite number of values, each roughly corresponding to a region in state space. An advantage of nonparametric filter is that a probability can have all kinds of shapes. In this section the representation of two nonparametric filters are described. They are both using linearized motion and measurement functions described above. The advantage of nonparametric filters is that they can handle high non-linearities. A disadvantage is the required computational power.

3.2.1 Histogram filter

Histogram filters decompose the probability into finitely many regions and represent a probability for each region by a single probability value. In Figure 3 a graphical interpretation can be seen, where p(x) is a certain probability, which is described using a histogram.

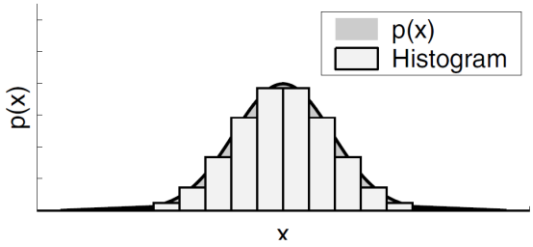


Figure 3 The histogram representation of a probability [4]

3.2.2 Particle filter

The key idea of the particle filter is to represent a probability by a set of random state samples. The denser a region particle, the higher the probability. A graphical interpretation is given in Figure 4.

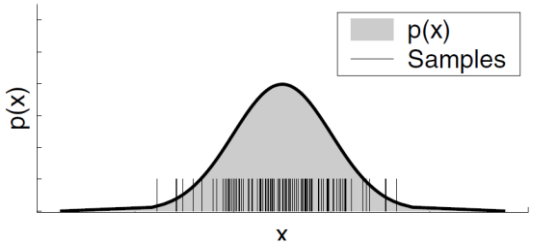


Figure 4 The particle representation of a probability [4]

3.3 Chosen SLAM algorithms

There are several SLAM algorithms based on the above mentioned filters. In this project it is not decided yet if the estimation has to be operated in real-time. Therefore both online and full SLAM can be chosen. The modeling for a human body does not have to necessarily be in real time. It can also acquire data in real time and analyze this data afterwards. There already exists a vSLAM algorithm which uses EKF SLAM (based on the Kalman filter) [5]. Therefore, this SLAM as well as alternatives will be used in this research. In vSLAM a large number of landmarks are desired to create a dense map. Therefore SLAM algorithms based on the information filter could be computationally interesting. Another interesting SLAM algorithm called FAST-SLAM is found. Since it is based on the particle filter, it can deal with high nonlinearities. An overview of the chosen algorithms is showed in Table 3.

Table 3 Chosen algorithms

Filter	Algorithm	Type
Kalman filter	EKF SLAM	Online
Information filter	SEIF SLAM	Online
	Graph SLAM	Full
Particle filter	FAST-SLAM 2.0	Online

3.4 EKF SLAM

EKF SLAM is one of most common SLAM algorithms. Since this algorithm is already used in the visual SLAM framework and explained in previous work[5], a short explanation is given in this report.

3.4.1 Prediction

In the prediction step the mean $\bar{\mu}_i$ and the associated covariance matrix $\bar{\Sigma}_i$ are calculated in (10) and (11), respectively. They are calculated using the linearized motion function $g(u_i, \mu_{i-1})$ with Jacobian G_i. More over linearizing is explained in section 3.1.3.

$$\bar{\mu}_i = g(u_i, \mu_{i-1}) \tag{10}$$

$$\bar{\Sigma}_i = \breve{G}_i \Sigma_{i-1} \breve{G}_i^T + F_x^T R_i F_x \tag{11}$$

where R_i is the covariance matrix related to motion noise. $R_i = V_i M_i V_i^T$ is the motion noise, with covariance matrix M_i of the relative measurements u_i and the Jacobian V, the derivative of the motion function with respect to the relative measurements. The Jacobian V ensures an approximate mapping between the motion noise in control space to the motion noise in state space. The projection matrix F_x maps the motion update to the state vector. The projection matrix has the structure given in (12).

$$\mathrm{F_x} = \Big[\underbrace{\mathrm{I}}_{\text{size of } x_i} \quad \underbrace{0}_{\text{size of } m_{1:J}}\Big] \} \text{ size of } x_i \tag{12}$$

With I an identity matrix and 0 a zero matrix.

The matrix $\breve{\mathrm{G}}_i$ in (11) is the Jacobian $\mathrm{G_i}$ but extended with zeroes to have the same size of the combined state vector. This is done using the projection matrix $\mathrm{F_x}$:

$$\breve{\mathrm{G}}_i = \mathrm{F_x^T G_i F_x} \tag{13}$$

3.4.2 Measurement update

For each landmark the Kalman matrix $\mathrm{K_i^k}$ is calculated to update the predicted mean $\bar{\mu}_i$ and covariance matrix $\bar{\Sigma}_i$.

$$\mathrm{S_i^k} = \breve{\mathrm{H}}_i^k \bar{\Sigma}_i \breve{\mathrm{H}}_i^{k\mathrm{T}} + \mathrm{Q_i^k} \tag{14}$$

$$\mathrm{K_i^k} = \bar{\Sigma}_i \breve{\mathrm{H}}_i^{k^{\mathrm{T}}} \mathrm{S_i^{k^{-1}}} \tag{15}$$

$$\mu_i = \bar{\mu}_i + \mathrm{K_i^k}\left(\mathrm{z_i^k} - \hat{\mathrm{z}}_i^k\right) \tag{16}$$

$$\Sigma_i = \bar{\Sigma}_i - \mathrm{K_i^k S_i^k K_i^k} \tag{17}$$

where $\mathrm{S_i^k}$ is the so-called innovation matrix, $\mathrm{K_i^k}$ is the so-called Kalman gain and $\mathrm{Q_i^k}$ the covariance matrix related to measurement noise. Note that predicted variables are indicated with a upper bar while the updated variables after measurement are not.

Equal to the Jacobian in the prediction, the Jacobian $\breve{\mathrm{H}}_i^k$ is the large matrix fitted to the size of the combined state vector. This in done in the following way:

$$\breve{\mathrm{H}}_i^k = \mathrm{F_x^T H_i^k F_x} \tag{18}$$

With $\mathrm{F_{x,j}}$ having the following shape:

$$\mathrm{F_{x,j}} = \begin{bmatrix} \underbrace{\mathrm{I}}_{\text{size of } x_i} & \underbrace{0}_{\text{size of } m_{1:j-1}} & \underbrace{0}_{\text{size of } m_j} & \underbrace{0}_{\text{size of } m_{j+1:J}} \\ \underbrace{0}_{\text{size of } x_i} & \underbrace{0}_{\text{size of } m_{1:j-1}} & \underbrace{\mathrm{I}}_{\text{size of } m_j} & \underbrace{0}_{\text{size of } m_{j+1:J}} \end{bmatrix} \begin{matrix} \} \text{ size of } x_i \\ \} \text{ size of } m_j \end{matrix} \tag{19}$$

Another way to do the measurement update is to build the Jacobian $\breve{\mathrm{H}}_i$ for all measurements at once. In this way only one Kalman gain $\mathrm{K_i}$ has to be calculated. Note that also three matrices have to be constructed for each of $\mathrm{Q_i^{1:K}}$, $\mathrm{z_i^{1:K}}$ and $\hat{\mathrm{z}}_i^{1:K}$.

3.4.3 Complexity

According to [2], the calculation of the prediction step consists of the next state (10) and the covariance matrix (11). These operations require $O(1)$ and $O(n)$, respectively, with n the number of landmarks. The measurement update, consisting of calculating the innovation matrix (14), Kalman gain (15), updated main (16), and update covariance matrix (17), requires $O(k^3)$, $O(nk^2)$, $O(n)$, $O(n^2k)$ respectively, with k the number of measurements. Calculating a Jacobian requires $O(n)$. Therefore the computational complexity of the update step is $O(n^2)$ and the total cost is known to be $O(n^3)$. The memory usage of EKF is $O(n^2)$.

3.5 SEIF SLAM

SEIF is an online SLAM algorithm based on the information filter [2]. It has a lot in common with EKF, but since the representation is different, some calculations are computational more effective, while in other steps the computation is more complex. The part where it deviates the most from EKF is the sparsification step. This step makes SEIF a very efficient algorithm and therefore it can be performed in real-time. A flow diagram of SEIF is shown in Figure 5.

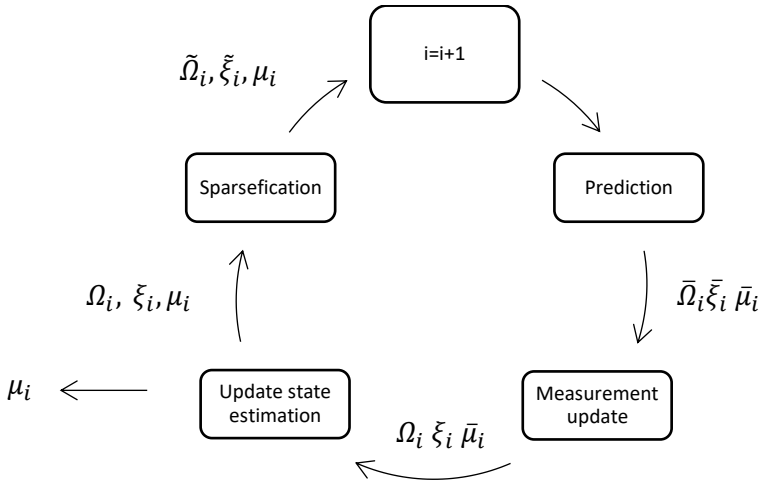


Figure 5 SEIF steps

In the prediction step, a new state is estimated using the motion model. This includes an information matrix $\bar{\Omega}_i$, an information vector $\bar{\xi}_i$ and a predicted mean $\bar{\mu}_i$. In the measurement update, the information matrix and vector are updated using absolute measurements. The mean is updated in the next step, called 'Update state estimation', using the updated information matrix and vector. Next, the sparsification is computed. In this step, the information matrix is made sparse without losing too much information. A sparsefied matrix or vector is indicated with a tilde above the expression. After the sparsification, the information matrix $\tilde{\Omega}_i$, information vector $\tilde{\xi}_i$ and the mean μ_i are up-to-date for time i. The next state can be calculated by repeating the above described process. In this chapter every step will be explained in detail.

3.5.1 Prediction

The predicted mean is equal to EKF. This equation is stated in (20). The calculation of the information matrix Ω in SEIF (21) is also based on the calculation of the covariance matrix Σ in EKF.

$$\bar{\mu}_i = g(u_i, \mu_{i-1}) \tag{20}$$

$$\begin{aligned} \overline{\Omega}_i = \overline{\Sigma}_i^{-1} &= (\breve{G}_i \Sigma_{i-1} \breve{G}_i^T + F_x^T R_i F_x)^{-1} \\ &= (\breve{G}_i \Omega_{i-1}^{-1} \breve{G}_i^T + F_x^T R F_x)^{-1} \end{aligned} \tag{21}$$

Equation (21) is computational expensive since it involves two inversions of large matrices. To make it computationally more attractive, the matrix inversion lemma is applied. If $\Phi_i^{-1} = \breve{G}_i\, {\Omega_{i-1}}^{-1} \breve{G}_i^T$, the equation is suitable for the inversion lemma:

$$\begin{aligned} \overline{\Omega}_i &= (\Phi_i^{-1} + F_x^T R F_x)^{-1} \\ &= \Phi_i - \Phi_i F_x^T \left(R^{-1} + F_x \Phi_i F_x^T\right)^{-1} F_x \Phi_i \end{aligned} \tag{22}$$

With

$$\begin{aligned} \Phi_i &= \left(\breve{G}_i\, {\Omega_{i-1}}^{-1} \breve{G}_i^T\right)^{-1} \\ &= \left(\breve{G}_i^T\right)^{-1} \Omega_i\, \breve{G}_i^{T-1} \end{aligned} \tag{23}$$

The inversion $R^{-1}$and $\left(R^{-1} + F_x \Phi_i F_x^T\right)^{-1}$ are inversions of low sized matrices. What remains is a large matrix $\breve{G}_i$and $\breve{G}_i^T$ that have to be inverted in (23). In the same manner as in (22), the matrix inversion lemma is used. In (24) the Jacobian is divided in an identity matrix I and the parts that characterize the change Δ_i. Then it is changed to the right form to apply the matrix inversion lemma afterwards.

$$\begin{aligned} G_i^{-1} &= \left(I + F_x^T \Delta F_x\right)^{-1} \\ &= \left(I - F_x^T I\, F_x + F_x^T I\, F_x + F_x^T \Delta F_x\right)^{-1} \\ &= \left(I - F_x^T I\, F_x + F_x^T (I + \Delta) F_x\right)^{-1} \\ &= I - F_x^T I\, F_x + F_x^T (I + \Delta)^{-1}\, F_x \\ &= I + \underbrace{F_x^T ((I + \Delta)^{-1} - I)\, F_x} \end{aligned} \tag{24}$$

Matrix $G_i^{T^{-1}}$ can be calculated in the same way. The only difference compared to (24) is the transpose of the curly embraced part.

To calculate the information vector ξ, equation (3) is rewritten to $\mu = \Sigma\, \xi$ and substituted into (20). Moving $\overline{\Omega}_i$ to the right hand side of the equation, results in the following equation:

$$\bar{\xi}_i = \overline{\Omega}_i\, (\, \Omega_i \xi_i + F^T \delta_i) \tag{25}$$

The equations of $\overline{\Omega}_i$ and $\bar{\xi}_i$ given above are rewritten to have common parts in the equation. These common parts are calculated first to avoid redundancy. This results in the following calculations that have to be executed in the prediction step:

$$\begin{aligned} \Psi_i &= F_x^T [(I + \Delta_i)^{-1} - I] F_x \\ \lambda_i &= \Psi_i^T \Omega_{i-1} + \Omega_{i-1} \Psi_i + \Psi_i^T \Omega_{i-1} \Psi_i \\ \Phi_i &= \Omega_{t-1} + \lambda_i \\ \kappa_i &= \Phi_i F_x^T \left(R_i^{-1} + F_x \Phi_i F_x^T\right)^{-1} F_x \Phi_i \\ \overline{\Omega}_i &= \Phi_i - \kappa_i \\ \bar{\xi}_i &= \xi_{i-1} + (\lambda_i - \kappa_i) \mu_{i-1} + \overline{\Omega}_i F_x^T \delta_{i-1} \\ \bar{\mu}_i &= \mu_{i-1} + F_x^T \delta_i \end{aligned} \tag{26}$$

3.5.2 Measurement update

The measurement update is based on the measurement update of the information filter:

$$\Omega_i = \overline{\Omega}_i \; + H_i^T Q_i^{-1} H_i \tag{27}$$

$$\xi_i = \bar{\xi}_i \; + H_i^T Q_i^{-1} (z_t - h(\mu_i) + H_i \mu_i) \tag{28}$$

where z_t is the actual measurement and $h(\mu_i)$ the noise free components of the approximated measurement function. In most cases, and especially in vSLAM, there are multiple observations per time. Therefore the equations are adapted to the following forms:

$$\Omega_i = \overline{\Omega}_i \; + \sum_j H_i^{k\,T} Q_i^{-1} H_i^k \tag{29}$$

$$\xi_i = \bar{\xi}_i \; + \sum_j H_i^{j\,T} Q_i^{-1} (z_i^k - \tilde{z}_i^k + H_i^j \mu_i) \tag{30}$$

3.5.3 Update state estimation

During the measurement update only Ω and ξ are updated. In this step, the predicted mean $\bar{\mu}_i$ is updated using the updated matrix Ω and vector ξ . The most obvious calculation to do this is using $\mu = \Omega^{-1}\xi$, obtained from equation (3). A disadvantage of this method is that this has a high computational cost.

Therefore an approximation is used to calculate the corrected mean. Furthermore, only the individual variables of interest are calculated: The state vector x_i and the active landmarks of the map. The definition of active landmarks will be explained in Section 3.5.4.

The problem can be solved using different methods. Thrun et al. [4] suggests an iterative hill climbing algorithm, which is an instantiation of the coordinate descent algorithm. In Gaussian distribution form it has the following shape:

$$\begin{aligned}\hat{\mu} &= \underset{\mu}{\operatorname{argmax}}\, p(\mu) \\ &= \underset{\mu}{\operatorname{argmax}} \exp\left(-\frac{1}{2}\mu^T\Omega\mu + \xi^T\mu\right) \\ &= \underset{\mu}{\operatorname{argmin}} \frac{1}{2}\mu^T\Omega\mu - \xi^T\mu \\ &= \underset{\mu}{\operatorname{argmin}} \frac{1}{2}\sum_a\sum_b \mu_a^T\Omega_{a,b}\mu_b - \sum_a \xi_a^T\mu_a\end{aligned} \tag{31}$$

This results in a form that makes the individual coordinate variable μ_a explicit, with 'a' the rows of Ω, ξ and μ, and 'b' the columns of Ω.

To find an optimal solution of μ_a the derivative of the last equation of (31) with respect to μ_a is taken:

$$\frac{\delta}{\delta\mu_a}\left\{\frac{1}{2}\sum_a\sum_b \mu_a^T\Omega_{a,b}\mu_b - \sum_a \xi_a^T\mu_a\right\} = \sum_j \Omega_{a,b}\mu_b - \xi_i \tag{32}$$

By setting this equation to zero an optimum of μ_a is found:

$$\begin{aligned}0 &= \sum_j \Omega_{a,b}\mu_b - \xi_i \\ -\Omega_{a,a}\mu_a &= \sum_{a\neq b} \Omega_{a,b}\mu_b - \xi_i \\ \mu_a &= \Omega_{a,a}^{-1} - \sum_{a\neq b} \Omega_{a,b}\mu_b + \xi_i\end{aligned} \tag{33}$$

To reduce the computation time, only the dimensions of the μ that are useful to know. For every state a in μ a projection matrix F is used:

$$\mu_a = \left(F_a\Omega F_a^T\right)^{-1}F_a\left(\xi - \Omega\mu + \Omega F_a^T F_a\mu\right) \tag{34}$$

where the projection matrix F has the following shape for calculating the state ($a = x_i$)

$$F_x = \left[\underbrace{I}_{\text{size of } x_i} \quad \underbrace{0}_{\text{size of } m_{1:J}}\right] \} \text{ size of } x_i \tag{35}$$

And for a landmark ($a=m_j$):

$$F_j = \left[\underbrace{0}_{\text{size of } x_i} \quad \underbrace{0}_{\text{size of } m_{1:j-1}} \quad \underbrace{I}_{\text{size of } m_j} \quad \underbrace{0}_{\text{size of } m_{j+1:J}}\right] \} \text{ size of } m_j \tag{36}$$

Not al the landmarks have to be updated. Only the active landmarks. This is explained in the next section.

Note that in the first iterations, when running the algorithm, the state has to be calculated with $\mu = \Omega^{-1}\xi$. Otherwise the above mentioned approximation will not hold.

3.5.4 Sparsification

The sparsification step reduces the number of non-zero off-diagonal elements of Ω to create sparseness. The operations in previous steps are computational more efficient when the matrix Ω is sparse. The

principal of the sparsification step is removing links between the pose and landmarks and use this information to make other links between landmarks stronger.

In Figure 6 a graphical interpretation of the relation between links and Ω can be seen. The red link corresponds to the red colored cells in Ω, shown on the right side of the figure. The grey links correspond to the grey colored cells. The associated landmarks are called active landmarks. As can be seen, there is no link between x_i and m_{j+1} . The link corresponds to the white cells in Ω and are zero. The landmark associated to this link, m_{j+1}, is called a passive landmark. When landmark m_j is chosen to be sparsefied, the red link will be removed and the corresponding red cells in Ω also become zero. This landmark m_j is become passive.

