B.Sc.Eng. Thesis
Bachelor of Science in Engineering

DTU Compute
Department of Applied Mathematics and Computer Science

SLAM for Eye Wear

Adapting Monocular Visual SLAM for The Tobii Pro Glasses 2

Frederik Warburg (s153847)

Kongens Lyngby 2018

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

This thesis considers the Simultaneous Localization And Mapping (SLAM) problem, where the objective is to adapt and evaluate an open-source SLAM implementation to the Tobii Pro Glasses 2.
It is important for companies, which use these glasses to collect data on consumer behavior, to know the position and surroundings of the person wearing the glasses.
Based on a comparison of 34 open-source projects, it is considered that the most auspicious implementation is the `EKFmonoSLAM` project. It is examined how this implementation can be adapted to data from the glasses through camera calibration, downsizing of the images by of factor 5, and reduction of the threshold for FAST-9 to 20.
Subsequently, the accuracy of the localization estimate is investigated. It is found that the scaling is very dependent on the initial inverse Gaussian distribution describing the depth of the features and that `EKFmonoSLAM` is sensitive to fast rotations. Through simulated data and image data, it is examined how the depth distribution can be estimated.
Good results are obtained when the glasses are hand-held, but even with a good estimates of the depth distribution, `EKFmonoSLAM` does not perform well when the glasses are head-worn. In an indoor experiment, the relative cumulative error is estimated to 125 % over a 12 meter distance. On the other hand, significantly better results are obtained in an outdoor experiment where the relative cumulative error can be reduced to 13 % over a distance of 104 meters.
Based on this empirical investigation, which includes extensive data-gathering, the thesis concludes that even the apparently best open-source SLAM project does not give sufficient precision to provide useful information about consumer behavior in a indoor setting. Further research and development is necessary in order to solve the SLAM problem for the Tobii Pro Glasses 2.

Resumé

Dette bachelorprojekt omhandler problemet om simultan lokalisering og kortdannelse (SLAM), hvor formålet er at tilpasse og evaluere en open-source implementation af SLAM problemet for Tobii Pro Glasses 2.
For firmaer, der bruger brillerne til at opsamle data om kundeadfærd, er det vigtigt at kende positionen og omgivelserne for personen, som har brillerne på.
Baseret på en sammenligning af 34 open source projekter er `EKFmonoSLAM` implementeringen fundet mest lovende. Det gennemgås hvordan implementeringen kan tilpasses data fra brillerne igennem kamera kalibrering, nedskalering af billedstørrelsen med en faktor 5 og reducering af tærskelværdien i FAST-9 til 20.
Dernæst undersøges præcisionen af lokalitetsestimatet. Resultatet af undersøgelserne viser, at skaleringen er meget afhængig af den indledende inverse normalfordeling, der beskriver dybden af landmærkerne, og `EKFmonoSLAM` er følsom overfor hurtige rotationer. Ved hjælp af simuleringer og billeddata er det undersøgt, hvordan dybdefordelingen kan estimeres.
Gode resulter findes, når billerne er håndholdt, men selv med gode estimater af dybdefordelingen performer `EKFmonoSLAM` ikke godt, når briller sidder på hovedet. I et indendørs eksperiment fås den relative kumulative fejl til 125 % over en 12 meters afstand. På den anden side fås væsentligt bedre resultater i et udendørs eksperiment hvor den relative kumulative fejl reduceres til 13 % over en strækning på 104 meter.
Baseret på denne emperiske undersøgelse, som indeholder stor data indsamling, må konkluderes, at selv det tilsyneladende bedste open-source projekt ikke giver tilstrækkelig præcision til at give nyttig information om indedørs kundeadfærd. Videre forskning og udvikling er nødvendigt for at løse SLAM problemet for Tobii Pro Glasses 2.

Preface

This thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for acquiring a B.Sc. degree in Engineering (Mathematics and Technology).

I would like to thank my two supervisors for helping me out through the entire process of developing the final work: Henrik Aanæs for always providing excellent and fast assistance when I have been stuck at various places, and Jonathan Dyssel Stets for helping with the experimental work and for always being available for excellent counseling.
Finally, a great thanks to friends, family, and Charlotte Kaae for encouragement and support.

Kongens Lyngby, April 7, 2018

Frederik Warburg (s153847)

Contents

CHAPTER 1

Introduction

Imagine drones delivering groceries through your window or driverless cars bringing you to and from work. These are just two examples of autonomous mobile robots. These autonomous robots have potential to revolutionize our way of life, our production processes, transport systems, cities, and much more.

A technical challenge with autonomous robots is that they have to understand their environment. It is crucial that the autonomous robots move and act safely in environments and in situations of which they have no prior knowledge. They have to understand and navigate safely in every conceivable situation.

This is referred to as the Simultaneous Localization And Mapping (SLAM) problem. To solve the SLAM problem, the robot has to reconstruct a map of its environment while maintaining information of its position within this map. The robot cannot have any prior knowledge of the environment nor its starting position. Furthermore, the SLAM problem has to be solved in real time requiring fast and efficient algorithms.

It is, therefore, crucial to have a stable solution to the SLAM problem to achieve a future with autonomous robots.

1.1 Motivation

The Swedish Eye Tracking Company, Tobii Technologies, has developed the Tobii Pro Glasses 2. These high-tech glasses are equipped with a front camera and gaze vision such that the eye movement of the person wearing the glasses can be tracked. Currently, these glasses are primarily used to tailor marketing of consumer products. The glasses are mobile and thus experiments about consumer behavior can be tested in real world settings. For instance, it is tested which products attract more attention on the super market shelves. However, the glasses are not incorporated with a tracking system, thus valuable information about consumer navigation patterns are not tracked. By developing smart software and solving the SLAM problem, it would be possible to obtain more information about consumer behavior without adding extra sensors or increasing the cost of the glasses. This additional information could potentially provide better consumer analysis.

1.2 Overview

The objective of this thesis is to adapt and test how well an open source SLAM implementation works with data recorded from the Tobii Pro Glasses 2. The thesis consist of the following chapters which together represent the necessary step for this evaluation.

Background

Chapter 2 serves three purposes. Firstly, to give some background information about the hardware, the Tobii Pro Glasses 2. Secondly, to lay out the process of selecting an adequate SLAM implementation for the glasses. An extensive comparison of 34 open source project are presented for readers with an interest in continuing working on another SLAM implementation. And thirdly, to provide some general background information about the chosen Matlab implementation, `EKFmonoSLAM`[Civ+10], and the previous work and results obtained using this implementation.

Theory

Chapter 3 gives an introduction to the theoretical background of the chosen implementation, `EKFmonoSLAM`. Most importantly, this includes a thorough explanation of the Extendend Kalman Filter, which serves as the backbone of this implementation, a description of the inverse depth parametrization used to store and initialize features, 1-point RANSAC - a computational efficient robust estimation methods, and feature matching with Normalize Cross Correlation and Active Search. Readers who are familiar working with SLAM systems or computer vision in general can go lightly over Chapter 3, but note that the theory described in this section will serve as a backbone throughout the thesis.

Experimental Design

Chapter 4 outlines the experimental design and considerations. The experimental work is summarized and divided into a section on simulating data and a section on video recordings using the Tobii Pro Glasses 2. The data from these recordings could be useful for future studies.

Implementation

The purpose of Chapter 5 is to show how the results obtained by Civera et al. [JD06] can be replicated using image data from the Tobii Pro Glasses 2. First, an overview of the software, `EKFmonoSLAM`, is provided. Then, a detailed description and validation of the camera calibration for the Tobii Pro Glasses 2 is given. Next, it is described how an adequate downsizing of the recorded images is chosen, and subsequently it is described why and how it is decided to reduced the thresholds for the feature detector in order to obtain a sufficient number of features. Finally, as a form of implementation validation, it is showed how Civera et al.'s results and experiments can be replicated by hand held recordings from the front camera of Tobii Pro Glasses 2.

Results & Discussion

In Chapter 6, the results and findings of this thesis are presented and discussed. It is found that the `EKFmonoSLAM` is very dependent on the initial inverse depth distribution of features and it is sensitive towards fast rotations. It is investigated how a good prior inverse Gaussian distribution can be chosen and why the software is so sensitive towards fast rotations. Finally, it is found that the software performs significantly better outdoor where more distant features are found.

Future Work

In Chapter 7 it is discussed which improvements are necessary in the future work. It is emphasized how sensor fusion could improve the stability of the solution.

Conclusion

In Chapter 8 the findings and conclusion of the thesis are presented.

1.3 Prerequisites

The thesis is written for a reader with basic knowledge of computer vision and statistics, but, as was the case for myself when I began this study, with no particular insight in neither SLAM, Extendend Kalman Filters, nor experimental work with vision systems.