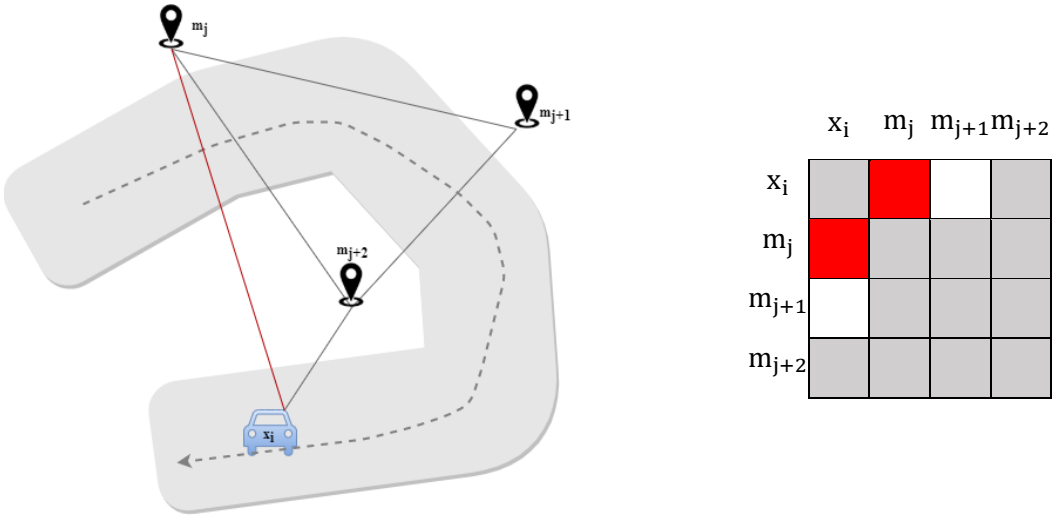


Figure 6 Graphical interpretation of sparsification

In the sparsification step has to be decided which landmarks remain active and which have to be made passive. There are different ways to decide which active features have to become passive. A simple and effective way is to make a queue and select the first, most important number of landmarks in the queue to be active. The active landmarks before the sparsification step consist of active landmarks from the previous step plus the observed landmarks that have not been active previously. The most simple procedure is to make a distinction between active landmarks, that where active in the previous time stamp and have not been observed, and the observed landmarks. The previous active landmarks that have not been observed are placed at the back of the queue.

The sparsification step is performed by assuming conditional independence of specified landmarks directly linked to the pose (red line in Figure 2). In a general formulation, the probability $p(a, b, c)$ can be approximated by a probability $\tilde{p}$ so that a and b are independent given c:

$$\begin{aligned} \tilde{p}(a|b, c) &= p(a|c) \\ \tilde{p}(b|a, c) &= p(b|c) \end{aligned} \tag{37}$$

This is done in the following way:

$$p(a, b, c) = p(a|b, c)\, p(b|c)\, p(c) \approx p(a|c)\, p(b|c)\, p(c) \tag{38}$$

where first standard definition of conditional independence with three variables is stated and then conditional independence is assumed by removing the conditional variable b in the first probability.

To apply this in the information form, the following calculations have to be performed to calculate the sparsefied information matrix $\widetilde{\Omega}_i$:

$$\Omega_i^0 = F_{x_i, m^+ m^0}\, F_{x_i, m^+ m^0}^T\, \Omega\, F_{x_i, m^+ m^0}^T F_{x_i, m^+ m^0} \tag{39}$$

$$\Omega_i^1 = \Omega_i^0 - \Omega_i^0 F_{m^0} \left(F_{m^0}^T \Omega_i^0 F_{m^0}\right)^{-1} F_{m^0}^T \Omega_i^0 \tag{40}$$

$$\Omega_i^2 = \Omega_i^0 - \Omega_i^0 F_{x_i,m^0} \left(F_{x_i,m^0} \Omega_i^0 F_{x_i,m^0}\right)^{-1} F_{x_i,m^0} \Omega_i^0 \tag{41}$$

$$\Omega_i^3 = \Omega_i - \Omega_i F_x \left(F_x^T \Omega_i F_x\right)^{-1} F_x^T \Omega_i^0 \tag{42}$$

$$\widetilde{\Omega}_i = \Omega_i^1 - \Omega_i^2 + \Omega_i^3 \tag{43}$$

with m^+ the set of active landmarks that remain active and m^0 the set of active landmarks to become passive. The projection matrices $F_{[.]}$ have the same shape as in (19), with at the dot the indices in the information matrix corresponding to the variables.

A derivation of above described calculations from a probabilistic point of view can be found in Appendix A.

The sparsefied information vector ξ_i can easily be calculated in de following way:

$$\begin{aligned} \tilde{\xi}_i &= \widetilde{\Omega}_i \mu_i \\ &= (\Omega_i - \Omega_i + \widetilde{\Omega}_i \mu_i \\ &= \Omega_i \mu_i + \left(\widetilde{\Omega}_i - \Omega_i\right)\mu_i \\ &= \xi + \left(\widetilde{\Omega}_i - \Omega_i\right)\mu_i \end{aligned} \tag{44}$$

3.5.5 Complexity

Computational complexity of the prediction of SEIF is constant $O(1)$ [4] [6]. The matrices Ψ_i, λ_i, Φ_i are calculations with sparse matrices and include only non-zero elements corresponding to the state vector x_i. However, the matrix Φ_i in the calculation of κ_i is more dense, but, due to the sparsification step, Φ_i is sparse. The multiplication of $F_x^T\left(R_i^{-1} + F_x \Phi_i F_x^T\right)^{-1} F_x$ with this matrix Φ_i (left and right) touches only rows and columns that correspond to active features in the map. Since the size of Φ_i does not depend on the number of active features, this step also requires $O(1)$. The same reason holds for calculating $\bar{\xi}_i$, the sparsification and the update state estimation. Since, in contrast to EKF, the measurement update is an addition, it also requires $O(1)$.

3.6 Graph SLAM

In this chapter, the Graph SLAM algorithm is explained. The working principle is derived from [2]. Since it is a full algorithm, all states are calculated at once. In Figure 7 a flow diagram of Graph SLAM is shown. In the initialization, the mean of the states $\bar{\mu}_x$ are calculated using relative measurements $u_{1:I}$ only. In the next step, the information matrix Ω and information vector ξ are calculated using this trajectory and absolute measurements $z_{1:I}^{1:K}$, including their correspondences $c_{1:I}^{1:K}$. Next, the mean μ can be obtained from the information matrix Ω and information vector ξ. However, to gain a more accurate estimation, the information matrix Ω and information vector ξ can be calculated again. Instead of using the mean of the states $\bar{\mu}_x$ calculated in the initialization, the mean of the states obtained from the information matrix Ω and information vector ξ μ_x is used. This results in a more accurate linearization, which results in an even more accurate estimation. This process can be repeated until the outcome converges.

To avoid unnecessary calculations by calculating the mean of the states μ_x, the reducing part extracts the information related to the states from Ω and ξ. This results in Ω_{red} and ξ_{red}. These matrices are used in the solving part to calculate the mean of state vector μ_x. Also the covariance matrix and the map μ_m could be calculated in an efficient way. The latter is usually done in the last iteration.

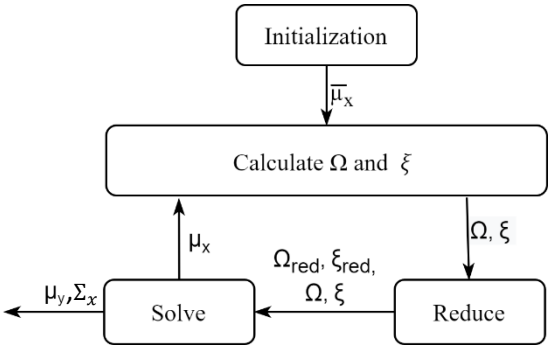


Figure 7 Flow Graph SLAM

In the next section, the theory behind Graph SLAM is explained. There, every step of the flow chart in Figure 7 will be covered.

3.6.1 Theoretical background Graph SLAM

Bayes rule plays an important role for calculating the posterior $p(y_{0:I}|(z_{1:I}^{1:K}, c_{1:I}^{1:K}, u_{1:I})$ in Graph SLAM. Bayes rule is defined as:

$$p(a|b) = \frac{p(b|a)p(a)}{p(b)} \tag{45}$$

The posterior $p(y_{0:I}|(z_{1:I}^{1:K}, c_{1:I}^{1:K}, u_{1:I})$ is equal to $p(y_{0:I}|(z_i^{1:K}, z_{1:I-1}^{1:K}, c_{1:I}^{1:K}, u_{1:I})$. If the variable 'a' in (45) represents state vector $y_{0:I}$, 'b' is replaced by measurements $z_i^{1:K}$ the following equation can be defined using Bayes rule:

$$p(y_{0:I}|z_i^{1:K}) = \frac{p(z_i^{1:K}|y_{0:I})\, p(y_{0:I})}{p(z_i^{1:K})} \tag{46}$$

The conditioning variables $z_{1:I-1}^{1:K}$, $c_{1:k}$and $u_{1:k}$ are added to each of the probability functions due to conditional independence:

$$p(y_{0:I}|z_{1:I}^{1:K}, c_{1:I}^{1:K}, u_{1:I}) = \frac{p(z_i^{1:K}|y_{0:I}, z_{1:I-1}^{1:K}, c_{1:I}^{1:K}, u_{1:I})\, p(y_{0:I}|z_{1:I-1}^{1:K}, c_{1:I}^{1:K}, u_{1:I})}{p(z_i^{1:K}|z_{1:I-1}^{1:K}, c_{1:I}^{1:K}, u_{1:I})} \tag{47}$$

The first probability in the numerator of (47) can be reduced, because the probability of $z_i^{1:K}$ can be estimated using the combined state vector y_i:

$$p(z_i^{1:K}|y_{0:I}, z_{1:I-1}^{1:K}, c_{1:I}^{1:K}, u_{1:I}) = p(z_i^{1:K}|y_i, c_i^{1:K}) \tag{48}$$

The second probability in the numerator of (47) can be partitioned in the probability of the combined state vector in the past ($y_{0:I-1}$) times the probability of the current pose using the previous state x_{i-1} and controls u_i:

$$p(y_{0:I}|z_{1:I-1}^{1:K}, c_{1:I}^{1:K}, u_{1:I}) = p(x_i|x_{i-1}, u_i)\, p(y_{0:I-1}|z_{1:I-1}^{1:K}, c_{1:I-1}^{1:K}, u_{1:I-1}) \tag{49}$$

The dominator of (47), $p(z_{1:i}|z_{1:I-1}^{1:K}, c_{1:i}, u_{1:i})$ is static. Therefore this part can be seen as normalizer η. Substituting equations (48) and (49) into (47) and replacing $p(z_{1:i}|, c_{1:i}, u_{1:i})$ by η gives the recursive definition of the full SLAM posterior:

$$p(y_{0:I}|(z_{1:I}^{1:K}, c_{1:I}^{1:K}, u_{1:I}) = \eta\, p(z_i^{1:K}|y_i, c_i^{1:K})\, p(x_i|x_{i-1}, u_i)\, p(y_{0:I-1}|z_{1:I-1}^{1:K}, c_{1:I-1}^{1:K}, u_{1:I-1}) \tag{50}$$

To make it more computable it can be defined in an iterative way:

$$p(y_{0:I}|(z_{1:I}^{1:K}, c_{1:I}^{1:K}, u_{1:I}) = \eta\, p(y_0) \prod_i \left[p(x_i|x_{i-1}, u_i) \prod_k p(z_i^k|y_i, c_i^k) \right] \tag{51}$$

As described in Section 3.1.2, a big advantage of the log of the posterior is that Bayes rule becomes an addition:

$$-\log p(y_{0:I}|(z_{1:I}^{1:K}, c_{1:I}^{1:K}, u_{1:I}) = c + \log p(y_0) + \sum_i \left[\log p(x_i|x_{i-1}, u_i) + \sum_k \log p(z_i^k|y_i, c_i^k)\right] \quad (52)$$

The probabilistic form (52) of the posterior can be described in information form. Since there is a map during the initialization, the log of $p(y_0)$ can be defined to be:

$$\log p(y_0) = \log p(x_0) = c - \frac{1}{2} x_0^T \Omega_0 x_0 \quad (53)$$

The probabilities $p(x_i|x_{i-1}, u_i)$ and $p\left(z_i^j \middle| y_i, c_i\right)$ can be defined in Gaussian form (1) in moments parameterization:

$$p(x_i|x_{i-1}, u_i) = \frac{1}{\sqrt{(2\pi)^D |R_i|}} exp\left(-\frac{1}{2}(x_i - \mu_{x_i})^T R_i^{-1}(x_i - \mu_{x_i})\right) \quad (54)$$

$$p(z_i^k|y_i, u_i, c_i^j) = \frac{1}{\sqrt{(2\pi)^D |Q_i^k|}} exp\left(-\frac{1}{2}\left(z_i^k - \mu_{z_i^j}\right)^T Q_i^{k^{-1}}\left(z_i^k - \mu_{z_i^k}\right)\right) \quad (55)$$

where R_i and Q_i^k are the covariance matrices relative to process and measurement noise, respectively. The estimations μ_{x_i} and $\mu_{z_i^k}$ can be obtained from the linearized Taylor functions (8) and (9). By substituting these equations, and rewriting the equation, the following expressions for both probabilities can be defined:

$$p(x_i|x_{i-1}, u_i) = \eta\, exp\left(-\frac{1}{2}\left([x_i\ x_{i-1}]\begin{bmatrix}-G_t^T \\ 1\end{bmatrix} + R_i^{-1}[-G_i^T\ 1]\begin{bmatrix}x_i \\ x_{i-1}\end{bmatrix} + [x_i\ x_{i-1}]\begin{bmatrix}-G_i^T \\ 1\end{bmatrix} R_i^{-1}\left(\tilde{x}_i - G_i\,\mu_{x_{i-1}}\right)\right)\right) \quad (56)$$

$$p(z_i^k|y_i, u_i, c_i^i) = \eta\, exp\left(-\frac{1}{2}\left({y_i}^T {H_i^j}^T Q_i^{-1} H_i^j + {y_i}^T {H_i^k}^T Q_i^{-1}(z_i^k - \tilde{z}_i^k + H_i^k(y_i - \mu_{y_i})\right)\right) \quad (57)$$

with the terms that remain static subsumed to normalizer η.

The result of the substitution is given in equation (58).

$$-\log p(y_{0:I}|(z_{1:I}^{1:K}, c_{1:I}^{1:K}, u_{1:I}) = c - \frac{1}{2} x_0^T \Omega x_0 - \frac{1}{2}\sum_i \left([x_i\ x_{i-1}]\begin{bmatrix}-G_i^T \\ 1\end{bmatrix} R_i^{-1}[-G_i\ 1]\begin{bmatrix}x_i \\ x_{i-1}\end{bmatrix} + [x_i\ x_{i-1}]\begin{bmatrix}-G_i^T \\ 1\end{bmatrix} R_i^{-1}\left[\tilde{x}_i - G_i\,\mu_{x_{i-1}}\right]\right) \frac{1}{2}\sum_j y_i^T {H_i^k}^T Q_i^{-1} H_i^k\, y_i + y_i^T {H_i^k}^T Q_i^{-1}\left(z_i^k - \tilde{z}_i^k + H_i^k\,\mu_{y_i}\right) \quad (58)$$

The equation highlights an essential characteristic of the full SLAM posterior in the information form. It is composed of a number of quadratic terms. The equation can be written in matrix form according (5) by collecting the parameters of the quadratic terms to matrix Ω and the parameters of the linear terms into ξ. This is explained in de next sections.

3.6.2 Initialization

In the initialization, μ_x is calculated for the whole path. A position for μ_0 is defined. Then the motion function is repeated I-1 times:

$$\mu_i = g(\mu_{i-1}, u_i) \quad (59)$$

3.6.3 Calculating the information matrix and information vector

As previously stated, equation (58) can be written in matrix form according (5) by collecting the parameters of the quadratic terms to matrix Ω and the parameters of the linear terms into ξ. The final Graph SLAM posterior in (58) is separated into five parts, where the parameters of the three quadratic terms can be collected to matrix Ω and the two parameters of the linear terms can be collected to ξ. The information matrix Ω is a square matrix where the size of the state vector 'y' times the number of variables a pose is described. In Figure 8 and Figure 9, the layouts of the matrix Ω and vector ξ are given. In this example a 2 dimensional Cartesian system (x,y) is chosen for the span in which the vehicle moves.

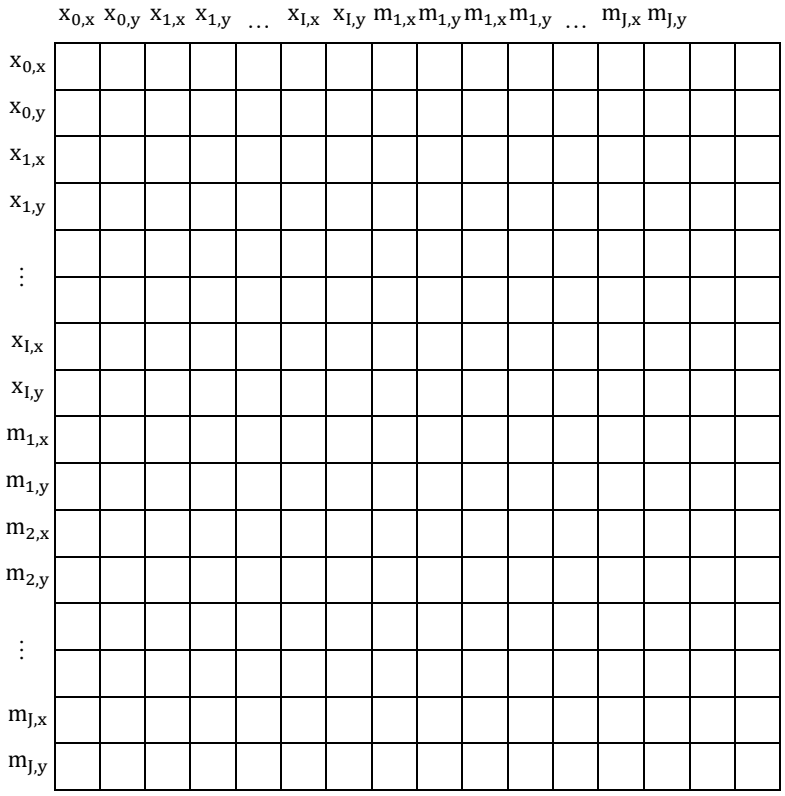


Figure 8 Layout of Ω

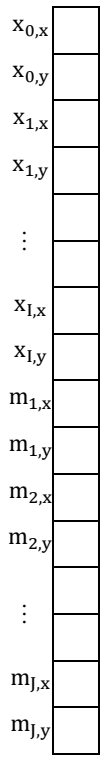


Figure 9 Layout of ξ

The parameters in the SLAM posterior (58) related to Ω and ξ are separated and given in the equations (60) to (64). The curly embraced equations (60) and (61) are calculated for each motion event between x_{i+1} and x_i. The results of (60) and (61) are a square matrix and a vector, respectively, with the size of $2x_i$. These results are added to the cells corresponding to x_{i+1} and x_i in the matrix Ω and vector ξ in Figure 8 and Figure 9. Adding the values to Ω and ξ can be done in an efficient way by matrix addition. This is explained in Appendix B.

$$\begin{bmatrix} -G_i^T \\ 1 \end{bmatrix} R_i^{-1} [-G_i \; 1] \tag{60}$$

$$\begin{bmatrix} -G_i^T \\ 1 \end{bmatrix} R_i^{-1} \left[\tilde{x}_i - G_i \, \mu_{x_{i-1}} \right] \tag{61}$$

A measurement event could be added by calculating equations (62) and (63). The resulting matrix and vector are added to the cells corresponding to x_i and m_j in the matrix Ω and vector ξ in Figure 8 and Figure 9.

$$H_i^{j\,T} Q_i^{-1} H_i^j \tag{62}$$

$$H_i^{j^T} Q_i^{-1} \left(z_i^i - \hat{z} + H_i^j y_i \right) \tag{63}$$

The initialization is given in (64).

$$x_0^T \underbrace{\Omega} x_0 \tag{64}$$

Usually the algorithm is initialized with a variance of 0, so Ω_0 is defined as follows:

$$\Omega_0 = \Sigma_0^{-1} = \begin{bmatrix} \infty & 0 \\ 0 & \infty \end{bmatrix} \tag{65}$$

3.6.4 Reducing

To calculate only the state x, the matrix and vector can be reduced by removing information of the map. This is done using the marginalization lemma (Appendix C). This leads to the following equations:

$$\Omega_{red} = \Omega_{x_{0:I},x_{0:I}} - \Omega_{x_{0:I},m}\, \Omega_{m,m}^{-1}\, \Omega_{m,x_{0:I}} \tag{66}$$

$$\xi_{red} = \xi_{x_{0:I}} - \Omega_{x_{0:I},m}\, \Omega_{m,m}^{-1}\, \xi_m \tag{67}$$

In an iterative way:

$$\Omega_{red} = \Omega_{x_{0:i},x_{0:i}} - \sum_j \Omega_{x_{0:i},j}\, \Omega_{j,j}^{-1}\, \Omega_{j,x_{0:i}} \tag{68}$$

$$\xi_{red} = \xi_{x_{0:i}} - \sum_j \Omega_{x_{0:i},j}\, \Omega_{j,j}^{-1}\, \xi_j \tag{69}$$

3.6.5 Solving

After the reducing part the state vector can be obtained in moments parameterization by rewriting (2) and (3):

$$\Sigma_x = {\Omega_{red}}^{-1} \tag{70}$$

$$\mu_x = \Sigma_x\, \xi_{red} \tag{71}$$

Obviously, the positions of the landmarks cannot be calculated using Ω_{red} and ξ_{red}. The conditioning lemma is applied to the non-reduced matrices Ω and ξ to obtain a mean and variance of the map.

$$\Sigma_m = \Omega_{m,m}^{-1} \tag{72}$$

$$\mu_m = \Sigma_m(\xi_m - \Omega_{m,x_{0:t}} \xi_{reduced}) \tag{73}$$

3.6.6 Complexity

The computational complexity of Graph SLAM is highly depended on the inversion of the information matrix. The sparseness of this matrix is highly depended on the number of landmarks, measurements and the number of states. Therefore it is difficult to express it in big O notation. There are several researchers that developed a Graph SLAM algorithm with reduced complexity [7], [8], but the complexity is not expressed in big O. The information matrix and vector are increasing to the number of landmarks and the number of states. Therefore, the memory usage grows linearly ($O(n)$) in the number of landmarks and the number of states.

3.7 FAST-SLAM

The FAST-SLAM algorithm is a combination of two estimators. The pose is estimated using the particle filter. The pose is therefore represented by a number of particles. Each particle is an instance of an estimated pose. The map is estimated using the EKF filter. Each particle has its own estimates of landmarks with a mean μ_{m_j} and a covariance matrix Σ_{m_j}. The landmarks are conditionally independent from each other, so there is only a link between the pose and each landmark. The landmarks are conditionally independent from each other, so there is only a link between the pose and each landmark.

In Figure 10 an overview of the working principle is shown. As FAST-SLAM is also based on the Bayes filter, it consists of a prediction step and a measurement update. The prediction, the measurement update and the importance weight are performed on a particle basis. The resampling step is done for the whole set of particles.

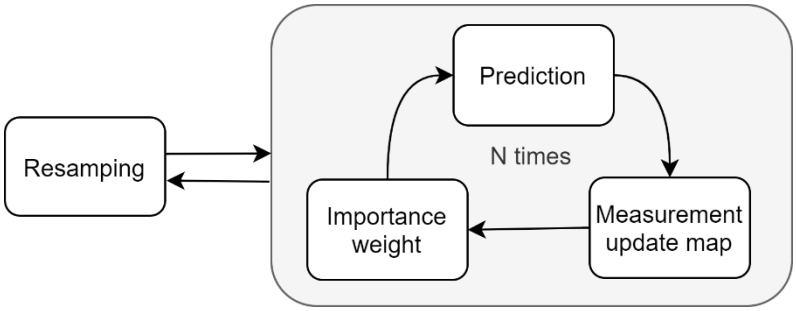


Figure 10 Operation flow FAST-SLAM

There are two versions of FAST-SLAM, namely FAST-SLAM 1.0 and FAST-SLAM 2.0. The difference between the two algorithms is the way how the prediction step is computed. In the next sections the algorithms will be explained in detail. An even more elaborated explanation can be found in [2]. First, the operation steps for FastSLAM 1.0 are explained. Afterwards the differences with FastSLAM 2.0 are described.

3.7.1 Prediction

Given the particle set representing the previous state, a prediction will be performed to compute the next state. This set of particles is also known as the proposal distribution.

In FAST-SLAM 1.0 the next pose is computed by sampling from the so-called proposal distribution. In case of FAST-SLAM 1.0 this distribution is the following transition probability:

$$x_i^{[n]} \sim p\left(x_i^{[n]} \mid x_{i-1}^{[n]}, u_i\right) \tag{74}$$

with n is the n-th particle. It is assumed that this transition is Gaussian with mean $g\left(u_i, x_{i-1}^{[n]}\right)$ and a covariance matrix that describes the uncertainty of the prediction.

3.7.2 Measurement update map

If a landmark has never been seen before, it will be initialized with a mean and a covariance matrix. The mean $m_{j+1}^{[n]}$ is calculated by taking the inverse of the measurement function.

$$m_{j+1}^{[n]} = h^{-1}\left(z_i^k, \bar{x}_i^{[n]}\right) \tag{75}$$

where z_i^k is the measurement vector.

Before the associated covariance matrix can be calculated, the Jacobian matrix of the measurement function with respect to δx and δm, evaluated at pose $x_i^{[n]}$ and landmark position $m_{j+1}^{[n]}$ has to be calculated first:

$$H_{j+1,i}^{[n]} = \left.\frac{\delta h(y_i, k)}{\delta x, \delta m}\right|_{x_i^{[n]}, m_{j+1}^{[n]}} \tag{76}$$

The covariance matrix becomes:

$$\Sigma_{k+1,i}^{[n]} = {H_{k+1,i}^{[n]}}^{-1} Q_i \left({H_{k+1,i}^{[n]}}^{-1}\right)^T \tag{77}$$

with the measurement noise Q_i.

When a landmark is already observed, the mean and covariance is updated using the EKF filter:

$$\begin{aligned} K &= \Sigma_{k,i}^{[n]} H_j^T Q_i^{-1} \\ \mu_{k,i}^{[n]} &= \mu_{k,i-1}^{[n]} + K\left(z_i^k - \tilde{z}_i^{k[n]}\right) \\ \Sigma_{k,i}^{[n]} &= (I - KH_k)\Sigma_{k,i-1}^{[n]} \end{aligned} \tag{78}$$

3.7.3 Assign importance weight

In this step, for each particle a so-called 'importance weight' is calculated. This value gives a rate of how certain the state of a certain particle is according to measurements. The importance weight is used in the resampling step explained in the next section. Per particle, for each measurement of the set $z_i^{1:k}$ the importance weight $w_i^{k[n]}$ is calculated. Importance weights are actually the likelihoods of the particles:

$$w_i{}^{[n]} = \prod_{k=1}^{K} w_i^{k[n]} \tag{79}$$

with

$$w_i^{k[n]} = p\left(z_i^k, x_i^{[n]}\right)$$

The importance factor is initiated as follows:

$$w_i^{k=0[n]} = 1 \tag{80}$$

Then the importance weight is updated in the following way for each measurement:

$$w_i^{k[n]} = w_i^{k-1[n]} \left|2\pi Q_i^{k[n]}\right|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\left(z_i^k - \tilde{z}_i^{k[n]}\right)\left(Q_i^{k[n]}\right)^{-1}\left(z_i^k - \tilde{z}_i^{k[n]}\right)\right\} \tag{81}$$

With $Q_i^{k[n]}$ the measurement covariance, z_i^k the measurement, $\tilde{z}_i^{k[n]}$ the predicted measurement using the measurement function of the k'^{th} measurement at time instant i

3.7.4 Resampling

The importance weight will be used for resampling, an important aspect of the particle filter. Particle sets with a small importance weight have a large chance to be removed from the set, while particles with a large importance weight have a large change to survive, and have even a large chance to be cloned. In this way there are more particles in areas where there is a high likelihood. The probability distribution after resampling is called a target distribution.

A popular variant of resampling is low variance sampling. It is a simple and computationally effective algorithm. An illustration of low variance resampling is given in Figure 11. The size of the bars resembles the importance weight of each particle. The sum of the importance weights is normalized to 1. First, a random number r is generated between 0 and N^{-1}, which is represented by the first arrow in Figure 11. Afterwards, N^{-1} is added in an iterative process, which is represented by the other arrows in Figure 11. The particle where each arrow points to will be selected. Note that during a resampling process, the number of particles before and after resampling always remains the same.

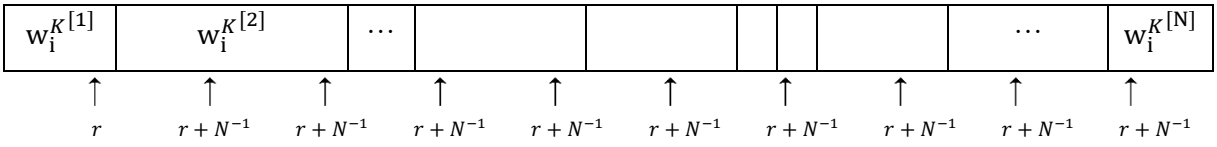


Figure 11 Illustration of low variance resampling

In Figure 12 an example of resampling is shown. In this example, a 2D (x,y) position is estimated using 1000 particles. The red particles show the proposal distribution, the green particles represent the target distribution. It can be seen that particles near the ground truth position, given in blue, survived. This example shows that the target distribution does indeed represent a better likelihood of the real position.

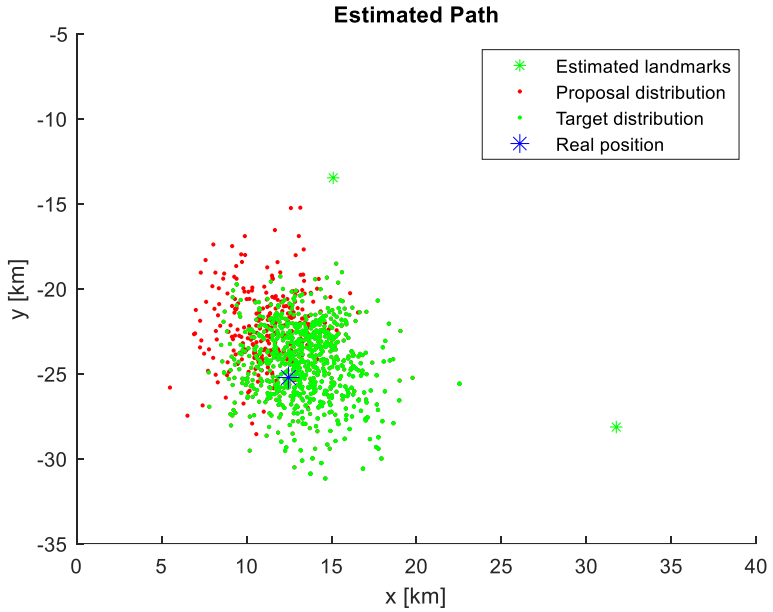


Figure 12 Resampling example

3.7.5 Difference FAST-SLAM 1.0 and 2.0

The algorithm is improved by using a proposal probability density that is more selective than the transition probability density. This is achieved by taking the landmark measurements into account:

$$x_i^{[n]} \sim p\left(x_i \mid x_{1:i-1}^{[n]}, u_{1:i}\, z_{1:i}^{1:K}\, c_{1:i}^{1:K}\right) \tag{82}$$

This is implemented by sampling according to a normal distribution by using a mean $\bar{x}_{i,K}$ and the associated covariance matrix $\Sigma_{x,1:K}$:

$$x_i^{[n]} \sim \mathcal{N}\left(\bar{x}_{i,K}, \bar{\Sigma}_{x,K}\right) \tag{83}$$

The vector $\bar{x}_{i,1:K}$ is calculated by predicting the next state with the motion function and adapting the state using the measurements. First the motion function is used to calculate a prediction:

$$\tilde{x}_i^{[n]} = g\left(x_{i-1}^{[n]}, u_i\right) \tag{84}$$

This state is used to predict the measurements:

$$\bar{z}_i^k = h\left(\tilde{x}_i^{[n]}, m_{j=c_i^k}^{[n]}\right) \tag{85}$$

With k=1, …, K. Note that this measurement is indicated with a bar. Since this is a different measurement prediction then $\tilde{z}_i^k$.

For each measurement two Jacobians of the measurement function have to be calculated. The first Jacobian (90) is calculated with respect to the state $\tilde{x}_i^{[n]}$ (86) and the second (91) with respect to the corresponding landmark $m_{j=c_i^k}^{[n]}$ (87) of the measurement.

$$Hx_{i,k}^{[n]} = \left.\frac{\delta h(\mu_i, j)}{\delta x}\right|_{\tilde{x}_i^{[n]}, m_{c_i^k}^{[n]}} \tag{86}$$

$$Hm_{i,k}^{[n]} = \left.\frac{\delta h(\mu_i, j)}{\delta m}\right|_{\tilde{x}_i^{[n]}, m_{c_i^k}^{[n]}} \tag{87}$$

$\bar{x}_{i,K}^{[n]}$ and $\bar{\Sigma}_{x,K}^{[n]}$can be calculated using each of the measurements:

$$Q_k^{[n]} = Hm_{i,k}^{[n]}\, \Sigma_{k,i-1}^{[n]}\, Hm_{i,k}^{[n]^T} + Q_t \tag{88}$$

$$\bar{\Sigma}_{x,k}^{[n]} = \left[Hm_k^{[n]}\, Q_k^{[n]^{-1}}\, Hm_k^{[n]} + \Sigma_{x,k-1}\right]^{-1} \tag{89}$$

$$\bar{x}_{i,k}^{[n]} = \bar{x}_{i,k-1} + \Sigma_{x,k} H_{x,j}^T Q_k^{[n]^{-1}}\left(z_i^k - \bar{z}_i^k\right) \tag{90}$$

with $\bar{x}_{i,k}^{[n]}$ and $\bar{\Sigma}_{x,k}^{[n]}$ initiated in the following way:

$$\bar{x}_{i,k=0}^{[n]} = \tilde{x}_i^{[n]} \tag{91}$$

$$\bar{\Sigma}_{x,k=0}^{[n]} = R_i^{-1} \tag{92}$$

with R_i the covariance matrix related to measurement noise.

The importance weight w_K is initiated with a value of '1' and then updated using each measurement in the following way:

$$w_k = w_{k-1} * |2\pi Q_k|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\left(z_i^k - \tilde{z}_i^k\right)^T Q_k^{-1}\left(z_i^k - \tilde{z}_i^k\right)\right\} \tag{93}$$

with $\tilde{z}_i^k = h\left(x_i^{[n]}, m_{j=c_i^k}^{[n]}\right)$

Note that the initiation of new landmarks also can be initiated with a much more accurate prediction of the state vector $x_i^{[n]}$. Therefore the initialization of new landmarks in FAST-SLAM 2.0 should always be performed after the calculation of the proposal distribution.

3.7.1 Complexity

The computational complexity of the FAST-SLAM algorithm, implemented as described above, is $O(NJ)$ with N the number of particles and J the number of landmarks. However the algorithm can be made more efficient. The linear complexity in N is unavoidable, given that we have to process N particles with each update. The linear complexity in J is the result of the resampling process. Whenever

a particle is drawn more than once, the associated set of landmarks can be copied. This results in a linear complexity in J. The complexity could be reduced to $\log(J)$ by introducing a data structure for representing particles that allow for more selective updates. The basic idea is to organize the map as a balanced binary tree. An example is given in Figure 13. It shows the mean and covariance matrices of the associated landmarks of particle $x_i^{[1]}$ on the leaves of the binary tree. In the resampling process this particle has a likelihood and will be duplicated to $x_i^{[2]}$. The mean $\mu_{1:8}$ and the covariance matrices $\Sigma_{1:8}$ of the landmarks do not differ, so instead of copying the whole set, the mean and covariance matrices of the new particle $x_i^{[2]}$ are pointing to the mean and covariance of the old particle $x_i^{[1]}$. In the next iteration states $x_{i+1}^{[1]}$ and $x_{i+1}^{[2]}$ are predicted and an absolute measurement z_i^k, corresponding to landmark j=3, is acquired. At this moment the mean μ_3 and covariance matrix Σ_3 change between the particles and therefore mean $\mu_3^{[1]}$ and covariance matrix $\Sigma_3^{[1]}$ the will be copied before the measurement will be incorporated. Efficient implementations like this binary tree require only $O(N \log(J))$.

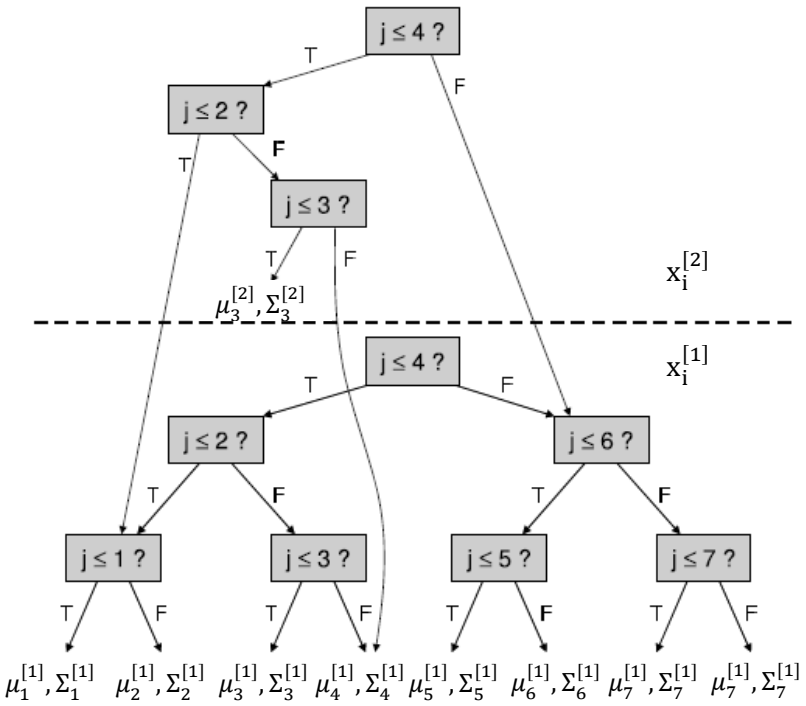


Figure 13 Binary tree [4]

Memory usage is also linear $O(N\,J)$ for ineffective implementations. However, the complexity can be reduced to more or less equal to the saving in computation time.

4. Methods

This chapter presents a series of experiment methods that will be executed. In the first section the goal of the experiments is described. The experimental set-ups are provided in the next section. The evaluation section describes what will be measured during the experiments.

4.1 Goals

The goal of the experiments is to answer research question 3:

- What is the performance of these algorithms within the vSLAM feature-based framework?

Also the hypothesis can be verified using the experiments:

- A high accuracy of the estimated 3D positions will be at the cost of computational complexity of an algorithm.

In Chapter 3 the theory of different SLAM algorithms are provided. The results can be compared with the theoretical data.

4.2 Experimental set-up

To achieve this goal, three data sets are presented. The first data set consists of 2D data. The second and the third data set consist of points and features from images. The second data set is simulated and the third data set is extracted form real images. The data sets are described in detail in the following sections.

4.2.1 2D map

This 2D environment includes data with relative information from a compass and absolute measurements from beacons. For this experiment, a data set of control and measurement variables is obtained from a ship that is sailing around the British Isles. Ground truth data and motion, measurement equations and measurements are obtained from [9].