1.4 Notation

In this thesis matrices will be indicated by capital, bold letters $\mathbf{F}$, vectors will be indicated by a bold, italic letters $\boldsymbol{x}$, and scalars will be written with italic letters a.

CHAPTER 2

Background

The purpose of this chapter is to investigate which open-source SLAM project will be most adequate for the Tobii Pro Glasses 2. First, Section 2.1 provides some background information about the hardware of this project, the Tobii Pro Glasses 2. Based on this knowledge, Section 2.2 argues why the `EKFmonoSLAM` implementation is chosen, and in Section 2.3 the previous work and results with this implementation will be described.

2.1 Hardware

The purpose of this section is to provide sufficient information about the hardware to choose an adequate open-source software implementation.

Figure 2.1 shows an image of the head unit of the Tobii Pro Glasses 2. In these glasses several sensors are included.

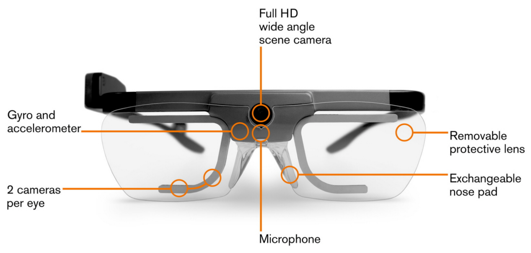


Figure 2.1: Head unit of the Tobii Pro Glasses 2 [Tob17].

The Tobii Pro Glasses 2 has a wide-angle lens located at the centre, between the eyes. A wide-angle lens has a shorter focal length and thus allows more of the scene to be included in the image. According to [Dav+04] a wide-angle lens in single-camera SLAM "demonstrate significantly improved SLAM results, with increased movement range, accuracy and ability to track agile motion". Since the camera catches more of the scene, landmarks will remains visible throughout larger camera movement. Thus, the landmark density can be decreased and camera movement can be increased.

Also an IMU (a gyroscope and an accelerometer) is mounted between the eyes of the glasses. The gyroscope measures the angular speed of the glasses in the 3 spatial dimensions and the accelerometer measures the acceleration in the 3 spatial dimensions. These sensors can be used to collect additional data.

Based on the front camera of the Tobii Pro Glasses 2, it is decided to chose an open-source SLAM project that solves the SLAM problem using only one camera. Furthermore, it is wished that the chosen implementation can be extended to support both data from the gyroscope and the accelerometer as these additional data source could potentially provide a more stable solution. In the next section an open source project will be chosen based on these requirements.

2.2 Software Selection

Many pioneer SLAM researchers have provided source code for their projects. Thus, based on the time limitation of the project and the many available open source implementations, it is assessed that it will provide a better result to adapt and modify an already implemented SLAM project rather than developing and implementing a solution from scratch. This section will describe the process of choosing an appropriate SLAM implementation and argue for the choice of software.

2.2.1 OpenSLAM.org

OpenSLAM.org is an online platform where SLAM researchers can share their SLAM implementations. ”The goal of OpenSLAM.org is to provide a platform for SLAM researchers which gives them the possibility to publish their algorithms.”[Bac17] As of 2017, researchers have provided 34 different solutions of the SLAM problem to OpenSLAM.org. According to OpenSLAM.org ”Published algorithm should have a certain degree of robustness.”[Bac17] This statement provides a quality stamp for all the projects. Furthermore, it is required that the source code of the projects are provided for users to use and modify.

It is possible to find open source SLAM implementations elsewhere (e.g on GIT), however it is chosen only to investigate the SLAM implementation from OpenSLAM.org because the variety and quality of the projects on this site are considered to be sufficient for the purpose of this thesis.

2.2.2 Selection of software

By way of introduction, all the open source projects provided via OpenSLAM.org are listed in table A.1 (Appendix). This table includes key stats about each implementation such as: Which core algorithm has been used, if the implementation supports 3D modelling, what languages it is developed in, etc. The table also provide a short description of each project. Table A.1 gives an overview of the available SLAM projects at OpenSLAM.org. Only readers who wish to work with their own open source SLAM implementation need to visit this table.

Most of the projects at OpenSLAM.org use a version of an Extended Kalman Filter (EKF) or Graph Optimization. It has been assessed that it will be more straightforward to use an EKF algorithm when using multiple sensor sources (a camera and an IMU) because the extra data and input parameters can rather easily be included in the model without changing the model structure. Therefore, it is decided to focus on the implementations that use an EKF as the main algorithm.
The majority of the projects at OpenSLAM.org are implemented in C++ or Matlab. It is preferred that the chosen SLAM project is implemented in Matlab over C++

because Matlab is a high level language that makes it easier and faster to modify projects. Furthermore, Matlab is good at handling images, it has fast vector calculations, and has several image toolboxes that make many image operations easy (E.g the camera calibration toolbox).

Thus, it is chosen only to look at the implementations that are written in Matlab and uses an Extended Kalman Filter as the main algorithm. In table 2.1 the five OpenSLAM.org projects that comply with the algorithm and software requirements are listed.

Table 2.1: The table shows the open source implementations from OpenSLAM that use the EKF and are implemented in Matlab. The content of the table originates from [Bac17]. In this thesis, the `EKFMonoSLAM` implementation has been chosen as the foundation for a SLAM solution for the Tobii Pro Glasses 2. The `EKFMonoSLAM` has been chosen because it support 3D modelling, it is the newest of the implementations, and it is written by the well-acknowledged SLAM researches: Javier Civera and J.M. M. Montiel.

	Hardware and software requirements	Algorithm	Author	Input Data	Type of map	Release Year	Description	3D support
EKFMonoSLAM	Matlab	EKF	Javier Civera and J. M. M. Montiel	Monocular image sequence and its camera calibration	A sparse 3D map of salient point features	2010	EKFmonocularSLAM contains Matlab code for EKF SLAM from a 6 DOF motion monocular image sequence. The algorithm takes as input a monocular image sequence and its camera calibration and outputs the estimated camera motion and a sparse map of salient point features. The code includes state-of-the-art contributions to EKF SLAM from a monocular camera: inverse depth parametrization for 3D points and efficient 1-point RANSAC for spurious rejection.	Yes
CAS-Toolbox	Matlab	EKF	Kai O. Arras	Sensor and odometry data	Feature maps	2007	This software is a GNU GPL licenced Matlab toolbox for robot localization and mapping. It is made for research and education and independent on the type(s) of feature and type(s) of sensors/ It can import a number of data file formats from any sensor. It allows you to plug in and out your feature extraction, odometry model, data association strategy, etc. and to plug in and out your SLAM or Localization approach. It furthermore comes with a number of useful tools and functions.	No
CEKF-SLAM	Matlab	Compressed Extended Kalman Filter	Haiqiang Zhang and Lihua Dou	Sensor and odometry data	Feature maps	2007	CEKF-SLAM was originally proposed by Jose Guivant and Eduardo Net. This algorithm reduces the computational complexity by dividing the system state vector into two parts: the active local state vector and the others. Only the local state vector is updated for each step by EKF, and the necessary information for updating the other states is compressed into some auxiliary cofficient matrices. When the local area changes, a full update was executed to get the same estimation results as EKF.	No
Pkg. of T.Bailey	Matlab	EFK, UKF, FastSLAM1, and FastSLAM2.	Tim Bailey	Sensor and odometry data	Feature maps	2007	This package is a collection of implemented SLAM approaches by Tim Bailey. The code is written in MatLab and performs EFK, UKF, FastSLAM 1, and FastSLAM2.	Unknown

Table 2.1 shows that `EKFMonoSLAM` is the only one of these projects that supports 3D models. The `EKFMonoSLAM` is implemented by Javier Civera and J. M. M. Montiel. The fact that `EKFMonoSLAM` is implemented by these two acknowledged SLAM researchers advocates for a well written, robust, and efficient implementation. Furthermore, `EKFMonoSLAM` is the newest of the projects listed in the table.

The `EKFMonoSLAM` project is chosen as the most adequate SLAM solution to the Tobii Pro Glasses 2. It is chosen based on the EKF algorithm, the Matlab implementation, the 3D modelling abilities, and the well acknowledged authors. The next section will outline some of the most contributing ideas of this project.

2.3 Related work

This section will highlight and summarize the results and findings from four significant papers related to the selected software, `EKFmonoSLAM`. The authors of the software, Civera and Montiel, have contributed to the three last mentioned articles and the main ideas described in these articles are all a central part of the software implementation. It is important to be aware of their findings, failures, and results in order to use and understand the software properly. This section will give an overview of the results and main ideas for the four articles, while the theory will be explained in chapter 3.