State vector

$$\mathrm{x}_i = \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \end{bmatrix} \tag{94}$$

Motion function

The ship is provided with two different types of sensors that provide relative information: a log which measures the speed $\mathrm{v(i)}$ of the ship and a compass that measures the heading $\Phi(\mathrm{i})$ of the ship.

$$\mathrm{u(i)} = \begin{bmatrix} \mathrm{v(i)} \\ \Phi(\mathrm{i}) \end{bmatrix} \tag{95}$$

with standard deviations related to noise of $\sigma_{\mathrm{v}} = 5\ \mathrm{km/s}$ and $\sigma_{\Phi} = 15°$.

The noise free component $\tilde{\mathrm{x}}$ of the linear approximation is calculated using the motion model given in (96), using relative measurements (95).

$$\begin{bmatrix} \tilde{x}_{x_{i+1}} \\ \tilde{x}_{y_{i+1}} \end{bmatrix} = \begin{bmatrix} \mu_{x_i} \\ \mu_{y_i} \end{bmatrix} + \Delta \ v(i) \begin{bmatrix} cos\ \phi(i) \\ sin\ \phi\ (i) \end{bmatrix} \tag{96}$$

The Jacobian G_i is calculated and given in (97).

$$G_i = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{97}$$

Measurement function

A beacon based measurement provides absolute measurements $z_i^{1:K}$. The noise free component $\hat{z}_t^k$ is calculated using the measurement function:

$$\hat{z}_t^k = m_{j=c_i^k} - x_i = \begin{bmatrix} m_{j_x} \\ m_{j_y} \end{bmatrix} - \begin{bmatrix} x_{i_x} \\ x_{i_y} \end{bmatrix} \tag{98}$$

Jacobian of the measurement function

The Jacobian H_i^k of the measurement function, is calculated and given below.

$$H_i^k = \begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \tag{99}$$

4.2.2 Visual SLAM

To analyze the algorithms in a vSLAM framework, two data sets are used. The first data set are points and KAZE features extracted from stereo images. Furthermore, it consists of 3D measurements between the position of the camera and points in the world coordinate system. The set of stereo images are pictures of a decorative globe. The first set that was captured is shown in Figure 14 and Figure 15.

Figure 14 Right image of stereo camera

Figure 15 Left image of stereo camera

The second data set consists of simulated points and KAZE features of a decorative globe that is used. The simulator mimics a camera and a globe. Based on these points, the simulator provides measurements including noise containing descriptors based on KAZE features. An advantage of the simulated data set is that the real data (without noise) is available.

In this experiment a feature based vSLAM algorithm provided by Shah [5] and van der Heijden [10] is used. This vSLAM algorithm handles the matching of feature points between stereo images, the bookkeeping of landmarks, and the visualization of the combined state vector. The bookkeeping of the vSLAM algorithm can be adjusted. The two most important settings are 'P.Mmax' and 'P.max_lm_inc_per_step'. The first setting is the maximum number of landmarks in the combined state vector. The latter is the maximum number of landmarks added at a time.

Originally the SLAM back-end used in this vSLAM algorithm is Error State EKF SLAM. The Error State extended Kalman filter (ES-EKF) is an extension of the classical Extended Kalman filters. It holds the advantage that the actual Kalman filter is only applied to signals with small magnitudes. This simplifies the estimation especially when working with quaternions to represent the orientation. For this project the algorithm will be rebuild to a vSLAM version based on a standard EKF with filter Euler angles. The vSLAM algorithms with other SLAM back-ends addressed in this project will also use Euler angles. In this way the SLAM-back ends are more comparable. More information about the models and the ES-EKF SLAM algorithm can be found in [11], [12].

Experiments on the visual SLAM data can be performed with visual SLAM algorithm based on EKF, SEIF and Graph. Due to time restrictions it was not possible to develop a visual SLAM algorithm based on FAST SLAM.

Since there is no prior information, the Graph SLAM algorithm can be initialized with all positions at zero. Unfortunately the linearization is worse which results in an unstable estimation. Therefore the Graph SLAM algorithm is initialized with subparts of 50 time instances which are each updated 5 times.

In the vSLAM algorithm with EKF, landmarks that have to be removed from the map are deleted by deleting the corresponding cells in the state vector and the covariance matrix. Deleting landmarks from the map in the information filter is more complex. For SEIF and Graph SLAM, the landmarks are removed by using the marginalization lemma. More information can be found in Section 3.6.4.

State vector

The state vector is represented with a 12D vector given in (100).

$$x_i = \begin{bmatrix} p_i \\ \eta_i \\ v_i \\ \omega_i \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \varphi \\ \theta \\ \psi \\ v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{100}$$

with p_i a 3D position, η_i the yxz euler angels, v_i 3D linear velocity and ω_i the 3D angular velocity.

Motion function

The motion function has the following form:

$$\begin{aligned} p_{i+1} &= p_i + R_i\, v_i \Delta \\ \eta_{i+1} &= \eta_i + J_i\, \omega_i \Delta \\ v_{i+1} &= \Gamma\, v_i \\ \omega_{i+1} &= \Lambda\, \omega_i \end{aligned} \tag{101}$$

with diagonal matrices Γ and Λ with coefficients between 0 and 1 to determine the predictability and R the rotation matrix as a function of the Euler angles η. Δ is the sampling period. The angular velocity transform J_i is calculated as follows:

$$J_i = J(\eta) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \dfrac{\sin(\psi)}{\cos(\varphi)} & \dfrac{\cos(\psi)}{\cos(\varphi)} & 0 \\ \tan(\varphi)\sin(\psi) & \tan(\varphi)\cos(\psi) & 1 \end{bmatrix} \tag{102}$$

Note that the motion function does not take relative sensor information into account. The next state is calculated based on the previous state.

Jacobian of the motion function

The Jacobian G of the motion function is:

$$G = \begin{bmatrix} I_{3\times3} & P_i & R_i\,\Delta & 0_{3\times3} \\ 0_{3\times3} & E_i & 0_{3\times3} & J_i\Delta \\ 0_{3\times3} & 0_{3\times3} & \Gamma & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & \Lambda \end{bmatrix} \tag{103}$$

With E the Jacobian matrix of the new Euler angles with respect to the old Euler angles:

$$E_i = E(\eta_i, \omega_i) = \begin{bmatrix} 1 & 0 & (-\omega_y c_\psi - \omega_x s_\psi)\Delta \\ \frac{s_\varphi(\omega_x s_\psi + \omega_y c_\psi)}{c_\varphi^2}\Delta & 0 & \frac{\omega_x c_\psi - \omega_y s_\psi}{c_\varphi}\Delta \\ \frac{\omega_y c_\psi + \omega_x s_\psi}{c_\varphi^2}\Delta & 0 & \frac{s_\varphi(\omega_x c_\psi - \omega_y s_\psi)}{c_\varphi} \end{bmatrix} \quad (104)$$

where $s_{\dots}$, $c_{\dots}$ are rebates of sinus and cosines, respectively. And P the Jacobian matrix of the new position with respect to the old position:

$$P_i = P(\eta, v) = \frac{\partial ef}{\partial \eta^T} p(t+1) = -R[v]_\times \Delta \quad (105)$$

with $[v]_\times$ the skew-symmetric matrix formed by linear velocity v.

Measurement function

$$\hat{z}_i^k = {}^w R_c^T \left(m_{j=c_i^k} - p_i \right) \quad (106)$$

where ${}^w R_c$ is the rotation matrix from world coordinates to camera coordinates.

Jacobian of the measurement function

The Jacobian of the measurement function has the following shape:

$$H_i^k = \begin{bmatrix} -{}^w R_c^T & 2\left[\hat{z}_i^k\right]_\times & 0_{3\times(3+3j)} & {}^w R_c^T & 0_{3\times(J-j)} \end{bmatrix} \quad (107)$$

with $\left[\hat{z}_i^k\right]_\times$ the skew-symmetric matrix formed by the noise free part $\hat{z}_i^k$ of the measurement function.

4.3 Evaluation criteria

This section describes the criteria how the performance of the SLAM algorithms will be evaluated.

4.3.1 Accuracy and consistency

In this section, the metrics are provided to evaluate the accuracy and consistency of the algorithms.

Poses and map

For each pose, the Absolute Error (AE) will be calculated.

$$AE_{p_i} = \left\| \mu_{p_i} - p_{i_{real}} \right\| \quad (108)$$

with $\mu_{p_i}$the estimated mean of the pose and $p_{i_{real}}$ the ground truth position at time i.

The absolute Error will also be calculated for the position each landmark:

$$AE_{m_j} = \left\| \mu_{m_j} - m_{j_{real}} \right\| \quad (109)$$

With $\mu_{m_j}$the estimated mean of the position and $m_{j_{real}}$ the ground truth position of the landmark j.

The vSLAM algorithms associates their own ID to a landmark. To match the landmark positions of the estimated data with the real data, the Matlab function ‘pdist2’ is used. The function finds the two smallest pairwise Euclidean distances to landmark positions in the set of real landmarks for each of the estimated landmarks in the combined state vector.

Furthermore, for both sets of absolute errors the mean of the absolute values will be calculated over time to get the Mean Absolute Error (MAE).

A filter is consistent when the variances and covariances of the real error matches the covariance matrices that are calculated in the algorithms. To check the filter's consistency, the Normalized Estimation Error-Squared (NEES) will be used for the poses and the map:

$$\mathrm{NEES}(\mathrm{p_i}) = \left(\mu_{\mathrm{p_i}} - \mathrm{p_{i_{real}}}\right)^{\mathrm{T}} \mathrm{C}_{\mathrm{p_i}}^{-1} \left(\mu_{\mathrm{p_i}} - \mathrm{p_{i_{real}}}\right) \tag{110}$$

$$\mathrm{NEES}(m_{\mathrm{j}}) = \left(\mu_{\mathrm{m_j}} - \mathrm{m_{j_{real}}}\right)^{\mathrm{T}} \mathrm{C}_{\mathrm{m_j}}^{-1} \left(\mu_{\mathrm{m_j}} - \mathrm{m_{j_{real}}}\right) \tag{111}$$

with $\mathrm{C}_{[.]}$ the associated covariance matrix of the estimates.

A Gaussian estimate is considered as consistent if the corresponding NEES follows a χ^2 distribution. When the NEES values exceed a determined boundary, the NEES does not follow a χ^2 distribution. The state vector of the 2D map consists of 2 degrees of freedom and a 95% one sided acceptance boundary is chosen. This means that the filter is not consistent when values exceed the bound of 5.991. For the visual SLAM experiment, this acceptance boundary is 7.82, with 3 degrees of freedom.

In many cases, the Root Mean Square Error (RMSE) over time is used [13] to measure the error. Normally RMSE is used to calculate the standard deviation of an estimated data set of an ergodic process. In the experiments of this project, the same data set is used. Therefore the input data is ergodic. However, since the NEES gives more insight of the consistency of the filter, RMSE does not provide extra information.

Orientation

In the visual SLAM experiment the orientation is also estimated. The orientation is in Euler angles and will be converted to the orientation error in the magnitude of rotation Φ of the axis-angle representation (φ, e).

$$\varphi_i = \arccos\left(\frac{1}{2}\left(trace\left(\mathrm{R}^{\mathrm{T}}(\eta_i)\mathrm{R}(\hat{\eta}_i)\right) - 1\right)\right) \tag{112}$$

with R the rotation matrix as a function of the Euler angles η.

The NEES is also calculated.

$$\mathrm{NEES}(\varphi_i) = {\varphi_i}^2 \, \mathrm{C}_{\varphi_i}^{-1} \tag{113}$$

with variance C_{φ_i} calculated as follows:

$$\mathrm{C}_{\varphi_i} = \mathrm{V} \, \mathrm{C}_{\eta_i}^{-1} \mathrm{V}_{\mathrm{T}} \tag{114}$$

with $\mathrm{C}_{\eta_i}$the covariance matrix of the Euler angles η_i and V the Jacobian of the function in (112) with respect to $\hat{\eta}_i$.

Used algorithm settings

For the above mentioned performance measures, the real data is required. Therefore the United Kingdom data and the vSLAM algorithm with simulated data are used. The settings for the vSLAM algorithm are 'P.Mmax=1000' and 'P.max_lm_inc_per_step=50'. In total there are 1390 time instances for the United Kingdom data and 500 time instances for the vSLAM algorithm with simulated data.

4.3.2 Computational complexity

Using tic toc functions of Matlab the computation time at each iteration will be measured. In this way an indication can be made for the computational complexity of the online algorithms. Note that measuring small computation times in Matlab is not accurate. According to [14] the code should take more than 0,1 second to run.

Furthermore, the profiler will be used, but when profiling is on, aspects of the JIT (Just In Time Compiler) might be turned off. It can take considerably longer to execute code when profiling is turned on and you can end up with severely distorted understandings of relative execution times. This means that all the algorithms have to run with and without profiling. Therefore the profiler is only used at the last time instant. In this way the computational complexity of each code can be analyzed in detail and the algorithms do not have to be run twice.

To evaluate the computation time the United Kingdom data and the vSLAM algorithm with real data will be used. The settings for the vSLAM algorithm are 'P.Mmax=10000' and 'P.max_lm_inc_per_step=50'. The vSLAM algorithm is computed for 120 time instances. In Graph SLAM the total time of an update will be measured. For the online algorithms several times will be measured per time instant: Motion, measurement, update state estimation(only SEIF), sparsification (only SEIF) and the total time of a execution per time instant. In this way, the remaining time can also be calculated. This remaining time is needed for matching features and the bookkeeping of landmarks.

Reading memory in MATLAB is not possible. According to [15], Matlab does not define what "used memory" is. For example if an array is shrinked, the OS can decide if the freed part of the memory is reused directly or not. Before it can be re-used, the contents is overwritten with zeroes, and the OS decides when this is done. With iteratively growing arrays similar problems occur. It is tried to read the memory usage by subtracting memory usage at specific times from each other. Unfortunately no useful information is generated. For example, it consists of negative numbers, which means that less memory is used before the algorithm is called. A second option is to use the profiler, but also the profiler does not give useful information about the used memory by the algorithm.

5. Results

In this chapter the results of the experiments, outlined in the previous chapter, are given.

5.1 Accuracy and consistency on 2D UK data

The estimated poses and landmarks by the SLAM algorithms on the 2D data are shown in Figure 16. The SEIF algorithm is computed with 5 active algorithms and FAST SLAM is computed with 50 particles. The estimation by the Graph SLAM algorithm is after one update.

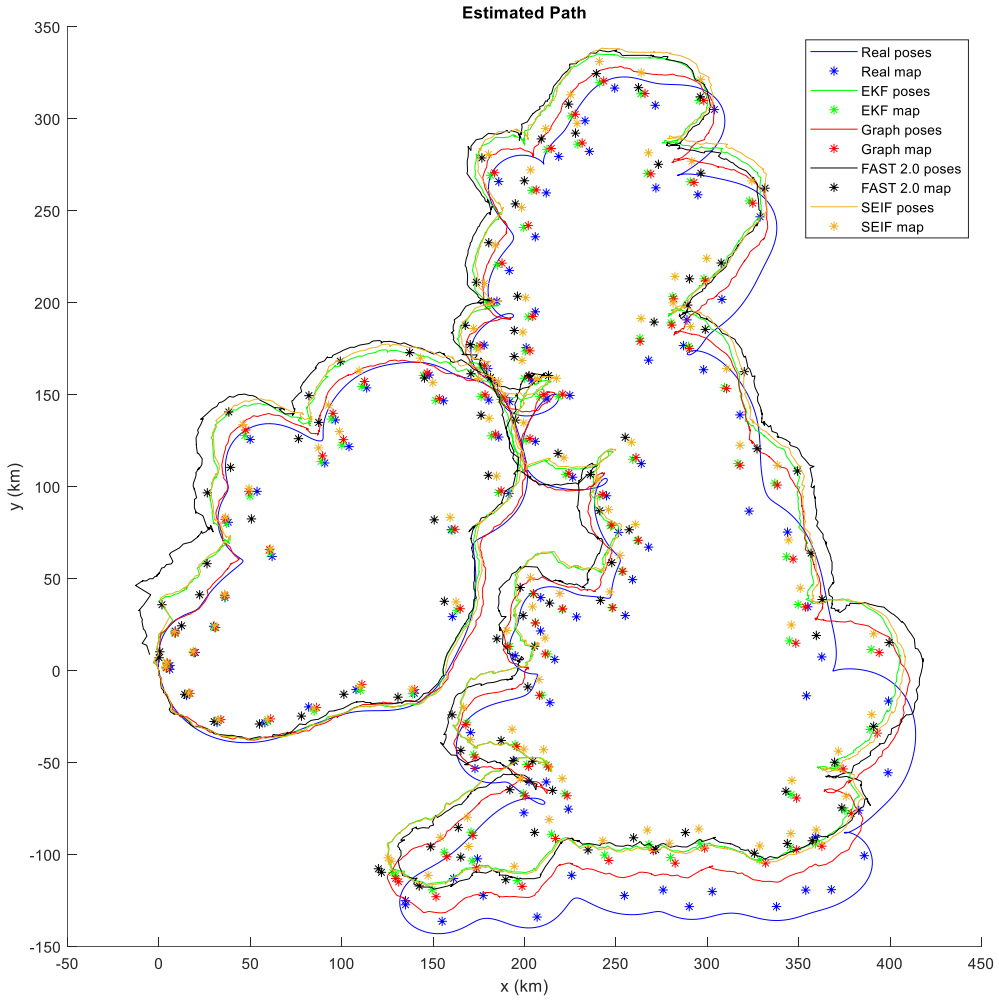


Figure 16 Estimated path of the algorithms in with 2D data

5.1.1 Pose

In Figure 17 (a), (b), (c) and (d), the pose errors of the algorithms are given. Except for FAST SLAM 2.0 the algorithms have the same shape. The estimation between time instants 500 and 900 are relatively high. The estimations at this time range corresponds to the estimates on the right of the plot, shown in Figure 16. EKF and SEIF are almost identical. The chosen number of active features of the SEIF algorithm does not change the estimated pose. The Graph SLAM has the highest accuracy. The pose error of FAST-SLAM is worse, compared to other algorithms. The is no clear winner in terms of the number of particles. The setting with 10 particles has until i=700 the lowest error, but after this time the FAST algorithm with other number of particles have a lower error. FAST SLAM with 20 particles gives the worst estimation.

For all algorithms, it can be observed in Figure 17 (d), (e), (f) and (g) that between time stamps i=600 and i=900 the NEES values are above 5.991, which is the 95% one sided acceptance boundary of for 2 degrees of freedom. The NEES of Graph SLAM is even higher in this time range. The pose error of SEIF with 5 and without sparsification is almost identical to EKF, with slight differences at time instances after i=1200. The NEES of SEIF with 4 active landmarks is signifyingly higher. Since the probability distribution of FAST SLAM by a function, the NEES cannot be computed directly. Other test variables can be computed for particle representations of a pdf [4], but due to time limitations there was no time left to work this out.

5.1.2 Map

Also for the map error given in Figure 18 (a), (b), (c) and (d), the errors of EKF, SEIF without sparsification and Graph are identical. The map error of SEIF is relatively worse when applying sparsification. It can be observed that even with 50 active landmarks, the algorithm could not match SEIF without sparsification. However, the error of the last landmarks is still lower when the active landmarks are increasing. Furthermore, for the map error in FAST SLAM, there is no relation between the number of particles and the map error. FAST SLAM with 10 particles have overall the lowers map error.

The results of the map also shows that the filter in not consistent at certain places and goes beyond the 5.991 acceptance boundary. It shows here as well that the NEES of the EKF, SEIF without sparsification and Graph seem to be equal.

5.1.3 MAE

In Table 4 the MAE of EKF is given. The high peak error between i=600 and i=900 ensure that the mean error of both the pose and the map is significantly increased.

Table 4 Mean Absolute Error EKF 2D map

Pose (km)	19,13
Map (km)	8,590

In Table 5 the MAE of SEIF is given. It can be seen that the average error of the pose slightly increases and the map error increases significantly with more active features. There is still a big difference in error between 50 active features and all active features.