Cameras are a popular sensor to use to solve the SLAM problem because of the high information level in images and the low cost of cameras. As early as in 2003 it was demonstrated by Davison [Dav03] that it is possible to solve the SLAM problem using just one camera, and thus the SLAM problem can potentially be solved by the front camera of the Toii Pro Glasses 2. Since Davison's discovery several researchers have contributed to the development of more sophisticated algorithms.

Civera et al. suggest in 2006 [JD06] (and in a more refined article in 2008 [JM08]) to parametrize point features using inverse depth coordinates instead of Euclidean coordinates. Through experimental work it is verified that this alternative parametrization have several advantages. It enables an unified representation of very distant and close points, it enables an uncertainty estimate of distant points, and it allows undelayed initialization. By using a linearity index, Civera et. al. are able to deal with low and high parallax features simultaneously without a binary decision tool. The only drawback of this inverse depth parametrization is that it requires six dimensions to represent a feature as opposed to three when using Euclidean coordinates. It is found that these additional dimensions have a 4-fold increase on the computational cost. However, if the number of features is kept under 100, it is still possible to obtain real-time performance.

In 2010 [Civ+10], Civera et al. showed that the dimensions of the robust estimation method, RANSAC, can be reduced to just one point by incorporating prior information from the Extend Kalman Filter framework. Reducing the dimensions to just one point as opposed to five points, which according to [Nis04] is the lowest number of points required to estimate the motion between two frames of a 6 degree of freedom camera, has huge computational savings. Through experimental work, Civera et. al. find that the overhead of 1-point RANSAC is less than 10 % which makes it suitable for real time implementations. Furthermore, Civera et. al. showed that the the errors of the trajectory path of large outdoor experiments was around 1 % when parred with wheel odemetry.

This section has presented the three most important and contributing ideas behind the `EKFmonoSLAM` implementation, namely single camera SLAM, the inverse depth parametrization, and the 1-point RANSAC method. Furthermore, it was outlined that Civera et. al. have obtained good results and real-time performance using the chosen `EKFmonoSLAM` Matlab implementation.

Chapter 2 has provided some background information about the Tobii Pro Glasses 2. It has been chosen to adapt and modify the `EKFmonoSLAM` solution to the SLAM problem. The three ideas most contributing for this software implementation, single camera SLAM, the inverse depth parametrization, and the 1-Point RANSAC, have been outlined. The next chapter will give a more thorough and theoretical explanation of these important ideas.

CHAPTER 3

Theory

This chapter explains the theory behind the EKFmonoSLAM implementation. First, some important theory about camera geometry is presented in Section 3.1 as these insights will be a central part of the understanding of the thesis and implementation. Then, in Section 3.2, a thorough explanation of the inverse depth parametrization of the landmarks is given. Following, the usage of quaternions is justified in Section 3.3. After establishing these concepts, the Extendend Kalman Filter is explained in three parts in Sections 3.4, 3.5, and 3.6. First, the basic idea behind the state space representation is explained and it is described how the inverse depth parametrization is used in this setting. This leads to an explanation of the two steps of the Extended Kalman Filter: Predict and update. Finally, a thorough description of the measurement operator used in EKFmonoSLAM is provided.
The chapter ends with an explanation of the theory behind the feature detection method, Features from Accelerated Segment Test (FAST) in Section 3.7, the matching method, Normalized Cross Correlation with Active Search in Section 3.8, and the robust estimation method, 1-Point RANSAC in Section 3.9.
As mentioned previously, readers familiar with the above described computer vision concepts can go lightly over this chapter.

3.1 Camera Geometry

The purpose of this section is to define the mathematical notation of two important models: The pinhole model and the homography. The pinhole model is fundamental to understand when working with cameras as this model explains how points are projected from 3D world coordinates into the 2D image plane. This model, and the parameters within it, will be referred to throughout the thesis. The notation of the homography is also outlined as this special case of the pinhole model is used in the camera calibration (Section 5.2). For a more detailed explanation of these models, the reader is referred to [Aan15].

A homogeneous 3D point $\boldsymbol{Q}_j$ in world coordinates is projected into the 2D image planes as a point via the pinhole model [Aan15].

$$\boldsymbol{q}_i = \mathbf{A}[\mathbf{R} \quad \boldsymbol{t}]\boldsymbol{Q}_j = \mathbf{P}\boldsymbol{Q}_j \tag{3.1}$$

where $\boldsymbol{q}_i$ is a 2D homogeneous point in the image plane, $\boldsymbol{Q}_j$ is the corresponding 3D homogeneous coordinate, $\mathbf{A}$ is the internal camera parameters, $\mathbf{R}$ is the rotation, $\boldsymbol{t}$ is the translation, and $\mathbf{P}$ is the camera matrix which is defined by $\mathbf{P} = \mathbf{A}[\mathbf{R} \quad \boldsymbol{t}]$.

The internal camera matrix $\mathbf{A}$ can be written as

$$\mathbf{A} = \mathbf{A}_p\mathbf{A}_q = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & \beta & u_0 \\ 0 & \alpha & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f & \beta & u_0 \\ 0 & f\alpha & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

where f is the focal length, α and β describe the affine image deformation (these are set to respectively 1 and 0 [Aan15] [JM08]), and u_0 and v_0 are the principal point offsets.

Furthermore, wide-angle lenses, as the front camera in the Tobii Pro Glasses 2, are often distorted. This distortion can be described by the distortion parameters κ_1 and κ_2 whose values results in the effects shown in Figure 3.1.

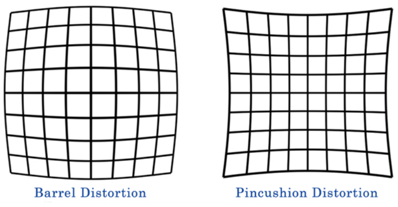


Figure 3.1: The figure shows the barrel and pincushion distortion of an square [Cyr17].

Note that when using the pinhole model 3.1 at least two images of a point are necessary to find the 3D position of the point [Aan15]. However, it is possible to solve equation 3.1 using only one image by adding the extra constraint that all points in the image lie in the same plane. This extra constraint reduces the 3D point $\boldsymbol{Q}_j$ to a 2D point $\boldsymbol{q}_j$. This extra constraint makes it possible to estimate the position of $\boldsymbol{q}_j$ from a single image of this point. This is referred to as a homography [Cyr17].

$$\boldsymbol{q}_i = \mathbf{H}\boldsymbol{q}_j = \mathbf{P}\begin{bmatrix} A & B & C \\ 0 & 0 & 1 \end{bmatrix}\boldsymbol{q}_j \tag{3.2}$$

where $\boldsymbol{q}_j$ is the observed 2D point, $\boldsymbol{q}_i$ is the corresponding 2D point projected onto the image plane, $\mathbf{P}$ is the camera matrix, and $\mathbf{H}$ is the homography that forces the

2D point $\boldsymbol{q}_j$ to be in the plane spanned by the vector $[A, B, C]$.

This point-to-point relationship and the decreased complexity compared to 3.1 makes the homography very desirable in camera calibration. Note that the homography 3.2 has to be solved as a minimization problem because of measurement errors. This is done by minimizing the difference between the $\boldsymbol{q}_j$ points mapped into the image plane by $\mathbf{H}$ and the points in the image plane $\boldsymbol{q}_i$.

$$\min_{\mathbf{H}} \sum_i ||\Pi(\boldsymbol{q}_i) - \Pi(\mathbf{H}\boldsymbol{q}_j)||_2^2 \tag{3.3}$$

where Π is the operator that maps homogeneous coordinates to an inhomogeneous coordinates. This can be expressed in 2D like $\Pi([x, y, s]^T) = [\frac{x}{s}, \frac{y}{s}]^T = [x, y]^T$.

This section has outlined the notation for both the pinhole model and the homography. The theory on the pinhole model will be referred to throughout the thesis, while the theory on the homography will primarily be used in the section on camera calibration (Section 5.2).

3.2 Inverse Depth Parametrization

This section will describe the inverse depth parametrization of landmarks presented by Civera et al. [JM08]. The desirable properties with the inverse depth parametrization is that it allows representation of points at infinity, it permit an unified handling and representation of close and distant features, it enables an uncertainty estimate of distant points, and it allows undelayed initialization of the Extended Kalman Filter (EKF).

The first real-time monocular camera SLAM solution by Davison [Dav03] used Euclidean coordinates to represent observed features. The Euclidean representation of features raised a problem for features at low parallax[1] because the EKF expected the estimated position uncertainty to be Gaussian distributed. However, the uncertainty estimate was inverse Gaussian distributed (Figures B.2 and B.3 in appendix). The flat tail of this distribution made it very difficult to determine whether a low parallax feature had a depth of 10, 100, or 1000 units. Therefore, Davison's original solution did not use distant features. This is a huge disadvantage because these low parallax features are very important to determine the rotation of the camera [JM08].

The EKF assumes that the observed landmarks have Gaussian distributed errors, which unfortunately is not the case when points are observed at low parallax. One approach to deal with these non Gaussian distributed errors suggests a delayed initialization of features observed at low parallax[BS05]. The features at low parallax are dealt with separately of the main map. Information about these features is accumulated over several frames to reduce uncertainty before inserting them into the main map. The drawback of this approach is that the features do not contribute to the camera localization until they are inserted into the main map. Furthermore, ”features at low parallax are often rejected”[JM08]. These low parallax features bears little information about the translation of the camera, but are very important to estimate the rotation of the camera. Thus, plenty of valuable information gets lost.

The inverse depth parametrization proposed by Civera et al. [JM08] cope with the issues of non Gaussian distributed position estimates. It enables undelayed initialization while coping with features at any depth. An important drawback of inverse depth parametrization is an extra computational cost. The inverse depth parametrization uses six coordinates to represent a point instead of three as in Euclidean coordinates. This means that a 3D point $\boldsymbol{Q}_i$ is represented in inverse depth coordinates as

$$\boldsymbol{Q}_i = [x_i, y_i, z_i, \theta_i, \phi_i, p_i]^T \tag{3.4}$$

where $[x_i, y_i, z_i]^T$ is the initial position of feature i described in the Euclidean coordinates. $[\theta_i, \phi_i, p_i]^t$ describe the current position of feature i relative to its initial

[1]The depth of the feature is high compared to the translation between the camera position at frame 1 and frame 2.

position in spherical coordinates. θ_i and ϕ_i is the azimuth and elevation and p_i is the depth. Figure B.4 in appendix describes the relation between spherical and Euclidean coordinates.

The relationship between Euclidean coordinates and the inverse depth parametrization can be described by:

$$\boldsymbol{Q}_i = \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} + \frac{1}{p_i}\boldsymbol{m}(\theta_i, \phi_i) \tag{3.5}$$

where $\boldsymbol{m}(\theta_i, \phi_i) = [\cos\phi_i \sin\theta_i, -\sin\phi_i, \cos\phi_i \cos\theta_i]^T$. Thus $\boldsymbol{m}(\theta, \phi)$ determines the direction from the initial location $[x_i, y_i, z_i]^T$ to the observed location $[X_i, Y_i, Z_i]$. The inverse depth $\frac{1}{p_i}$ determines the distance between the two points along this direction. It is important to note that the inverse depth is given by $d = \frac{1}{p}$ so when $p \to \infty$ the inverse depth $\frac{1}{p} \to 0$. This is very useful because computers are much better handling 0 than ∞.

The relationship in equation 3.5 is showed in Figure 3.2. This figure shows how a point ● is represented with the inverse depth parametrization.

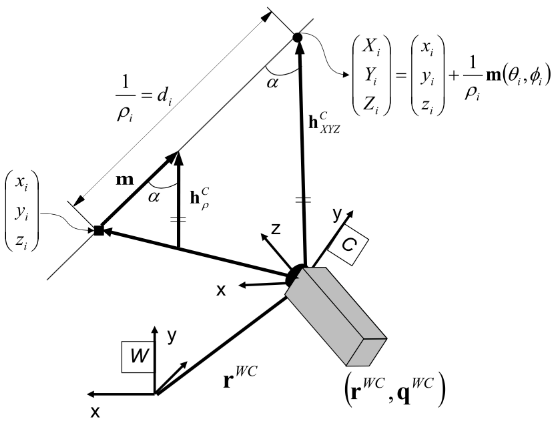

$\begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} + \frac{1}{\rho_i}\mathbf{m}(\theta_i, \phi_i)$
$\frac{1}{\rho_i} = d_i$
α
$\mathbf{h}^C_{XYZ}$
$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$
$\mathbf{m}$
α
$\mathbf{h}^C_\rho$
z
y
C
x
y
W
x
$\mathbf{r}^{WC}$
$(\mathbf{r}^{WC}, \mathbf{q}^{WC})$

Figure 3.2: The figure describes how a point ● is described with inverse depth parametrization. The point is first observed at ■ represented by $[x_i, y_i, z_i]^T$. The point's position is represented relative to this initial position by the inverse depth parametrization (Equation 3.5).

As mentioned, the inverse depth representation makes the position estimate of features at low parallax Gaussian distributed. Figure 3.3 shows the probability distributions of features represented with Euclidean coordinates (a) and inverse depth representation (b).

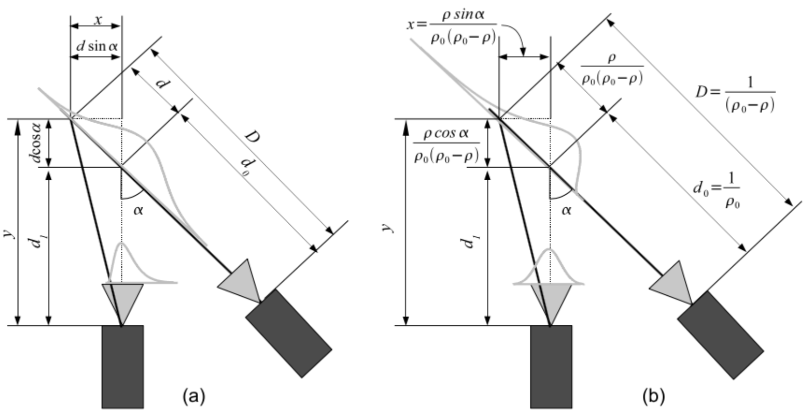

x
$d \sin\alpha$
d
D
d_0
$d\cos\alpha$
y
d_1
α
(a)
$x = \frac{\rho \sin\alpha}{\rho_0(\rho_0-\rho)}$
$\frac{\rho}{\rho_0(\rho_0-\rho)}$
$D = \frac{1}{(\rho_0-\rho)}$
$\frac{\rho\cos\alpha}{\rho_0(\rho_0-\rho)}$
$d_0 = \frac{1}{\rho_0}$
y
d_1
α
(b)

Figure 3.3: The figure shows the probability distributions of a point observed at one camera projected into the image plane of another camera. Subfigure (a) shows the probability distribution when the point is represented with Euclidean coordinates and subfigure (b) shows the probability distribution when the point is represented by inverse depth coordinates. Note that the distribution of the projected point in subfigure (b) is Gaussian distributed [JM08].

In figure 3.3 (a) a point is observed by the camera to the right. According to epipolar geometry, the depth to this point is unknown. When the same point is observed by the camera to the left, the point is known to lie on the epipolar line and this constraint can be used to estimate the depth of the point. However, because of the assumption of Gaussian distributed measurement noise the position estimate also becomes Gaussian distributed. Thus, when constructing the epipolar line by projecting this Gaussian distribution onto the image plane of the left camera the depth estimate becomes inverse Gaussian distribution. This is an issue for low parallax features when initializing the EKF.
The same setup is showed in (b), however here the inverse depth parametrization is used. Again, the measurement noise is assumed Gaussian, but because of the inverse depth parametrization the estimate becomes inverse Gaussian distributed. When projecting this inverse Gaussian distribution onto the image plane, it becomes Gaussian distributed [JM08]. This means that the point can be correctly initialized in the EKF.

The inverse depth parametrization relative to the camera location from which they were first observed has the advantage of Gaussian distributed position estimates. This allows for undelayed initialization of the EKF. Furthermore, the inverse depth copes better with points at infinity because these are represented as zeros.

The drawback is the extra computational cost of using six parameters. The `EKFmonoSLAM` solution reduces this extra cost by converting high parallax features into Euclidean coordinates. According to [JM08] ”there is no penalization in accuracy” for this conversion.

Thus, the inverse depth parametrization has several advantages: it enables representation of points at infinity, and it allows undelayed and unified initialization of the EKF for both close and very distant point.
Another alternative number representation used by Civera et al. is quaternions. The quaternions are advantageously used to represent rotations in `EKFmonoSLAM` and will be explained in the next section.

3.3 Quaternions

Throughout the `EKFmonoSLAM` implementation, Civera et al. use quaternions to handle rotations. The objective of this section is to explain the concept of quaternions and argue why quaternions are a better choice for handling rotation than the well known matrix implementation.

In `EKFmonoSLAM`, quaternion are used to define the camera orientation in the state vector. The intuition behind a quaternion is that it represents the rotation as an angle around an arbitrary rotation axis. More formally, a quaternion consist of four numbers: A number q_0 that describes the scaling, a number that describe the angle that the vector should be rotated, and two numbers that give the plane in which the vector should be rotated [EL98]. This can be expressed as

$$q_0 + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k}$$

where q_0, q_x, q_y, q_z are real numbers and $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$ are the quaternion units. For short quaternions are often represented as $[q_0, \mathbf{v}]$ where $q_0 \in \mathbb{R}$ and $\mathbf{v} = (q_x , q_y , q_z) \in \mathbb{R}^3$.