Table 5 Mean Absolute Error 2D SEIF

Active lm	4	5	6	7	8	50	All
Pose (km)	19,8210	19,5655	19,7628	19,7833	19,5590	19,5641	19,433
Map (km)	13,6132	13,029	13,2527	13,1154	12,8565	12,4693	8,4729

In Table 6 it is visible that the second update of Graph SKAM does not change the error. Furthermore, the values are low compared other algorithms.

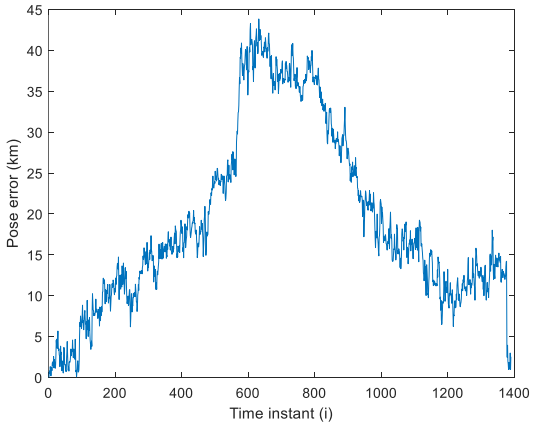
Table 6 Mean Absolute Error Graph 2D map

Updates	1	2
Pose (km)	9,9076	9,9076
Map (km)	7,9215	7,9215

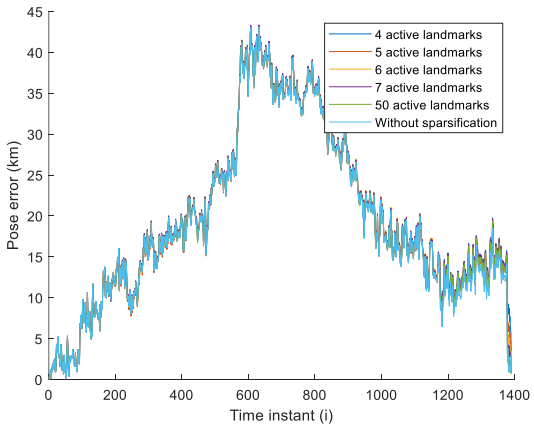
In Table 7 the MAE of FAST 2.0 is shown. Here it can also be observed that FAST 2.0 with 10 particles has the lowest error in both the pose and the map.

Table 7 Mean Absolute Error FAST 2.0 2D map

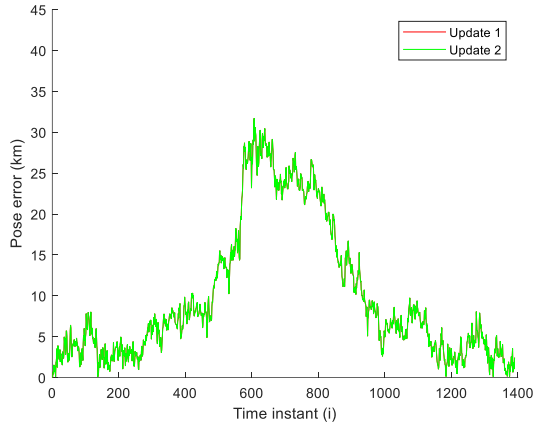
Particles	5	10	15	20	50
Pose (m)	21,2005	15,9390	17,2480	38,4606	19,7086
Map (m)	14,6202	11,6669	13,2907	19,0153	13,8570



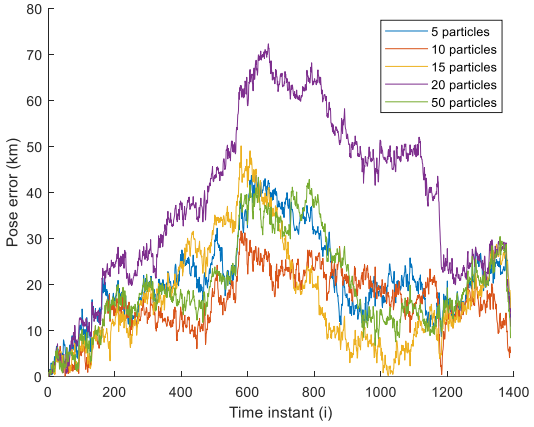
(a) Pose error EKF



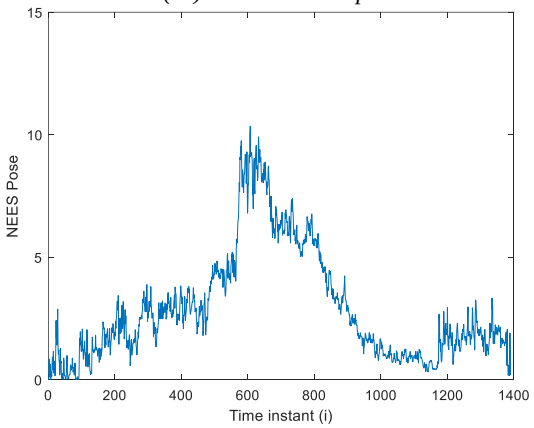
(b) Pose error SEIF



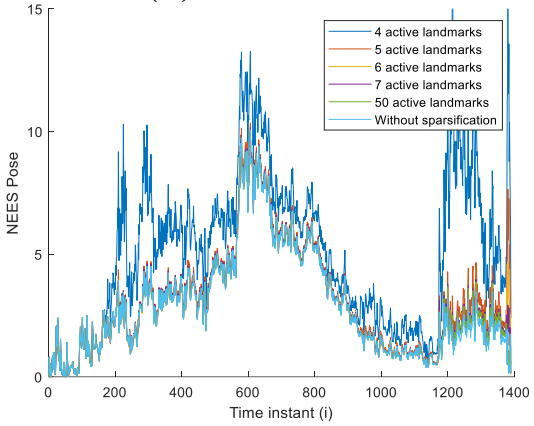
(c) Pose error Graph



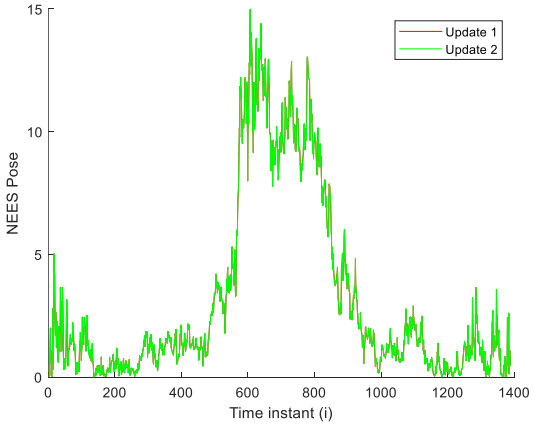
(d) Pose error Fast 2.0



(e) NEES of the pose EKF

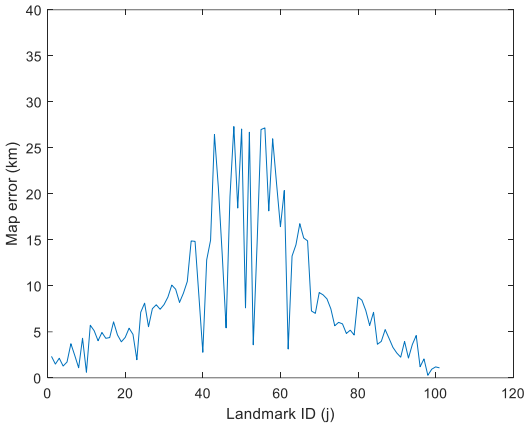


(f) NEES of the pose SEIF

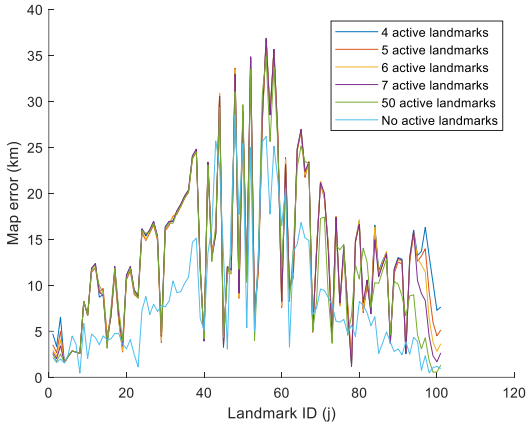


(g) NEES of the pose Graph

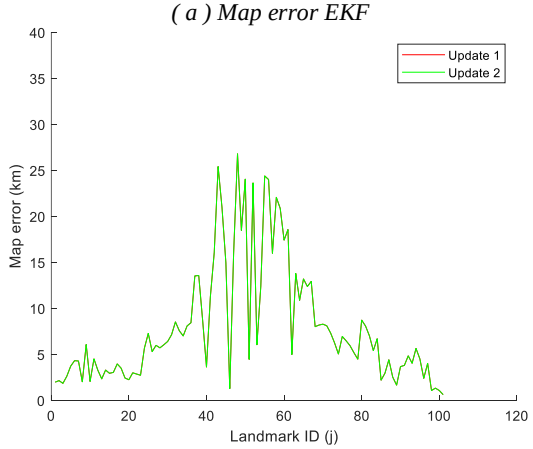
Figure 17 The error and NEES of the pose on 2D data



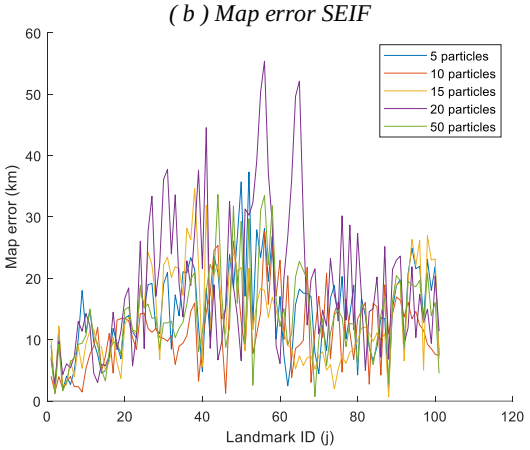
(a) Map error EKF



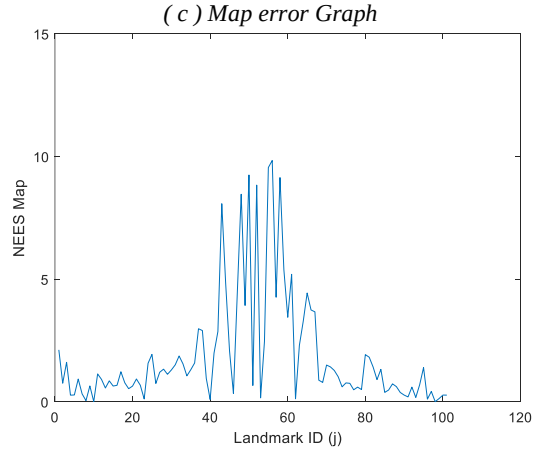
(b) Map error SEIF



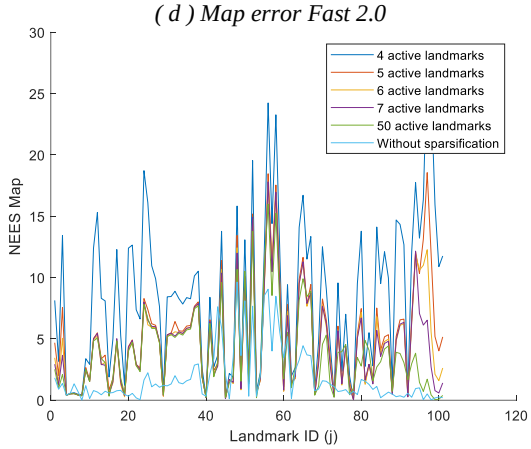
(c) Map error Graph



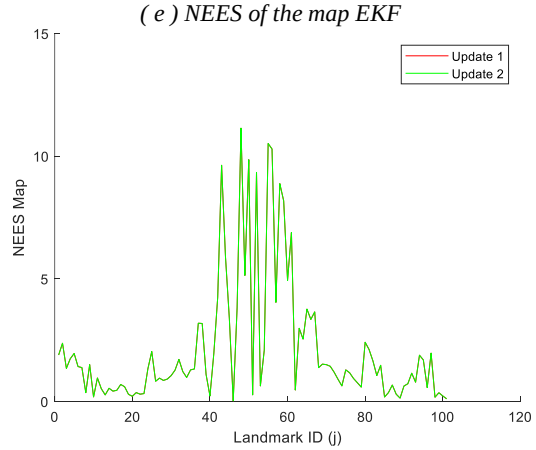
(d) Map error Fast 2.0



(e) NEES of the map EKF



(f) NEES of the map SEIF



(g) NEES of the map Graph

Figure 18 The error and NEES of the map on 2D data

5.2 Accuracy and consistency on 3D simulated vSLAM data

In this section the results of tools to measure the accuracy of the simulated vSLAM data are described. Figure 19 shows the estimation of the Graph-based vSLAM algorithm based on the simulated data.

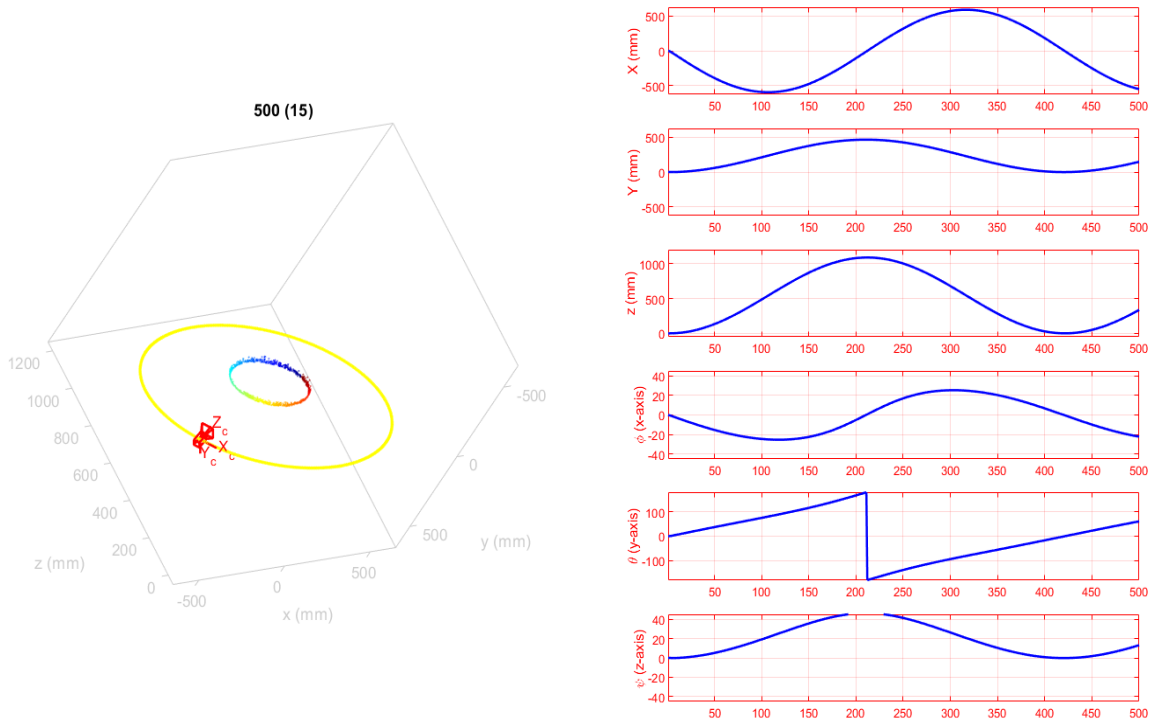


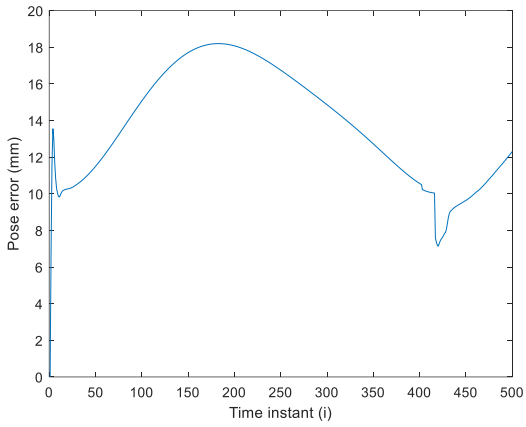
Figure 19 3D plot of the Graph-based vSLAM algorithm

The position of the camera starts at zero. Then it is moving clockwise around the globe. At about i=420 the camera has made a circle and at that moment the loop closes with the positions estimated at i=1.

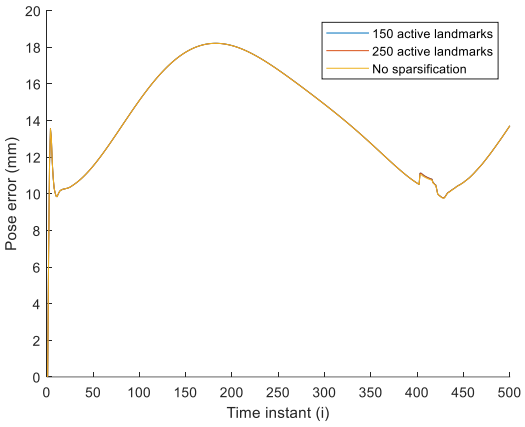
5.2.1 Pose

The pose error of the algorithms on the simulated vSLAM data is given in Figure 20 (a), (b) and (c). The shape of the algorithm is more or less the same. When the loop closes at i=420, the error is the lowest. At this point there are small differences between the EKF and SEIF algorithm. The maximum error is around i=175. The error increases faster before this peak than it does after the peak. The Graph SLAM has overall the lowest error. There is also no difference between the chosen number of active landmarks in SEIF. The same holds for the number of updates in Graph SLAM. Note that the number of updates in Graph SLAM are the number of global updates, which are performed after generating a prior path using submaps.

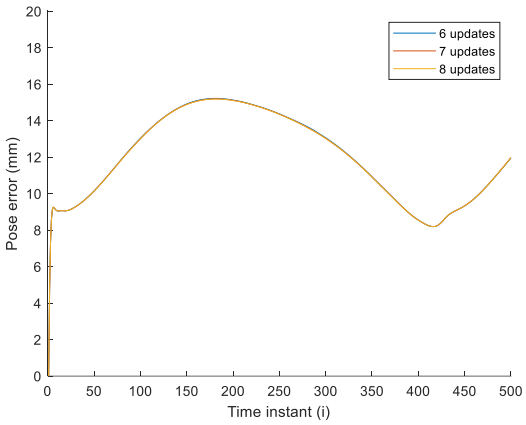
The NEES of the pose of the algorithms are shown in Figure 20 (d), (e) and (f). The NEES of the EKF, SEIF without sparsification and Graph are almost identical. Graph SLAM shows a slightly higher NEES. Remarkable is the increase of the NEES of the SEIF algorithm with sparsification. From i=200 the NEES values are greater than 7.82, which means the filter is not consistent at this time.



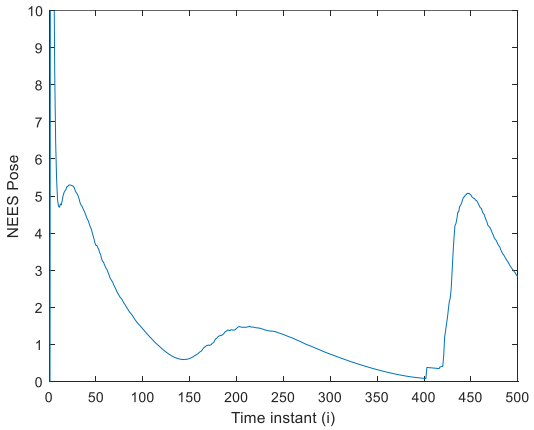
(a) Pose error EKF



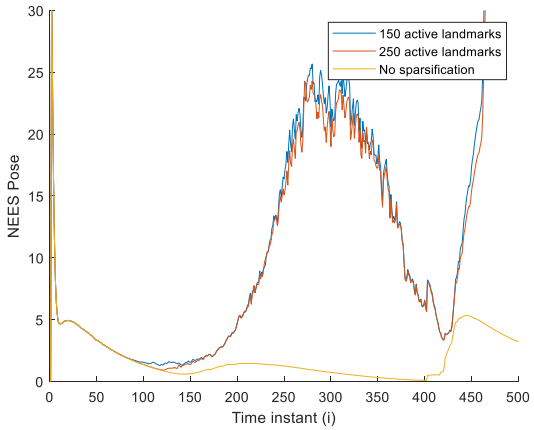
(b) Pose error SEIF



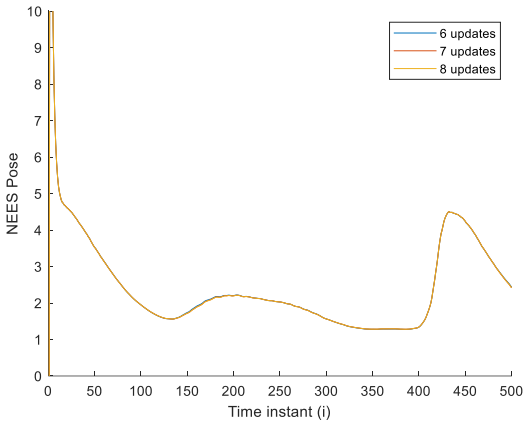
(c) Pose error Graph



(d) NEES of the pose EKF



(e) NEES of the pose SEIF



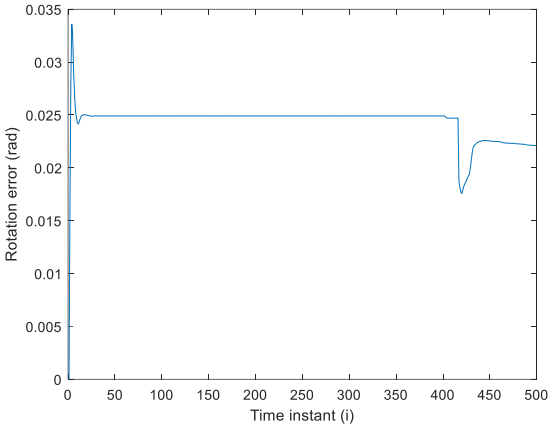
(f) NEES of the pose Graph

Figure 20 Accuracy and consistency of the pose on simulated vSLAM data

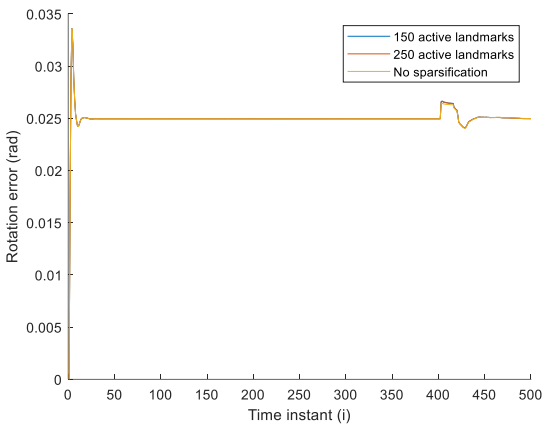
5.2.2 Rotation

The rotation error and the NEES of the rotation of the algorithms are shown in Figure 21 (a) to (f). There are small differences in error at the rotations estimated by the EKF and SEIF algorithms. The Graph algorithms provide the smallest error. From about i=10 to i=400 the EKF and SEIF the error is constant, where the error of the estimation of Graph algorithm has a more oscillating behavior. At the time where the loop closes, there is a difference in the shape of the error between the algorithms. The error of EKF and Graph is decreasing, while the SEIF error increases.

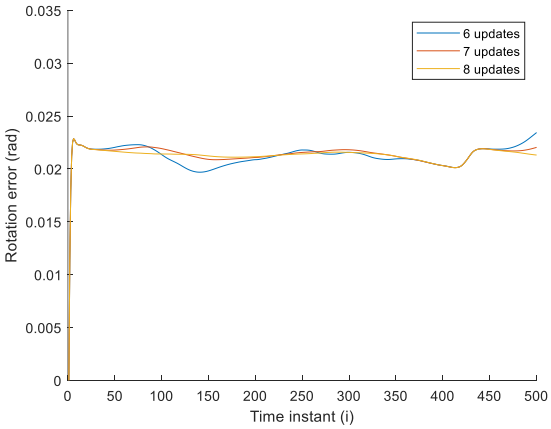
At i=0 the NEES of the rotation of all algorithms is very high. After that, it starts to decrease to more or less 0.2. At the loop closing at i=400 the NEES starts to increase. Except for the high peak in the beginning, the NEES of all algorithms are below the bound of 7.82. The EKF algorithm shows the lowest NEES, followed by the NEES of the Graph SLAM algorithm. In this case, the SEIF algorithm without sparsification shows a NEES which is equal to EKF. The SEIF algorithm with sparsification shows higher values from i=100. The values are also higher when the loop closes at i=400.



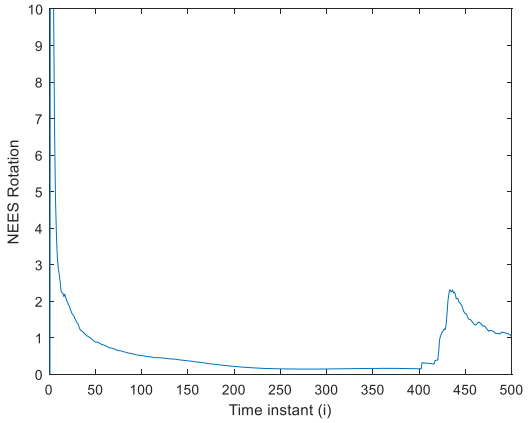
(a) Rotation error EKF



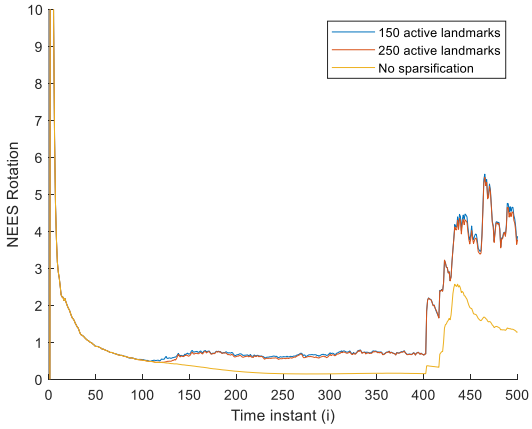
(b) Rotation error SEIF



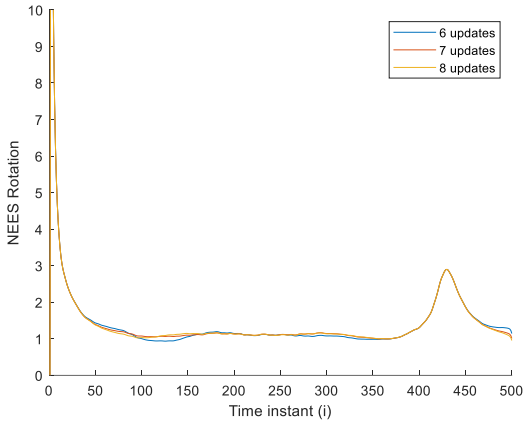
(c) Rotation error Graph



(d) NEES of rotation EKF



(e) NEES of rotation SEIF



(f) NEES of rotation Graph

Figure 21 Accuracy and consistency of the rotation on simulated vSLAM data

5.2.3 Map

As described in 4.3.1 the settings for the bookkeeping in the vSLAM algorithm are defined as 'P.Mmax=1000' and 'P.max_lm_inc_per_step=50'. In Figure 22 can be seen that only at i=100, the maximum number of landmarks added to the pool per step is reached. Then there is a small number of landmarks that are added to the pool, with an average of about 8 landmarks. At about i=110 the pool is full and there are landmarks deleted to insert new landmarks.

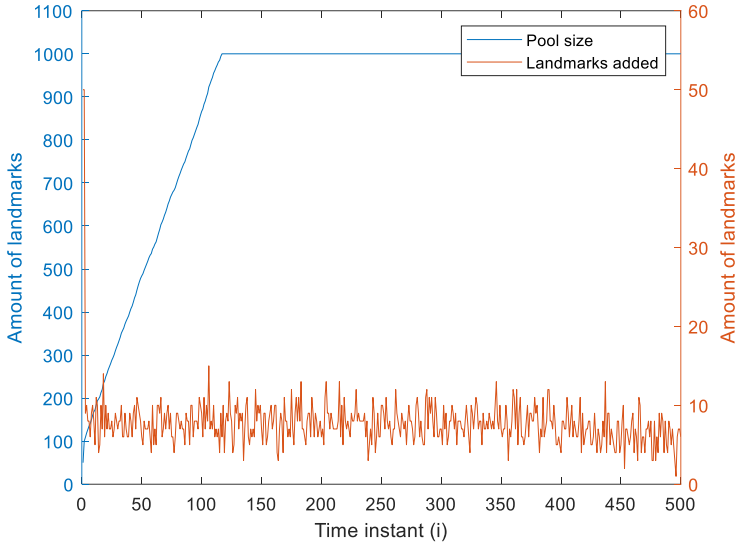
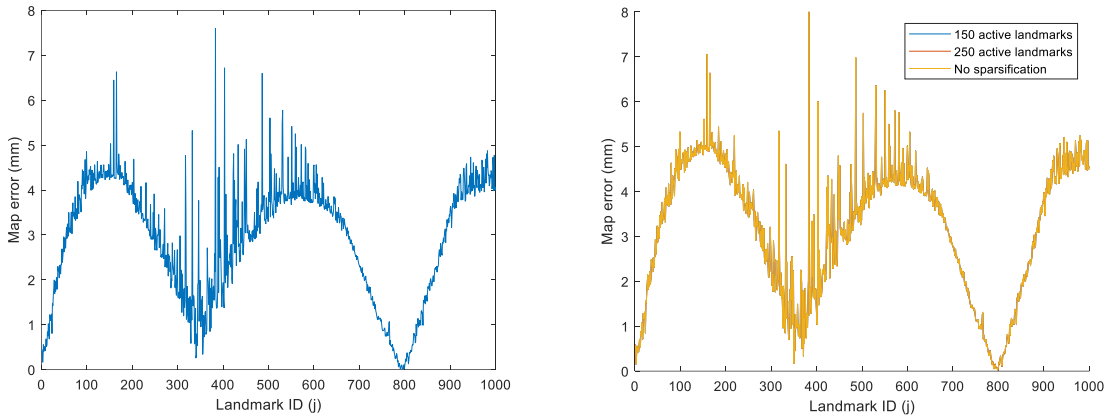


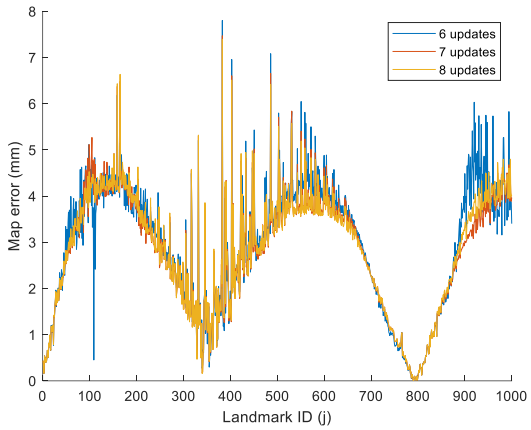
Figure 22 Bookkeeping on the vSLAM simulated data

The map error and map are shown in Figure 23 (a), (b) and (c). The NEES of the map is shown in Figure 23 (d), (e) and (f). The landmarks are sorted by when they are initialized. The map error and the NEES of the map of all three algorithms have a shape of three arcs. There are no big differences in map error between the algorithms. The Graph SLAM shows overall the lowest error. There are slight differences between the amount of updates, with 8 updates the lowest error. The EKF shows a sightly higher error, followed by the SEIF algorithm. The map error of SEIF is not different with or without sparsification. Furthermore, the NEES of all the algorithms consist of rather high peaks which are much higher than the bound of 7.82. The NEES of the Graph SLAM algorithms show the lowest values, but the peaks are still above 7.82. While the shape of the NEES of EKF is the same, the values are a bit higher. The consistency of the map of SEIF with 150 and 250 active landmarks is worse. Around landmarks ID's j=0, j=350 and j=800 there are some values below the bound of 7.82. Overall the algorithm with sparsification is not consistent. However the SEIF algorithm without sparsification shows lower results, which are almost identical to EKF.

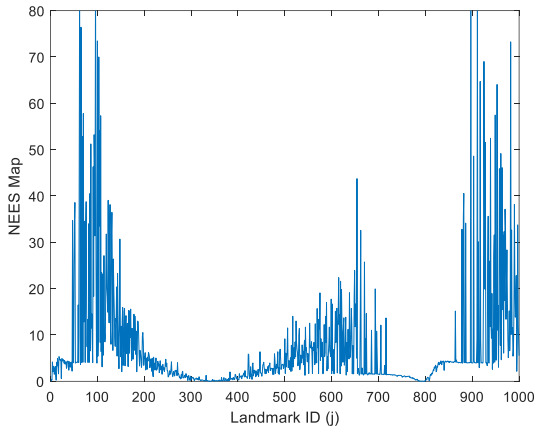


(a) Map error EKF *(b) Map error SEIF*

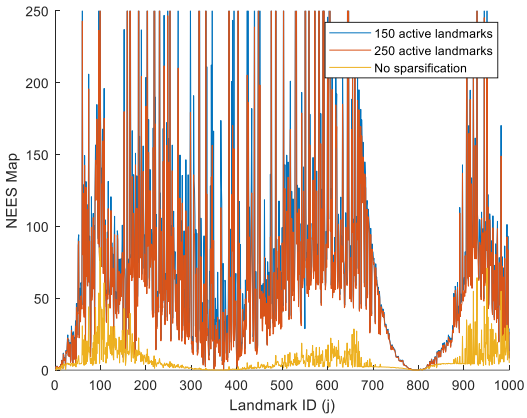
Figure 23 Accuracy and consistency of the map on simulated vSLAM data



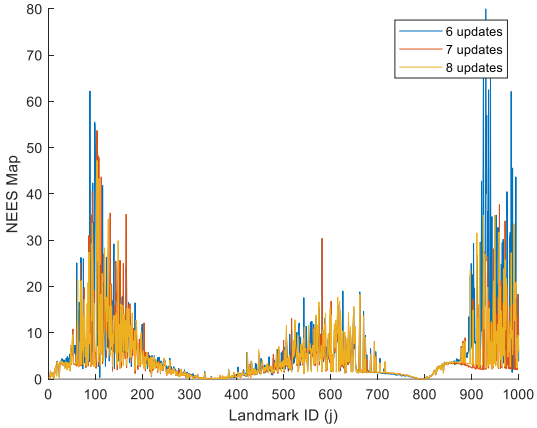
(c) Map error Graph



(d) NEES of map EKF



(e) NEES of map SEIF



(f) NEES of map Graph

Figure 23 Accuracy and consistency of the map on simulated vSLAM data

5.2.4 MAE

Table 8 shows the MAE of the EKF SLAM algorithm for the simulation data.

Table 8 Mean absolute error EKF vSLAM simulation data

Position (mm)	13,7029
Rotation (rad)	0,0475
Map (mm)	2,9034

Furthermore, the MAE of SEIF algorithms in Table 9 show that the error is similar compared to EKF.

Table 9 Mean absolute Error SEIF vSLAM simulation data

Active landmarks	150	250	No sparsification
Position (mm)	13,967	13,965	13,962
Rotation (rad)	0,0475	0,0475	0,0475
Map (mm)	3,1886	3,1874	3,1875

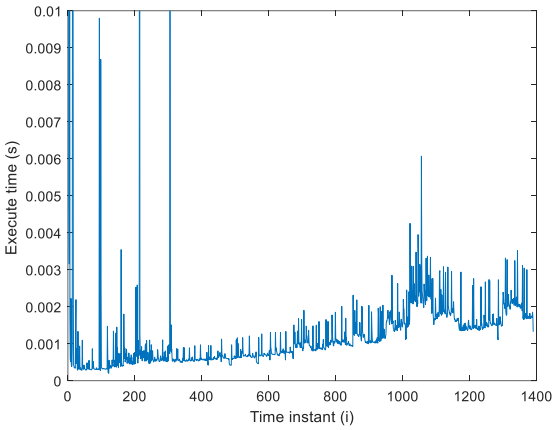
From the MSE values in Table 10 can be also be seen that the 7th and 8th update are identical. The accuracy is the lowest of all algorithms.

Table 10 Mean absolute error Graph vSLAM simulation data

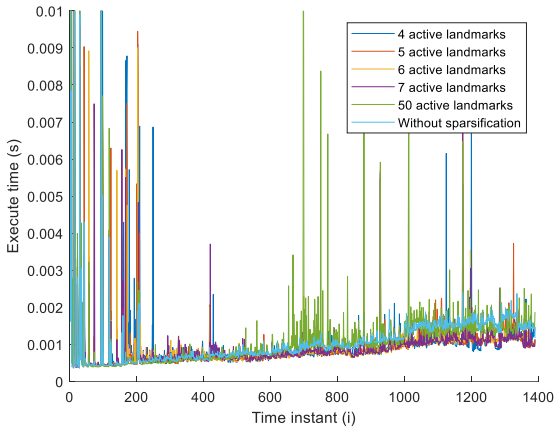
Update	1	2	3	4	5	6	7	8	9	10
Position (mm)	47,77	32,87	15,23	12,69	12,04	11,93	11,92	11,92	11,92	11,92
Rotation (rad)	0,052	0,049	0,047	0,047	0,047	0,04	0,047	0,047	0,047	0,047
Map (mm)	31,57	23,56	10,91	4,262	3,506	2,936	2,800	2,821	2,839	2,840

5.3 Computation time on 2D data

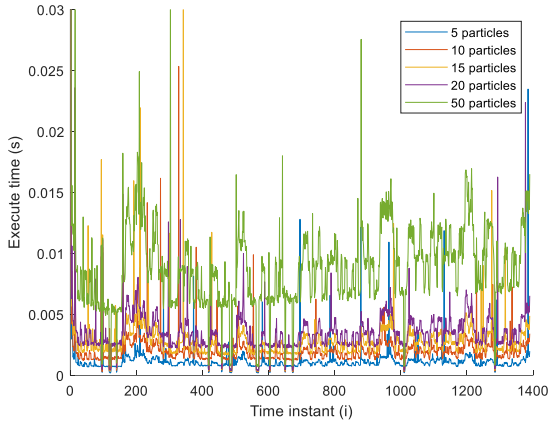
As shown in the Figure 24 the computation time of the SEIF algorithm is the lowest, but shows more high peaks. Also the graph of SEIF has a less exponential growth compared to EKF. The computation of the FAST algorithm is more constant.



(a) EKF



(a) SEIF



(a) Fast 2.0

Figure 24 Computation times of EKF, SEIF and FAST 2.0 on 2D data

The total computation time of EKF, SEIF, FAST 2.0 and Graph are given in Table 11, Table 12, Table 14 and Table 13, respectively. The high peaks are effecting the total time.

Table 11 Total computation time EKF 2D map

Time (seconds)	1,605

Table 12 Total computation time SEIF 2D map

Active features	4	5	6	7	8	50	All
Time (seconds)	1,4599	1,4272	1,3761	1,3613	1,3343	1,5240	1,541

Table 13 Total execution time FAST 2D map

Particles	5	10	15	20	50
Time (seconds)	1,7374	2,8851	3,9861	5,0054	12,3010

Table 14 Total execution time Graph 2D map

Updates	1	2
Time (seconds)	3,9941	3,7043

5.4 Computation time on real vSLAM data

Figure 25 shows the number of landmarks in the pool, added landmarks to the pool and deleted landmarks from the pool. Note that around i=65 the number of landmarks added to the pool is less then the maximum number of landmarks that are allowed to be added. After this time, less useful points are extracted from the images.