Dam et al. [EL98] highlight several disadvantages with the well known matrix representation. These are shortly summarized.

1. **Lack of intuition:** It is rather tedious to rotate about an arbitrary axis using homogeneous rotation matrices.
2. **The order matters:** When using two rotations, the order of which these are applied have importance for the result.
3. **Gimbal lock:** It is possible to make series of rotations where one degree of freedom is lost from the general rotation matrix. This results in a degenerated rotation matrix that can only rotate about two axis.
4. **Ambiguous correspondence to rotations:** It is difficult to solve the inverse problem and in general there is no unambiguous solution to the inverse problem.
5. **Representation is redundant:** The general homogeneous rotation matrix uses 16 places for the necessary four dimensions. Furthermore, there are 6 zeros in this matrix.

The primary advantages with the matrix representation is that translation, projection, and other basis transformations can be represented in the same homogeneous matrix.

This leads to the main disadvantage with quaternions. Quaternions can in practice only represent rotations. According to [EL98], it is possible to represent translation, but not as elegant as with the matrix representation. Quaternions are therefore in practical circumstances only used to represent rotations.

However, Dam et al. point out several advantages with quaternions. These are listed in short.

1. **Obvious geometrical interpretation:** Quaternions represent rotation as an angle around a rotation axis, which gives a more intuitive and elegant interpretation of rotation than the matrix representation.

2. **Unambiguous mapping:** The mapping between quaternions and Euler angles is unambiguous, except every angle can be represented by two quaternions. The reason is that angles comes in pairs and the two quaternion representations depend on which way you rotate about a given axis.

3. **Coordinate system independence:** The choice of coordinate system does not influence the rotation from the quaternions.

4. **Compact representation:** Quaternions use only four dimensions to represent the rotation.

5. **No Gimbal lock** The Gimbal lock problem does not appear with quaternions.

6. **Intuitive composition:** It is straight forward to apply rotation $\boldsymbol{q}_1$ followed by rotation $\boldsymbol{q}_2$. This is done just by rotation with the product of these $\boldsymbol{q}_2\boldsymbol{q}_1$.

As outlined the advantages of using the quaternions outweigh the disadvantages compared to using the well known matrix representation. Dam et. al. [EL98] believe that it is primary based on historical reasons that the matrix representation is still used. The main disadvantages with using the matrix representation is that it is tedious and troublesome that rotation must be expressed as rotation about three explicit axes, where the order is important. Furthermore, Gimbal lock can be encountered. These disadvantages are handled by using the quaternions which gives a more intuitive representation of angles and avoid Gimbal lock. Also, quaternions give a more compact and efficient representation of angles. Based on these findings Dam et. al. conclude that quaternions are superior to the well known matrix representation for representing rotation.

In this section, the advantages and disadvantages of quaternions and the well known matrix are summarized. Hopefully, these arguments make it clear why quaternions are preferred in `EKFmonoSLAM` over the well known matrix representation.

Now that the concepts of the inverse depth parametrization and the quaternions have been established we can move on to the core of the `EKFmonoSLAM` implementation, namely the Extended Kalman Filter (EKF). In order to properly explain the EKF, it is necessary to establish the state space representation.

3.4 State Space Representation

A state space representation is a formulation where the state of an object, in this case the camera and the landmarks, is set up as a vector and seen as a function of previous state values and model parameters. This can be described by the transition equation

$$\boldsymbol{x}_{k+1} = \mathbf{F}\boldsymbol{x}_k \tag{3.6}$$

where $\boldsymbol{x}$ is the state vector and $\mathbf{F}$ is the transition matrix that make a one step prediction. Thus, $\mathbf{F}$ sends the state vector from step k to step $k+1$.

When modelling the SLAM problem, it is assumed that the glasses are moving, while the landmarks are stationary. Thus, it is wished to model the glasses and the landmarks differently. First, the modelling of the glasses is described and then the modelling of landmarks is described.

3.4.1 Transition Equation of the Camera

To model the state of the Tobii Pro Glasses 2, the state vector is composed of the position $\boldsymbol{g}$, the orientation $\boldsymbol{q}$, the linear velocity $\boldsymbol{v}$, and angular velocity $\boldsymbol{\omega}$ of the glasses.

$$\boldsymbol{x}_g = [\boldsymbol{g}, \boldsymbol{q}, \boldsymbol{v}, \boldsymbol{\omega}]^T = [g_x, g_y, g_z, q_o, q_x, q_y, q_z, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]^T \tag{3.7}$$

where $\boldsymbol{x}_g$ is the state vector for the glasses. Note that the orientation is represented with the quaternion $\boldsymbol{q}$.

By using this state vector (Equation 3.7), the transition equation to model the glasses' movement with constant linear and angular velocity can be describes as

$$\boldsymbol{x}_g^{k+1} = \begin{bmatrix} \boldsymbol{g}_{k+1} \\ \boldsymbol{q}_{k+1} \\ \boldsymbol{v}_{k+1} \\ \boldsymbol{\omega}_{k+1} \end{bmatrix} = \mathbf{F}_1 \cdot \begin{bmatrix} \boldsymbol{g}_k \\ \boldsymbol{q}_k \\ \boldsymbol{v}_k \\ \boldsymbol{\omega}_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{g}_k + \boldsymbol{v}_k \Delta t \\ \boldsymbol{q}_k \times q(\boldsymbol{\omega}_k \Delta t) \\ \boldsymbol{v}_k \\ \boldsymbol{\omega}_k \end{bmatrix} \tag{3.8}$$

where $\boldsymbol{q}$ is the quaternion of $\boldsymbol{\omega}_k \Delta t$. This transition equation is less general than described in [JM08], because it assumes constant linear and angular velocities. Thus, $\boldsymbol{v}$ and $\boldsymbol{\omega}$ do not vary over time. It is seen that this transition equation is non linear. The reader, interested in the linearization of the transition equation, is referred to Appendix C.

3.4.2 Transition Equation of the Landmarks

It is assumed that the landmarks are stationary, thus the linear and angular velocity are zero. Furthermore, the landmarks are modelled as point features, so they do not have an orientation. Therefore, the state vector describing a landmark can be expressed in Euclidean coordinates as

$$\boldsymbol{x}_{l,E} = [x, y, z]^T \tag{3.9}$$

and in inverse depth coordinates as

$$\boldsymbol{x}_{l,ID} = [x, y, z, \theta, \phi, \rho]^T \tag{3.10}$$

where $\boldsymbol{x}_{l,E}$ and $\boldsymbol{x}_{l,ID}$ is the state vector describing a landmark in respectively Euclidean and inverse depth coordinates.

Because of the assumption of stationarity, the transition equation for the landmarks in Euclidean coordinates becomes

$$\boldsymbol{x}_{l,E}^{k+1} = \begin{bmatrix} x^{k+1} \\ y^{k+1} \\ z^{k+1} \end{bmatrix} = \mathbf{F}_2 \cdot \begin{bmatrix} x^k \\ y^k \\ z^k \end{bmatrix} = \begin{bmatrix} x^k \\ y^k \\ z^k \end{bmatrix} \tag{3.11}$$

where $\mathbf{F}_2$ is a 3x3 identity matrix. And the transition equation in inverse depth coordinates becomes

$$\boldsymbol{x}_{l,ID}^{k+1} = \begin{bmatrix} x^{k+1} \\ y^{k+1} \\ z^{k+1} \\ \theta^{k+1} \\ \phi^{k+1} \\ \rho^{k+1} \end{bmatrix} = \mathbf{F}_3 \cdot \begin{bmatrix} x^k \\ y^k \\ z^k \\ \theta^k \\ \phi^k \\ \rho^k \end{bmatrix} = \begin{bmatrix} x^k \\ y^k \\ z^k \\ \theta^k \\ \phi^k \\ \rho^k \end{bmatrix} \tag{3.12}$$

where $\mathbf{F}_3$ is a 6x6 identity matrix. The superscript indicate the time step, and the subscript indicate the spacial dimension. This transition equation models the position of a landmark as stationary.

To solve the SLAM problem, we need to model the location of the glasses and the position of the landmarks simultaneous. Therefore, the system's state vector is composed of the state vector for the glasses $\boldsymbol{x}_g$ and for each observed landmark $\boldsymbol{x}_l$.

$$\boldsymbol{x} = [\boldsymbol{x}_g^T, \boldsymbol{x}_{l_1}^T, \boldsymbol{x}_{l_2}^T, \boldsymbol{x}_{l_3}^T, \quad \dots \quad \boldsymbol{x}_{l_n}^T]^T \tag{3.13}$$

where $\boldsymbol{x}$ is the system's state vector. Note that this vector change size depending on how many landmarks have been observed.