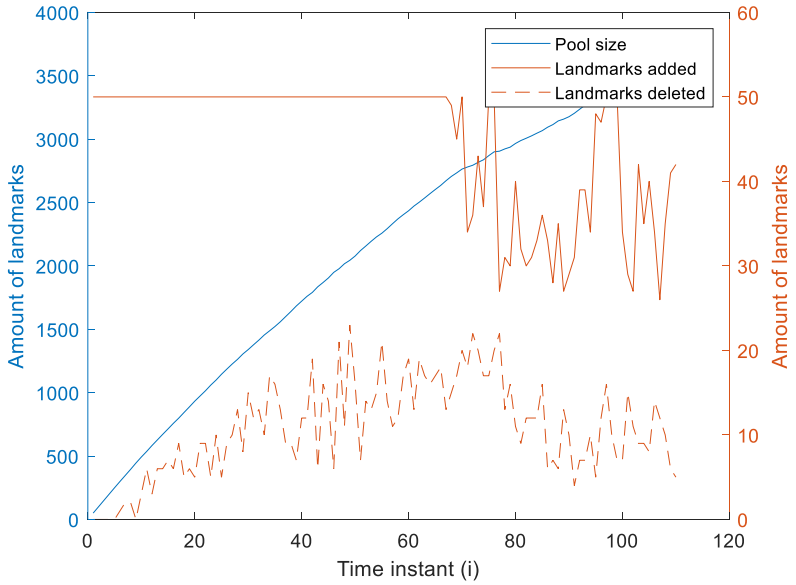


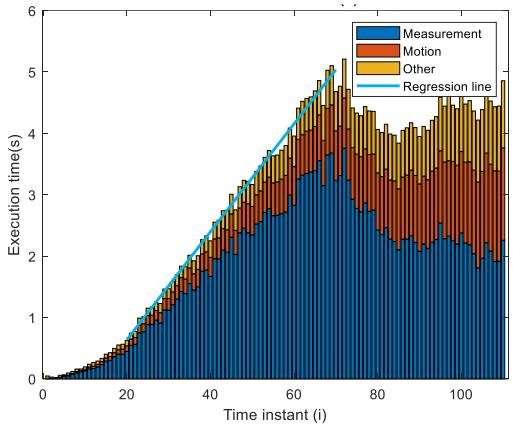
Figure 25 Bookkeeping of landmarks

The total computation times are given in Table 15. It can be observed that the SEIF algorithm with sparsification has the lowest computation time, but the SEIF algorithm without sparsification is even higher than EKF.

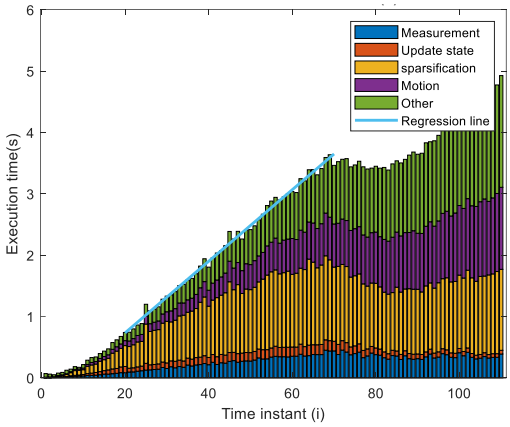
Table 15 Total computation time SEIF real vSLAM data

Algorithm [setting]	EKF	SEIF 150 active	SEIF 250 active	SEIF all active
Time (seconds)	323,28	273,33	274,86	355,57

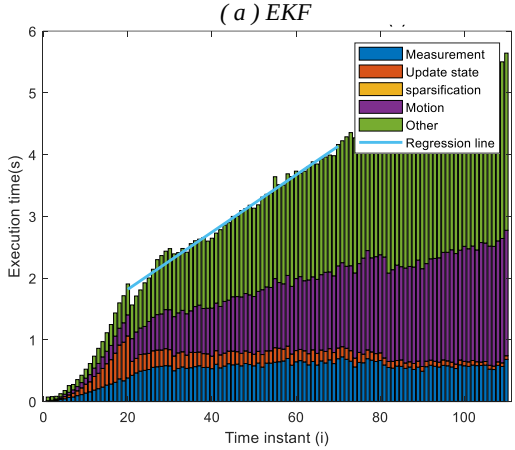
In Figure 26 the computation times of the online algorithms are plotted. Table 16 shows the total computation times of the online algorithms and the values of the regression function. This function is generated with data between i=20 and i=60. Since SEIF 150 and SEIF 250 are almost identical, only SEIF 150 is plotted. The computation time of EKF increases more than SEIF with 150 active landmarks, but after around i=80 the computation time of SEIF is more increasing. After i=80, the sparsification, motion and other parts of SEIF are increasing, while the measurement and update state estimation remain more or less constant. For both EKF and SEIF, it can be seen that the computational complexity is linear in the number of landmarks. Overall, it can be observed that the 'other' parts, mainly consisting of matching feature points and the bookkeeping of landmarks, take a high part of the total computation. Especially in Figure 26 (c), it can be seen that the approximation of the update state starts at i=20. The computation time of this part decreases over 50 percent.



(a) EKF



(b) SEIF 150 active features



(c) SEIF 1000 active features

Figure 26 Computation times vSLAM real data

Table 16 Total execution times of online algorithms

	EKF	SEIF 150	SEIF 250	SEIF 1000
Time (s)	323,287	273,331	274,864	355,573
Fitted function	0,0880 i -1,1221	0,0583 i – 0,4335	0,0601 i – 0,5129	0,0463 i + 0,8912

The total execution times of Graph SLAM per update is given. It can be observed that each update is more or less equal. Until the 10th update the error is still converging, but after 4 updates the error is increasing slower. The total time until update 4 is finished is 517.88 seconds.

Table 17 Execution times per update Graph SLAM

Update	Prior	1	2	3	4	5	6	7	8	9	10
Time (s)	6,4	122,7	125,0	136,6	127,0	127,0	126,8	127,0	126,0	126,2	127,4

6. Discussion

This section discusses the results of the experiments.

6.1.1 Goals

The goal of the experiments was to provide an answer to the following research question:

- What is the performance of several SLAM back-ends within the vSLAM feature-based framework?

Also the following hypothesis was stated:

- A high accuracy of the estimated 3D positions will be at the cost of computational complexity of an algorithm.

The performance on the algorithms is measured on accuracy, consistency and compactional complexity. The accuracy is measured by calculate the difference between real values and estimated values of the combined state vector. The filter is consistent when the variances and covariances of the real error matches the covariance matrices that are calculated in the algorithms. The Normalized Estimation Error Squared (NEES) is used to check the filter's consistency. A Gaussian estimate is considered as consistent if the corresponding NEES follows a χ^2 distribution. When the NEES values exceed a determined boundary, the NEES does not follow a χ^2 distribution. The state vector of the 2D map consists of 2 degrees of freedom and a 95% one sided acceptance boundary is chosen. This means that the filter is not consistent when values exceed the bound of 5.991. For the visual SLAM experiment, this acceptance boundary is 7.82, with 3 degrees of freedom. The computational complexity is determined by measuring the computation times of the algorithms.

The experiments verify that a high accuracy of the estimated 3D positions will be at the cost of computational complexity of an algorithm. Graph SLAM is the clear winner in terms of accuracy and consistency. However, Graph SLAM is a full algorithm with a high computational complexity. The SEIF algorithm is nearly close to the accuracy of EKF SLAM, while the computational complexity of SEIF is less. However the filters' consistency of SEIF with sparsification is less compared to EKF. Especially for the map, the filter is not consistent. The FAST-SLAM, only tested on the 2D data, shows a relatively low accuracy and a higher number of particles do not increase the accuracy.

6.1.2 Accuracy and consistency on 2D data

General

As shown in Figure 17 it can be seen that in all estimations the error between time instants i=500 and i=900 is the highest. The estimations at this time range correspond to the estimates on the right of the plot, shown in Figure 16. A relatively high error in y direction arises due to the effect of a relatively high uncertainty of the angle of the relative measurements which is $\sigma_{\Phi} = 15°$. In this area there are no loop closings and the path is relatively straight compared to the rest of the paths. Therefore it is difficult to perform an accurate estimation in this area based on absolute measurements. When there are more corners, measurements are taken form different perspectives which results in a more accurate estimation. It can be seen that the error decreases when there are more corners in the path. The drop in error around i=1200 is at the loop closing. A loop closing is very effective for SLAM algorithms since landmarks estimated in the past can be compared to measurements at the current time.

A high error in pose also results in a high error in the map, shown Figure 18. New landmarks are initialized based on the estimated pose. Therefore there is a rather high correlation between the error and NEES of the pose and the map. It can also be observed in the map error and NEES of the map that there are outliers with low values. This cannot be seen in the plot in Figure 16. This is due to how the real positions of the landmarks are matched with the estimated positions of landmarks. This is described

in section 4.3.1. In this case poorly estimated positions of landmarks lead to wrong matches of landmark ID's between the set of the estimated positions and the set of real positions of landmarks.

Graph SLAM

The Graph SLAM takes all the poses and landmarks into account and therefore it creates the most links between poses/landmarks estimation. Also a big advantage is that it re-linearizes the path at each update, while other algorithms do not. Therefore it provides the most accurate estimation. This can be seen in the pose errors, shown in Figure 17, and the MEA, given in Table 4 to Table 7. Also the NEES, in Figure 17 and Figure 18, is overall lower. However, the NEES of the pose provided by the Graph SLAM algorithm is higher between time instants i=500 and i=900, compared to EKF and SEIF. While the error is lower, the Graph SLAM algorithms predicts a lower uncertainty. It can be concluded that when a path is taken without loop closings and the algorithm has to rely more on relative measurements, the consistency of the algorithm is lower.

EKF and SEIF

Because EKF and the SEIF have a dual relationship, the error should be similar. However, the sparsification and the update of the state of SEIF is based on an approximation, so this can result in a less accurate estimation and consistency of the filter. Less active landmarks will cause lower accuracy and consistency of the estimation. In the graph of the pose error, shown in Figure 17, this difference is not observable. Except for SEIF with 4 active landmarks the NEES of the pose is also almost equal. The MEA, given in Table 5, shows a small difference in error of the pose. Even without sparsification, the MEA of the pose estimation by the SEIF algorithm is still higher. This means that this small error is caused by the updating step of the state in SEIF.

FAST SLAM

The results of the FAST SLAM algorithm are not as expected. The error of the pose, given in Figure 17, and the map, shown in Figure 18, is large. More particles should result in a more accurate estimation, since it gives a better representation of the probability density. However, this is not the case with the algorithm used for the experiments on the 2D data. When analyzing the particle representation of the probability density during the experiments, it could be observed that there are different area's that have a high probability. An example is shown in Figure 27 on the next page. Especially on the lower side of the plot of the 2D data the probability density is spreading out over more less two groups. The mean of the particles is somewhere in-between. A high probability at different area's lead to a mean that does not provide a good representation of the probability density. At the loop closing, multiple areas with a high probability are reduced to one area. In the 'General' subsection of this section, it is already explained why the error of the pose and map is in general high on the right and lower side of the 2D map. For the FAST-SLAM algorithm this problem causes a probability density in multiple areas, especially because of the conditional independency between landmarks.

6.1.3 Accuracy and consistency on 3D simulated data

General

Because there are no relative measurements, the motion function can only use the linear and angular velocities to predict the next state. These linear and angular velocities have to be corrected by absolute measurements. This creates an increase in the pose error, shown in Figure 20, of all the three algorithms until i=175 and also in the first iterations in the rotation error, shown in Figure 21. Once these linear and angular velocities are corrected, a prediction of the pose can be done more accurately and the error decreases. The loop closing can clearly be seen in the pose and rotation error of all the algorithms.

The shape of the pose error at about i=5 to i=100 is the same as at about i=430 to i=500. In this time range the loop closes and the poses are overlapping. It can also be observed that the error of the rotation of all algorithms is relatively small and between i=20 and i=400 the rotation error is almost constant. The reason for this is that the predicted measurement in the measurement update is very sensitive for a rotation error. This results in an effective correction in the measurement update of the algorithms.

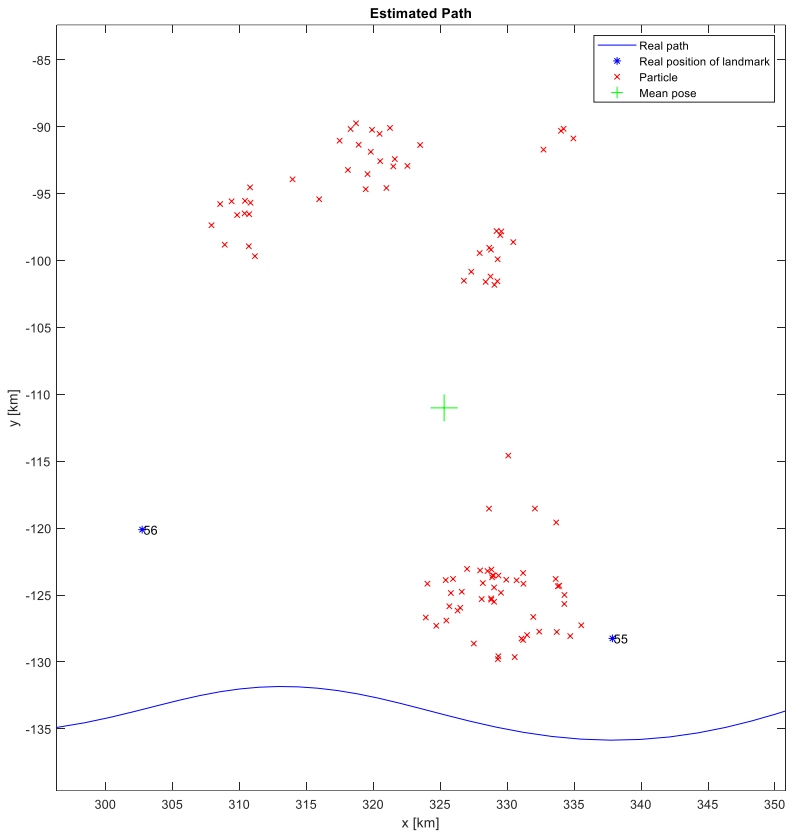


Figure 27 Example of inaccurate calculation of the mean by the Fast SLAM algorithm on 2D data.

The NEES of the pose and rotation, shown in Figure 20 and Figure 21, of all the algorithms is starting very high. Since the system is initialized with a zero variance, a small error already results in an inconsistent estimation. After about i=100 the variance of the pose becomes greater due to the incorporation of measurement and motion noise, which declared the decrease of the NEES of the pose. The small increase in NEES of the pose at a local peak at i=200 is a consequence of the higher error of the pose in this time range. However this is still beyond the boundary of 7.82. At the loop closing around i=400, the NEES of both the pose and the rotation increases, since the algorithms take the low variance at i=1 into account. This results in a reduced variance of the pose and rotation, which causes a higher NEES.

The shape of three arcs of the error and NEES in the map, given in Figure 23, is mostly the effect of the loop closing. Note that the map error and NEES are not shown over time, but for each landmark. However the landmarks are sorted by when they are initialized, so there is a correlation with time. Since around i=110 the pool is already full, the algorithm decides which landmarks have to be removed to add new landmarks. New landmarks will be added at the end of the state vector. It is obvious the dip around the landmark ID j=800 is around the loop closing. However the dip at j=350 could not be explained. The peaks in the map error and NEES of the map are caused by the way how landmark positions of estimated data and real data are matched. In section 4.3.1 is described how the real positions of the landmarks are matched with the estimated positions of landmarks in the experiment. Also in the case of the vSLAM algorithms, poorly estimated positions of landmarks lead to wrong matches of landmark ID's between the set of the estimated positions and the set of real positions of landmarks. This explains the peaks in the map error and NEES of the map. Also the landmark ID's of outliers are not matched correctly.

EKF and SEIF

Furthermore, in this experiment the sparsification and the update of the state of SEIF can result in a less accurate estimation and consistency of the filter. However there is no difference in the error of the pose

(Figure 20), rotation (Figure 21) and map (Figure 23). The only difference between EKF and SEIF in pose and rotation error is around i=400. There is no difference in pose error between SEIF with sparsification and without. However during the execution of the experiments it was experiences that with less active features the error increases.

The NEES of the pose, rotation and the map of the SEIF algorithm with sparsification, given in Figure 20, Figure 21 and Figure 23, is remarkable. Especially the NEES of the map. Overall the NEES is much higher compared to other algorithms and almost at all time instants inconsistent. From about i=210 to the loop closing at i=400 the NEES of the pose is above the boundary of 7.82 and the algorithm is therefore inconsistent in this time range. The error of the pose, rotation and the map of the SEIF algorithm with and without sparsification are globally the same. This means that the calculated variance by the SEIF algorithm with sparsification is worse compared to the algorithm without sparsification. The approximation in the sparsification step in the SEIF algorithm could be an explanation. Another reason could be an error in the code of the sparsification step.

This experiment is highly linear and there is just one loop closing. When there are more loop closings and the input data is more non linear, it would be interesting to see how the SEIF algorism behaves.

Graph

An advantage of Graph SLAM is the re-linearization of the states and the incorporation of all states. This results in a much smoother and lower pose, rotation and map error, shown in Figure 20, Figure 21 and Figure 23. Especially at the loop closing at i=400 there is no instant change in the error of the pose and the map. Furthermore, the NEES of all states is therefore overall slightly lower compared to the other algorithms. The oscillation behavior in rotation error of the Graph SLAM algorithms is caused by the way how the prior information is calculated. As explained in section 4.2.2, the states are initialized by running the algorithm in sections of time. It can already be observed that the higher number of updates leads to a lower oscillating behavior. Therefore it is expected that after a sufficient number of global updates this error should be filtered out due to the re-linearization process of Graph SLAM.

6.1.4 Computation time on 2D data

The measured computation times, shown in Figure 24, include very high peaks. Due to relatively low computation times, the accuracy of the measured times by Matlab is not reliable. Globally, it can be seen that the SEIF algorithm has low computation times compared to EKF. The computation times of the vSLAM algorithm are more accurate since these algorithms involve a much bigger combined state vector. Therefore a more detailed evaluation about EKF, SEIF and Graph is done in the next section.

It can be seen that the implementation of FAST SLAM increases linearly with a low slope. However this computation time depends heavily on the number of particles. It look like this implementation is of the order $O(N\,J)$. Compared to the theory a better implementation could make the computational complexity to $O(N\log(J))$.

6.1.5 Computation on 3D simulated data

SEIF is based on the information filter and therefore it can incorporate measurements less costly compared to EKF. This can be seen in the results, shown in Figure 26. Another advantage is that SEIF can achieve even less computation times when applying sparsification.

It was expected that EKF should have a total cost known to be $O(n^3)$ with n the number of landmarks and $O(1)$ for SEIF. While SEIF performs worse than expected, the EKF algorithm is with $O(n)$ better than expected.

It can also be concluded that the bookkeeping and feature matching requires a big part of the total computation time. Therefore the total time of the vSLAM algorithm depends less on the SLAM back end algorithm.

Since the Graph SLAM algorithm is a full algorithm, the computational complexity is not comparable in execution time compared the online algorithms. However after 4 update there are already competitive results in accuracy compared to the online algorithms. After 4 updates the total time of the Graph SLAM algorithm is 517.7 seconds, while the runtime of EKF is 323,3 seconds.

6.2 Limitations

During the project the following limitations were experienced:

- The vSLAM experiments are tested in a smooth environment. Both the simulated data and the data acquired from captured images, create a straight circle around the globe. Capturing images with a handheld device can be more nonlinear. Also an environment with more loop closings could provide a more precise result of the accuracy for the 3D visualization of the human skin. Furthermore, the accuracy of the algorithms could deviate more.
- The computational complexity of the algorithms in Matlab can only be roughly estimated. A more low-end code could provide a more precise estimation of the computational complexity.
- During the implementation of SEIF and Graph for the experiments, it was found that calculations with matrices did not remain symmetrical. This was due to rounding errors that were caused by Matlab. Since the information matrix has to be inverted, this could have high implications on the results. This had to be repaired by adding costly operations.
- The map error of the vSLAM simulated data could not be evaluated in detail. A solution could be to compose an overview of on which time stamp each landmark is initialized. Furthermore, a map of the positions of the estimated map and the real map could be helpful what causes the error.
- The FAST SLAM algorithm does not work in the vSLAM algorithms. Therefore the comparison of the results were not complete.

7. Conclusions and recommendations

7.1 Conclusion

In this project the goal was to find several SLAM back-end algorithms that can perform in a feature-based vSLAM framework that is capable of running on a handheld device that scans a target area by manually pointing the device around this area. To achieve this goal, three research questions and a hypotheses were stated.

The first research question was: Which SLAM back-ends are suitable for the mapping on a handheld device? Different filters were presented that can be used for a SLAM back-end algorithm. Based on these filters, three common SLAM algorithms were selected in addition to EKF SLAM that could perform competitive in a vSLAM framework: SEIF SLAM, Graph SLAM and FAST SLAM.

The second question was: What are the working principles of these algorithms? The working principle of the algorithms is explained in this report, including a mathematical derivation. The theoretical background is explained from a probabilistic perspective. Furthermore, the decisions that were made to make the algorithms interesting in performance, is explained.

The next question is answered using the experiments: What is the performance of these algorithms within the vSLAM feature-based framework? The following hypothesis is also verified using the experiments: A high accuracy of the estimated 3D positions will be at the cost of computational complexity of an algorithm. For this experiment seven scripts were made to perform experiments of the EKF, SEIF, Graph and FAST SLAM on 2D data and EKF, SEIF and Graph on 3D visual SLAM data. The results show that, overall, the Graph SLAM algorithm provides the highest accuracy and consistency. However, it is computationally expensive and therefore it could not be run in real-time. The online algorithms EKF and SEIF have a dual relationship and this can be seen in the results of the experiments. Both algorithms are computationally more attractive compared to Graph SLAM and can therefore operate in real-time. This is however at the cost of the accuracy. Both EKF and SEIF show almost the same accuracy with a sufficient number of active landmarks in SEIF, but SEIF is computationally more attractive. However, the needed computation power is higher than expected. It is also shown that the sparsification step in SEIF ensures a low constancy of the filter. Therefore this algorithm should be tested on more nonlinear data with more loop closings to give a more precise evaluation. FAST-SLAM is only tested on the 2D data. More particles did not give a better accuracy on the 2D date. However, according to the theory, this algorithm could perform better on more nonlinear data with more loop closings, compared to EKF and SEIF. Depending on how much particles are needed, FAST-SLAM could compete with EKF in terms of computational complexity.

To conclude, the overall research shows that there are several SLAM-back ends that can perform in a feature-based framework. Depending on the desired accuracy, the available computation power, the available knowledge to implement and the need to perform in real-time, a SLAM back-end can be chosen. However, to measure the performance in a more realistic environment, the algorithms need to be tested in more nonlinear situation with more loop closings.

7.2 Recommendations

To provide a better estimation, adding prior information such as information from a gyroscope or an accelerator could improve the accuracy of the algorithms in general.

The vSLAM framework contains an algorithm to provide correspondences of the measurements. However, data association could also be performed using the SLAM back-end algorithm[4]. Some algorithms have several advantages above the general method. For example, in FAST SLAM, the data association can be made per particle. This means that different data associations can be made with a single measurement at the same time. This improves the consistency of the filter and makes is more robust. Also other algorithms have their own way to find correspondence in an efficient way.

The algorithms are not tested on robustness. According to the theory of the SLAM algorithms the parametric algorithms cannot deal well with outliers. However, FAST-SLAM has a implementation for removing outliers. In a real situation outliers may occur. It is recommend to find methods to deal with outliers.

As described earlier, to measure the performance in a more realistic environment, the algorithms need to be tested in more nonlinear situations with more loop closings.

Appendix A

Sparsification of SEIF derived form a probabilistic point of view

Before applying the sparsification, probability (38) is rewritten in the following form:

$$p(a|c)\,p(b|c)\,p(c) = \frac{p(a|c)p(b|c)}{p(c)} = p(a|c)\frac{p(c)}{p(c)}p(b|c)\,p(c) \tag{115}$$

(38) Next the standard form can be applied to the SLAM case. For readability the conditioning variables $u_{1:i}$, $z_{1:i}^{1:k}$, $c_{1:i}^{1:k}$ in each probability are removed:

$$\begin{aligned} p(y) &= p(x_i\, m^+, m^0, m^-) \\ &= p(x_i|\, m^+, m^0, m^-)\, p(\, m^+, m^0, m^-) \\ &= p(x_i|\, m^+, m^0, m^- = 0)\, p(\, m^+, m^0, m^-) \end{aligned} \tag{116}$$

where in the last row the set of passive landmarks m^- is set equal to 0, since there is no link between x_i and m^-. Now the approximation using conditional independence can be applied in the same manner as in (38).

$$\begin{aligned} &p\big(x_i|\, m^+, m^0, m^- = 0,\ u_{1:i},\ z_{1:i}^{1:k},\ c_{1:i}^{1:k}\big)\, p\big(\, m^+, m^0, m^-|\ u_{1:i},\ z_{1:i}^{1:k},\ c_{1:i}^{1:k}\big) \\ &\quad \approx p\big(x_i|\, m^+, m^- = 0,\ u_{1:i},\ z_{1:i}^{1:k},\ c_{1:i}^{1:k}\big)\, p\big(\, m^+, m^0, m^-|\ u_{1:i},\ z_{1:i}^{1:k},\ c_{1:i}^{1:k}\big) \\ &\quad \approx \frac{p\big(x_i, m^+|\, m^-=0,\ u_{1:i}, z_{1:i}^{1:k}, c_{1:i}^{1:k}\big)}{p\big(m^+|\, m^-=0,\ u_{1:i}, z_{1:i}^{1:k}, c_{1:i}^{1:k}\big)}\, p\big(\, m^+, m^0, m^-|\ u_{1:i},\ z_{1:i}^{1:k},\ c_{1:i}^{1:k}\big) \end{aligned} \tag{117}$$

To calculate the sparse matrix $\widetilde{\Omega}_{i+1}$ from (117) first calculate each individual probability is calculated: Ω_i^1 for $p\big(x_i, m^+|\, m^- = 0,\ u_{1:i},\ z_{1:i}^{1:k},\ c_{1:i}^{1:k}\big)$, Ω_i^2 for $p\big(m^+|\, m^- = 0,\ u_{1:i},\ z_{1:i}^{1:k},\ c_{1:i}^{1:k}\big)$ and Ω_i^3 for $p\big(\, m^+, m^0, m^-|u_{1:i},\ z_{1:i}^{1:k},\ c_{1:i}^{1:k}\big)$. Afterwards the individual matrices will be combined get the sparse matrix $\widetilde{\Omega}_i$.

To get Ω_i^1 and Ω_i^2 first Ω is conditioned on $m^- = 0$ by extracting the cells from Ω related to x_i, m^+, m^0 but m^- using projection matrix F_{x_i,m^+m^0}:

$$\Omega_i^0 = F_{x_i,m^+m^0}\, F_{x_i,m^+m^0}^T\, \Omega\, F_{x_i,m^+m^0}^T F_{x_i,m^+m^0} \tag{118}$$

To get Ω_i^1 the matrix inversion lemma is used to marginalize m^0 from Ω_i^0:

$$\Omega_i^1 = \Omega_i^0 - \Omega_i^0 F_{m^0}\left(F_{m^0}^T \Omega_i^0 F_{m^0}\right)^{-1} F_{m^0}^T\, \Omega_i^0 \tag{119}$$

The calculation of Ω_i^2 is done in the same manner. In this case variables x_i and m^0 are marginalized from Ω_i^0:

$$\Omega_i^2 = \Omega_i^0 - \Omega_i^0 F_{x_i,m^0}\left(F_{x_i,m^0} \Omega_i^0 F_{x_i,m^0}\right)^{-1} F_{x_i,m^0}\, \Omega_i^0 \tag{120}$$

The matrix Ω_{i+1}^3 is calculated by marginalizing x_i from Ω_i:

$$\Omega_i^3 = \Omega_i - \Omega_i F_x\left(F_x^T \Omega_i F_x\right)^{-1} F_x^T\, \Omega_i^0 \tag{121}$$

The sparse matrix $\widetilde{\Omega}_{i+1}$ is calculated by combining the matrix calculated before, according to (117):

$$\widetilde{\Omega}_i = \Omega_i^1 - \Omega_i^2 + \Omega_i^3 \tag{122}$$

Appendix B

Detailed explanation of filling the information matrix of Graph SLAM

For every motion event the curly embraced equations (60) and (61) have to be calculated. The calculated matrix has the shape of the matrix where the underbrace points to. Each element refers to a location in Ω (Figure 8) for the calculated matrix in (60) and ξ (Figure 9) for the calculated vector in (61). The format of the location is 'n m', with n the row label and m column label.

$$\begin{matrix}[x_{i-1}\ x_i]\underbrace{\begin{bmatrix}-G_i^T\\ 1\end{bmatrix}R_i^{-1}[-G_i\ 1]}\begin{bmatrix}x_{i-1}\\ x_i\end{bmatrix}\\ \begin{bmatrix}x_{i-1,x}\,x_{i-1,x} & x_{i-1,x}\,x_{i-1,y} & x_{i-1,x}\,x_{i,x} & x_{i-1,x}\,x_{i,y}\\ x_{i-1,y}\,x_{i-1,x} & x_{i-1,y}\,x_{i-1,y} & x_{i-1,y}\,x_{i,x} & x_{i-1,y}\,x_{i,y}\\ x_{i,x}\,x_{i-1,x} & x_{i,x}\,x_{i-1,y} & x_{i,x}\,x_{i,x} & x_{i,x}\,x_{i,y}\\ x_{i,y}\,x_{i-1,x} & x_{i,y}\,x_{i-1,y} & x_{i,y}x_{i,x} & x_{i,y}\,x_{i,y}\end{bmatrix}\end{matrix} \tag{123}$$

$$\begin{matrix}[x_{i-1}\ x_i]\underbrace{\begin{bmatrix}-G_i^T\\ 1\end{bmatrix}R_i^{-1}\begin{bmatrix}\tilde{x}_i-G_i\,\mu_{x_{i-1}}\end{bmatrix}}\\ \begin{bmatrix}x_{i-1,x}\\ x_{i-1,y}\\ x_{i,x}\\ x_{i,y}\end{bmatrix}\end{matrix} \tag{124}$$

A measurement event could be added by following this procedure with equations (62) and (63).

$$\begin{matrix}y_i^T\underbrace{H_i^{j\,T}Q_i^{-1}H_i^j}\,y_i\\ \begin{bmatrix}x_{i,x}\,x_{i,x} & x_{i,x}\,x_{i,y} & x_{i,x}\,m_{j,x} & x_{i-1,x}\,m_{j,y}\\ x_{i,y}\,x_{i,x} & x_{i,y}\,x_{i,y} & x_{i,y}\,m_{j,x} & x_{i-1,y}\,m_{j,y}\\ m_{j,x}\,x_{i-1,x} & m_{j,x}\,x_{i-1,y} & m_{j,x}\,m_{j,x} & m_{i,x}\,m_{i,y}\\ m_{j,y}\,x_{i-1,x} & m_{j,y}\,x_{i-1,y} & m_{i,y}\,m_{i,x} & m_{i,y}\,m_{i,y}\end{bmatrix}\end{matrix} \tag{125}$$

$$\begin{matrix}y_i^T\underbrace{H_i^{j\,T}Q_i^{-1}\left(z_i^j-\hat{z}+H_i^j\,y_i\right)}\\ \begin{bmatrix}x_{i,x}\\ x_{i,y}\\ m_{j,x}\\ m_{j,y}\end{bmatrix}\end{matrix} \tag{126}$$

The initialization is given in (64).

$$\begin{matrix}x_0^T\underbrace{\Omega_0}\,x_0\\ \begin{bmatrix}x_{0,x}\,x_{0,x} & x_{0,x}\,x_{0,y}\\ x_{0,y}\,x_{0,y} & x_{0,y}\,x_{0,y}\end{bmatrix}\end{matrix} \tag{127}$$

An motion update or measurement update can achieved with the following equations, respectively.

$$\Omega_{new}=\Omega_{old}+F^T\Omega_{sub}F \tag{128}$$

$$\xi_{new}=\xi_{old}+F^T\xi_{sub} \tag{129}$$

With F for a motion update is:

$$F_{x,m_j} = \begin{bmatrix} \underbrace{0}_{\text{size of } x_{0:i-1}} & \underbrace{I}_{\text{size of } x_i} & \underbrace{0}_{\text{size of } x_{i+1}} & \underbrace{0}_{\text{size of } x_{x+2:I}} & \underbrace{0}_{\text{size of } m_{1:J}} \\ \underbrace{0}_{\text{size of } x_{0:i-1}} & \underbrace{0}_{\text{size of } x_i} & \underbrace{I}_{\text{size of } x_{i+1}} & \underbrace{0}_{\text{size of } x_{x+2:I}} & \underbrace{0}_{\text{size of } m_{1:J}} \end{bmatrix} \begin{matrix} \} \text{ size of x} \\ \} \text{ size of } m_j \end{matrix} \tag{130}$$

With F for a measurement update:

$$F_{x,j} = \begin{bmatrix} \underbrace{0}_{\text{size of } x_{0:i-1}} & \underbrace{I}_{\text{size of } x_i} & \underbrace{0}_{\text{size of } m_{1:j-1}} & \underbrace{0}_{\text{size of } m_j} & \underbrace{0}_{\text{size of } m_{j+1:J}} \\ \underbrace{0}_{\text{size of } x_{0:i-1}} & \underbrace{0}_{\text{size of } x_i} & \underbrace{0}_{\text{size of } m_{1:j-1}} & \underbrace{I}_{\text{size of } m_j} & \underbrace{0}_{\text{size of } m_{j+1:J}} \end{bmatrix} \begin{matrix} \} \text{ size of x} \\ \} \text{ size of } m_j \end{matrix} \tag{131}$$

With I the identity matrix and 0 a zero matrix.

Appendix C

Marginalization lemma and conditioning lemma in moments and information form

Table 18 expresses the dual relationship between moments and information form, with respect to marginalization and conditioning

Table 18 relationship between moments and information form [6]

$$p(\boldsymbol{\alpha},\boldsymbol{\beta}) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_\alpha \\ \boldsymbol{\mu}_\beta\end{bmatrix}, \begin{bmatrix}\Sigma_{\alpha\alpha} & \Sigma_{\alpha\beta} \\ \Sigma_{\beta\alpha} & \Sigma_{\beta\beta}\end{bmatrix}\right) = \mathcal{N}^{-1}\left(\begin{bmatrix}\boldsymbol{\eta}_\alpha \\ \boldsymbol{\eta}_\beta\end{bmatrix}, \begin{bmatrix}\Lambda_{\alpha\alpha} & \Lambda_{\alpha\beta} \\ \Lambda_{\beta\alpha} & \Lambda_{\beta\beta}\end{bmatrix}\right)$$

	MARGINALIZATION $p(\boldsymbol{\alpha}) = \int p(\boldsymbol{\alpha},\boldsymbol{\beta})\, d\boldsymbol{\beta}$	CONDITIONING $p(\boldsymbol{\alpha} \mid \boldsymbol{\beta}) = p(\boldsymbol{\alpha},\boldsymbol{\beta})/p(\boldsymbol{\beta})$
COV. FORM	$\boldsymbol{\mu} = \boldsymbol{\mu}_\alpha$ $\Sigma = \Sigma_{\alpha\alpha}$	$\boldsymbol{\mu}' = \boldsymbol{\mu}_\alpha + \Sigma_{\alpha\beta}\Sigma_{\beta\beta}^{-1}(\boldsymbol{\beta} - \boldsymbol{\mu}_\beta)$ $\Sigma' = \Sigma_{\alpha\alpha} - \Sigma_{\alpha\beta}\Sigma_{\beta\beta}^{-1}\Sigma_{\beta\alpha}$
INFO. FORM	$\boldsymbol{\eta} = \boldsymbol{\eta}_\alpha - \Lambda_{\alpha\beta}\Lambda_{\beta\beta}^{-1}\boldsymbol{\eta}_\beta$ $\Lambda = \Lambda_{\alpha\alpha} - \Lambda_{\alpha\beta}\Lambda_{\beta\beta}^{-1}\Lambda_{\beta\alpha}$	$\boldsymbol{\eta}' = \boldsymbol{\eta}_\alpha - \Lambda_{\alpha\beta}\boldsymbol{\beta}$ $\Lambda' = \Lambda_{\alpha\alpha}$

Bibliography

[1] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual SLAM algorithms: A survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*. 2017, doi: 10.1186/s41074-017-0027-2.

[2] H. Zamzuri, M. A. Bin, A. Rahman, S. A. Mazlan, A. Hadi, and A. Rahman, "Computational cost analysis of extended Kalman filter in simultaneous localization and mapping (EKF-SLAM) problem for autonomous vehicle." Accessed: Apr. 13, 2021. [Online]. Available: https://www.researchgate.net/publication/283824088.

[3] Wikipedia contributors, "Gaussian function --- {Wikipedia}{,} The Free Encyclopedia." 2020, [Online]. Available: https://en.wikipedia.org/w/index.php?title=Gaussian_function&oldid=994692675.

[4] S. Thrun, "Probabilistic robotics," *Commun. ACM*, 2002, doi: 10.1145/504729.504754.

[5] N. S. Shah, "D STEREOVISION FOR QUANTIFICATION OF SKIN DISEASES SUB-MILLIMETRE D RECONSTRUCTION N . (Nimish) Shah," 2020.

[6] R. Eustice, M. Walter, and J. Leonard, "Sparse extended information filters: Insights into sparsification," *2005 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS*, no. September 2005, pp. 3281–3288, 2005, doi: 10.1109/IROS.2005.1545053.

[7] E. Eade, P. Fong, and M. E. Munich, "Monocular graph SLAM with complexity reduction," *IEEE/RSJ 2010 Int. Conf. Intell. Robot. Syst. IROS 2010 - Conf. Proc.*, pp. 3017–3024, 2010, doi: 10.1109/IROS.2010.5649205.

[8] A. Dine, A. Elouardi, B. Vincke, and S. Bouaziz, "Graph-based simultaneous localization and mapping," *IEEE Robot. Autom. Mag.*, vol. 23, no. 4, pp. 160–173, Dec. 2016, doi: 10.1109/MRA.2016.2580466.

[9] S. Riisgaard and M. R. Blas, "SLAM for Dummies A Tutorial Approach to Simultaneous Localization and Mapping By the 'dummies.'" .

[10] F. van der Heijden, "implementation details ES-EKF-SLAM," no. January, 2021.

[11] F. Van Der Heijden, "Derivatives of Euler angles," 2021.

[12] F. van der Heijden, "Math for Computer Vision and Navigation," p. 29, 2019.

[13] A. Kasar, "Benchmarking and Comparing Popular Visual SLAM Algorithms," *arXiv*, Nov. 2018, Accessed: Apr. 25, 2021. [Online]. Available: http://arxiv.org/abs/1811.09895.

[14] MathWorks, "Measure the Performance of Your Code - MATLAB & Simulink - MathWorks Benelux." https://nl.mathworks.com/help/matlab/matlab_prog/measure-performance-of-your-program.html (accessed May 22, 2021).

[15] "how can I compute memory usage? - MATLAB Answers - MATLAB Central." https://nl.mathworks.com/matlabcentral/answers/384369-how-can-i-compute-memory-usage (accessed May 22, 2021).