As the concept of the state space representation has been established the EKF can be explained.

3.5 The Extended Kalman Filter

The state space representation, described in the previous section, is the underlying model of the camera state and landmark position. Neither of this information is given to begin with, thus the parameters of the model will be recursively updated based on the observed landmarks as time progresses. Some noise is associated with the observed position of a landmark and as the camera position is dependent on these noisy observations the entire state space model will be noisy. The goal of the Extended Kalman Filter (EKF) is to reduce the Gaussian noise from the observed data by weighing the model against the observations. This section will describe the process of the EKF in more detail.

The EKF is one variation of the Kalman Filter, which applies for non linear state space representations [Mad07]. The state space representation described in the previous section is highly non linear because of the varying orientations of the camera. The EKF linearize the transition operator (shown in Appendix C) and the measurement operator by differentiating these with respect to the state vector. After linearizing the system, a Kalman Filter is applied to estimate the states of the system.

The EKF is is well suited to the SLAM problem and widely used by SLAM researches [Bac17]. It has a recursive structure that allows real-time execution without requiring lots of memory. This recursive structure allows the EKF to correct itself and take advantage of new observations.

This section will describe in more detail how the EKF works and how it is applied to SLAM. The recursive structure of the EKF is explained by two steps: Predict and update.

3.5.1 Predict

In the prediction stage, the EKF uses the current state of the system to predict the next state. Thus, the generic transition equation 3.12 becomes

$$\boldsymbol{x}_{k+1} = \mathbf{F}\boldsymbol{x}_k + \boldsymbol{\Gamma}\boldsymbol{\epsilon}_k \tag{3.14}$$

where $\boldsymbol{x}_{k+1}$ is the predicted state vector. $\boldsymbol{\Gamma}\boldsymbol{\epsilon_k}$ has been added compared to equation 3.12 with $\boldsymbol{\Gamma}$ being the error operator and $\boldsymbol{\epsilon}$ being Gaussian noise describing the model error.

The EKF also calculates the uncertainty measure of this prediction which can be expressed as

$$\mathbf{P}_{k+1} = \mathbf{F}\mathbf{P}_k\mathbf{F}^T + \boldsymbol{\Gamma}\mathbf{Q}_k\boldsymbol{\Gamma}^T \tag{3.15}$$

where $\mathbf{P}$ is the state covariance matrix and $\mathbf{Q}_k$ is the covariance matrix for the errors described by $\boldsymbol{\epsilon}_k$.

3.5.2 Update

The EKF is used to update a model by incorporating information about incoming observations. The incoming observations have a measured uncertainty and thus can be described as

$$\boldsymbol{z} = \boldsymbol{z}_o + \boldsymbol{r} \tag{3.16}$$

where $\boldsymbol{z}_o$ is the true position of the observed landmark, $\boldsymbol{r}$ is Gaussian distributed noise related to the measurement error, and $\boldsymbol{z}$ is the measured location of the landmark.

This observation is compared to the model's estimate of this landmark. This is done by first mapping the model state, the 3D position of the landmark, to the measurement space, the 2D image plane.

$$\boldsymbol{h} = \mathbf{H}\boldsymbol{x}_{k+1} \tag{3.17}$$

where $\mathbf{H}$ is the measurement operator that maps from the model space to the measurement space. The size of $\mathbf{H}$ is the number of sensor dimensions times the number of states. In this case, the sensor dimensions equals 2 (the image plane) and the number of state equals 13 state from the camera plus 3 for each landmark represented in Euclidan coordinates plus 6 for each landmark represented in inverse depth coordinates.

The information of the observed landmarks is weighted against the model estimate of the respective landmarks with the Kalman gain. The Kalman gain is essentially a vector that is computed as the observation uncertainty divided by the system uncertainty (the uncertainty of the model and the observation).

$$\boldsymbol{K} = \mathbf{P}_{k+1}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k+1}\mathbf{H}^T + \mathbf{R}_k)^{-1} \tag{3.18}$$

where $\mathbf{R}$ is the covariance of the uncertainty of $\boldsymbol{r}$.

The state vector can be updated by multiplying the Kalman gain to the difference between the measured and estimated location. This is known as the innovation.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{K}(\boldsymbol{z} - \boldsymbol{h}) \tag{3.19}$$

Since the information about the measurements have been incorporated in the uncertainty, the state covariance matrix also has to be updated.

$$\mathbf{P}_{k+1} = \mathbf{P}_{k+1} - \boldsymbol{K}\mathbf{H}\boldsymbol{K}^T \tag{3.20}$$

Thus, from this equation it is seen that the state uncertainty is lowered relative to the Kalman gain.

This section have described the recursive structure of the EKF and the transition equations for both the camera and the landmarks. The next section will explain the measurement operator $\mathbf{H}$ used in the `EKFmonoSLAM` implementation.

3.6 Measurement operator

This section will describe in more detail how the measurement operator from Equation 3.17 maps from the model space to the measurement space in this SLAM implementation.

In this implementation, landmarks are modelled in 3D world coordinates, but measurements are in the 2D image plane in camera coordinates. Thus, in order to compare these two, we need to identify the measurement operator that maps from model space to measurement space (Equation 3.17).

To go from the model space to the measurement space we first need to go from world coordinates to camera coordinates. If the landmark position is modelled in inverse depth this is done by

$$\boldsymbol{h}_{i,ID}^{C_k} = \mathbf{R}_W^{C_k}\left(\rho_i\left(\begin{array}{c} X_i \\ Y_i \\ Z_i \end{array} - \boldsymbol{g}_{C_k}^W\right) + \boldsymbol{m}(\theta_i^W, \phi_i^W)\right) \tag{3.21}$$

and if the landmark is modelled in Euclidean coordinates, it is done by

$$\boldsymbol{h}_{i,E}^{C_k} = \mathbf{R}_W^{C_k}(\boldsymbol{y}_{i,E}^W - \boldsymbol{g}_{C_k}^W) \tag{3.22}$$

where $\mathbf{R}_W^{C_k}$ represent the rotation matrix computed from the state quaternion $\boldsymbol{q}_{C_k}^W$. The features are translated into camera coordinates by subtracting with $\boldsymbol{g}_{C_k}^W$, and they can be projected onto the image plane by using the pinhole model.

$$\boldsymbol{h}_u = \begin{pmatrix} u_u \\ v_u \end{pmatrix} = \begin{pmatrix} u_0 - \frac{f\,h_x^C}{d_x\,h_z^C} \\ v_0 - \frac{f\,h_y^C}{d_y\,h_z^C} \end{pmatrix} \tag{3.23}$$

where f is the focal length, $(u_0, v_0)^T$ are the principal offset, and d_x, d_y which are the ratio between the image size and the chip size. The image plane is distorted, thus we need to convert the undistorted coordinates $[u_u\,,\,v_u]$ into distorted coordinates $[u_d, v_d]$ by using the two parameter model

$$\begin{pmatrix} u_u \\ v_u \end{pmatrix} = \begin{pmatrix} u_0 + (u_d - u_0)(1 + \kappa_1 r_d^2 + \kappa_2 r_d^4) \\ v_0 + (v_d - v_0)(1 + \kappa_1 r_d^2 + \kappa_2 r_d^4) \end{pmatrix} \tag{3.24}$$

where $r_d = \sqrt{(d_x(u_d - u_0))^2 + (d_y(v_d - v_0))^2}$ and κ_1, κ_2 are the two distortion parameters.

Thus, the measurement operator is most intuitively described in three steps: From world coordinates to camera coordinates, from 3D point to 2D point, and then from the undistorted image plane to distorted image plane. All three steps are non linear and have to be linearized in order to put the measurement operator on the form as in Equation 3.17.

To summarize: The core of the `EKFmonoSLAM` implementation, the Extended Kalman Filter, has been explained in two step: Predict and update. The state space representation and transition operator for both the camera and the landmarks have been explained. Also, the measurement operator that maps from model space to the measurement space has been explained. The remaining part of this chapter will deal with detecting intrinsic features and matching these features between frames.

3.7 Feature Detector

To obtain real time SLAM for the Tobii Pro Glasses 2, it is important with fast and efficient algorithms. The Features from Accelerated Segment Test (FAST) has been developed by Rosten et al. [RD05] with speed and efficiency in mind. According to [RD05] the FAST algorithm has a pixel rate of 179, whereas other feature detectors like Harris Corner detector and DoG have respectively a pixel rate of 7.90 and 5.10. Thus, FAST is considerably faster.

3.7.1 Features from Accelerated Segment Test (FAST)

The FAST algorithm works by selecting a pixel C with intensity I_C. A Bresenham' circle of radius 3, thus containing 16 pixels (numerated 1 to 16 in Figure 3.4) is drawn around the centre point C. The criteria of defining C as a feature is that the intensity of N contiguous pixels out of the 16 need to be either above $I_C + T$ or below $I_C - T$ where T is a specified threshold. To improve the speed of the algorithm, the intensity of pixel 1, 5, 9, 13 are first compared with the I_C. If less than two of these points satisfies the threshold criterion, then C is not a feature. Otherwise, if the threshold criterion is satisfied for two or more of these points, the threshold criterion is checked for all 16 points. This process is repeated for all points.

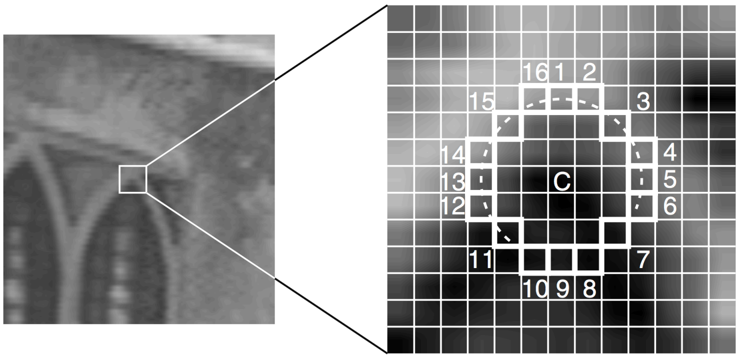


Figure 3.4: The figure to the right shows a zoom of an interest point with a Bresenham' circle with radius 3. The 16 pixels in the circle are numerated [RD05].

The feature detector applied in the `EKFmonoSLAM` implementation is FAST-9. Thus, 9 contiguous point out of the 16 in the Bresenham' circle has to be either above or below the threshold criterion. Alternatively, FAST-10, FAST-11, or FAST-12 could be applied. However, FAST-9 has been shown to give the best results compared to speed[Ros+08].

Many adjacent features are detected. These adjacent features bears nearly the same information about the image, thus we are only interested in the most significant feature in a small neighborhood. This can be obtained by applying a non maximum suppression to the found features.

The FAST algorithm does not compute a corner response function, thus the non maximum suppression cannot be used directly. However, a score function V can be calculated for potential feature points as the sum of the absolute difference between the pixel value of the center pixel I_C and the pixel values I_n of the N contiguous pixels [Vis17].

$$V = \sum_{n \in N} |I_n - I_C| \tag{3.25}$$

The score function of adjacent features are compared and the feature with highest score function within a mask of 3x3 pixel is chosen as the feature point [Ros+08]. Thus, removing the features that are directly next to each other.

This section has explained that the FAST-9 algorithm classifies a point as a feature if 9 contingous points of the 16 points in the Bresenham' circle satisfy the threshold criteria. The algorithm is very fast and therefore well suited in the SLAM framework. To obtain high repeatability Rosten et al. [RD05] suggest between 500 - 1500 features per frame.

3.8 Feature Matching

`EKFmonoSLAM` uses Normalized Cross Correlation (NCC) with Active Search to match features. This method is used because it is fast and rather intuitive. This section will describe the idea behind standard NCC and the NCC with Active Search suggested by [CD08].

3.8.1 Normalized Cross Correlation

Normalized Cross Correlation (NCC) is a widely used algorithm to match features. It is simple, it can be used in many different applications, and it is fast compared to other feature matching algorithms. The basic idea of NCC is to use the patches associated with each feature as descriptors, and then use the correlation between the pixel values in the patches as a measurement of how alike the two features are. The features where the descriptors have the highest correlation are matched [Aan15].
NCC have some drawbacks. If there is little variance in the image, the noise will be dominant. Thus, to achieve correct matches it is necessary that the patch have variance in two directions to avoid that the motion direction is ambiguous (the Aparture problem). Also, the correlation is sensitive to image rotations and changes in scale.

3.8.2 Active Search

When matching features between sequential images there is a physical constraint to how far two correctly matched features can lie from each other. E.g most often it is not necessary to search for a match in the upper left corner if a feature has been detected in the lower right corner.

Active Search exploit this prior knowledge and tries to search only in those parts of the image where true matches are most probable. By only searching in these areas it is possible to reduce the processing operations required to achieve global matching by a factor of 4 to 8 [CD08].

The Active Search method suggested by [CD08] use the uncertainty information in the covariance matrix from the EKF as a prior knowledge of where to search for matches. As the state vector and covariance matrix are updated, the search regions are likewise updated. Thus, areas with low probability of a match are never examined.

Thus, if the uncertainty of a feature (measured as the eigenvalues of the uncertainty matrix of the predicted measurement $\boldsymbol{h}$) is big, it is not wished to search for a match in this region. However, if the uncertainty is less than a certain threshold the search region is accepted. Thus, for a low threshold only small search areas are included.

This section has described the main idea behind NCC with Active Search. This algorithm is used in `EKFmonoSLAM` because it is fast. In the next section, it will be explained how more robust matches are obtained without adding too much computational overhead.

3.9 Robust Estimation Methods

In the `EKFmonoSLAM` implementation, a robust estimation method is applied after the NCC with Active Search which leads to more reliable matches. Robust estimation methods such as Random Sample Consensus (RANSAC) checks the consistency of matches against a global model, and thus is able to discard incorrect matches. RANSAC is one of the most successful robust estimation methods in the computer vision community[Civ+10].

This section will comment on the differences between standard RANSAC and 1-Point RANSAC suggested by Civera et al.[Civ+10]. For simplicity this comparison is done in 2D.

3.9.1 Standard RANSAC

RANSAC aims to efficiently find the model with highest support from the data. Figure 3.5 shows the idea behind RANSAC. In this case the model is a line, thus two points are the minimum number of points necessary to estimate the model. The minimum number is preferred because it reduces the odds of fitting with an outlier. Two random points are drawn and used to estimate the model. The support (consensus) of the model is counted. The model with highest consensus is chosen. The points outside the confidence intervals of this line are considered outliers while the points inside the confidence intervals are considered inliers. Finally, the model is reestimated using all the inliers.

The number of hypothesis n_{hyp} that should be made to get a model that is not estimated by outliers is found with probability p by

$$n_{hyp} = \frac{log(1-p)}{log(1-(1-\epsilon)^m)} \tag{3.26}$$

where ϵ is the outlier ratio and m is the minimum number of points in order to estimate the model. The outlier ratio is calculated as the support of the model divided by the total number of points. This changes for each model, thus the number of hypothesis is recalculated for each iteration.

3.9.2 1-point RANSAC

In SLAM, we have sequential structure from motion (video), thus the frames are taken shortly after each other, and therefore approximately from the same position and angle. This prior information can be used to reduce computational cost of the

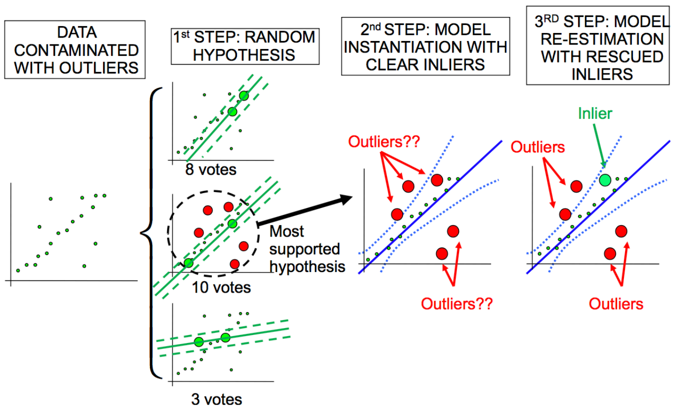

DATA CONTAMINATED WITH OUTLIERS
1st STEP: RANDOM HYPOTHESIS
2nd STEP: MODEL INSTANTIATION WITH CLEAR INLIERS
3RD STEP: MODEL RE-ESTIMATION WITH RESCUED INLIERS
8 votes
10 votes
3 votes
Most supported hypothesis
Outliers??
Outliers??
Inlier
Outliers
Outliers

Figure 3.5: RANSAC steps for the 2D line estimation. In step 1, random hypothesis of sample size two, the minimal number to form a line, is drawn. Then, the line with highest consensus (most votes) is chosen. In step 2, the point outside a confidence intervals are considered outliers and the points inside are considered inliers. In step 3, the model is re-estimated only using inliers [Civ+10].

standard RANSAC algorithm. With this prior information, Civera et al. showed that it is only necessary to use 1 point to estimate a model[Civ+10]. E.g. in Figure 3.6, where the prior information is illustrated as a line with confidence intervals. Step 1 of Figure 3.6 shows that the models are forced through a drawn point and the prior lines x-intercept. Thus, just 1 point can be used to estimate a model.

By using 1-Point RANSAC the computational cost is drastically reduced. To illustrate this cost reduction Equation 3.26 is used. According to [Nis04], minimum 5 points are necessary to estimate the correspondence between two camera frames in a 6 DOF. Assuming an error ratio $\epsilon = 0.4$ and a probability $p = 0.99$, we get from Equation 3.26 $n_{hyp} = 57$. When using 1-point RANSAC and assuming that prior information is available, we get $n_{hyp} = 5$. Thus 1-point RANSAC greatly reduces the computational cost. This means that this robust estimation method can be used while still obtaining real-time performance.

In this chapter, some of the most important theory used in `EKFmonoSLAM` has been explained. First, the concepts of the pinhole model and the homography was established. Then, the inverse depth parametrization was explained, and it was argued for the usage of quaternions. This led the way to explain the core of the software, namely the EKF, in three steps: The state space model, the recursive predicting and updating, and the measurement operator. The second part of the chapter dealt with

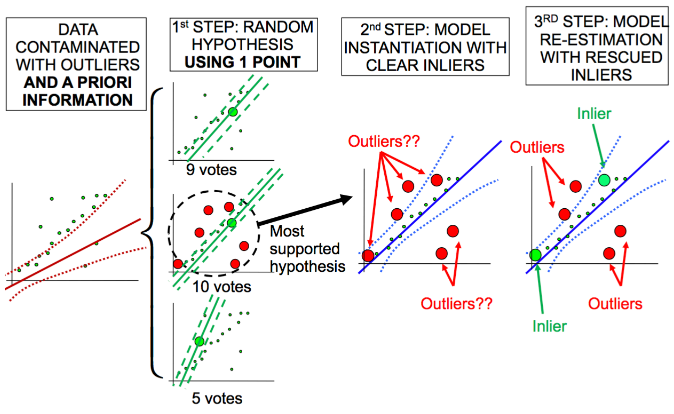

DATA CONTAMINATED WITH OUTLIERS AND A PRIORI INFORMATION
1st STEP: RANDOM HYPOTHESIS USING 1 POINT
2nd STEP: MODEL INSTANTIATION WITH CLEAR INLIERS
3RD STEP: MODEL RE-ESTIMATION WITH RESCUED INLIERS
9 votes
10 votes
5 votes
Most supported hypothesis
Outliers??
Outliers??
Outliers
Outliers
Inlier
Inlier

Figure 3.6: 1-Point Ransac steps for the 2D line estimation. The main difference to Figure 3.5 is that the algorithm assumes a prior distribution over the model parameters is known in advance. Thus, it becomes possible to estimate a line by just one point. [Civ+10]

detecting and matching features. First, it was explained how features are detected using the FAST and the non max-suppression algorithms. Then, it was explained how the features were matched with NCC with Active Search, and finally it was explained how the robust estimation method, 1-Point RANSAC, can be used to get more reliable matches. These three steps all have efficiency and speed in focus which is necessary to obtain real time performance for the SLAM solution.
The next chapter will explain the experimental work used to adapt and evaluate the `EKFmonoSLAM` to the Tobii Pro Glasses 2.

CHAPTER 4
Experimental Design

A big part of this project has been experimental work in order to collect high quality data. This chapter is divided into two sections: First the data collection with the Tobii Pro Glasses 2 is explained. Then it is explained how data is simulated.

It has been chosen to present all the experiments and the considerations behind these in one section such that the reader can get an understanding of the iterative experimental process and such that the reader easily can use this chapter as an overview of available data. All raw data are provided for interested readers in appendix D.

4.1 Experiments & Considerations

This section shortly summarizes the performed experiments and the considerations behind the experiments. The experiments are presented in chronological order such that the reader can get an idea of the iterative process of collecting the data. Links for the raw data can be found in appendix D.

4.1.1 Experiment 1

The first experiment consists of three recordings each with their individual path (see Figure 4.1).

In the first video (Figure 4.1 A), the camera is only moving slightly back and forth, and slightly up and down. Thus, the scene is the same during the entire video. This means that most landmarks are visible during the entire video. This video should test how well the SLAM system finds the same landmarks.

In the second video (Figure 4.1 B), I am walking around in a small closed loop. I am trying to keep my head steady. This video should test if the 3D model constructed via the open source SLAM system is precise enough to recognize that I am walking in a closed loop.

In the final video (Figure 4.1 C), I am walking more or less randomly around the office. I am walking slightly faster and I turn my head faster. This video is more challenging because of the increased speed, head rotation, and distance.

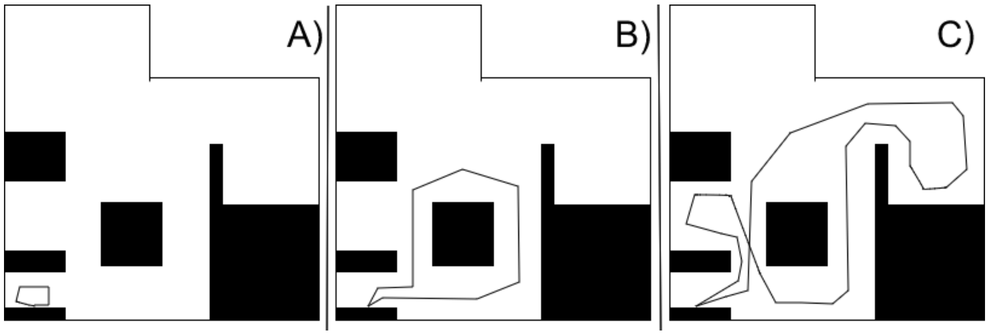


Figure 4.1: The figure illustrates a rough sketch of image laboratory seen from above. The solid black squares illustrate equipment and tables in the lab. The thin black line shows the route I recorded in the three test.

4.1.2 Experiment 2

Recording B) and C) were repeated with slower movement and better lighting. The purpose for these recordings was to have some easier video material for testing.

4.1.3 Experiment 3

Close features are good to estimate translation, however distant features are better for estimating rotation [JM08]. Thus, it was decided to make some recordings with more distant features. Therefore, recordings were also preformed outside.

Several loops around buildings, in parking lots, and in open squares where performed. More than an hour of video was recorded outdoor. These videos should test how the `EKFmonoSLAM` handles distant features and longer paths. The sun was shining very bright that day, thus on several occasions the camera was blinded. Also, big light and dark contrasts and shadows are found in the video.

4.1.4 Experiment 4

The outdoor experiment 3 was repeated on a bright, but cloudy day. The objective of this experiment was to test how `EKFmonoSLAM` performed when it was not blinded by the sun and disturbed by shadows.

4.1.5 Experiment 5

The IMU data from experiment 1 was lost. Therefore, a new recording of the loop in Figure 4.1 B was performed. This recording should be used to test if information about another sensor, the IMU, could improve the estimated path.

4.1.6 Experiment 6

Because the objective is to evaluate the accuracy of the `EKFmonoSLAM`, it is necessary to know the true path. This was done for both an indoor and an outdoor recording.

In the indoor experiment the true path was approximated by an ellipse. Stickers were put on the floor to make sure that I walked on the true path. Thus, by measuring the distance between the stickers the elliptic shape I had walked could be calculated.

An outdoor experiment was also performed to test how well the system behaved to greater distances. In the outdoor experiment it was chosen to use the GPS of a mobile phone to get an true path. The phone's GPS measurements were accessed via the Matlab app which enabled me to get the measurements in latitude and longitude coordinates. The units were converted to obtain the path in meters.

4.1.7 Experiment 7

An replication of the experiments done by Civera et al. [JD06] was performed. Instead of wearing the glasses on my head, the recordings were hand held to obtain more steady images. Three recordings were performed.

In the first recording, a bench with a box on top was recorded. The camera was panned from side to side in a half circle to view the bench and box from the left and right. This movement was done ultra slow. This was to test if the same results could be obtained as presented in [JD06].

The second recording was of an auditorium where I walked from behind the counter and up between the stands. The purpose of this recording was likewise to check if the results presented in [JD06] could be obtained.

Finally, an outdoor video was recorded of the bench and box where the camera was manoeuvred in an square from side to side and back and forth. This recording was to test how well the `EKFmonoSLAM` could estimate a path with minimum rotation.

This section has outlined the performed experiments. Interested readers are encouraged to test the videos on their own SLAM implementations. Note that links for all the data is provided in appendix D. The next section will describe how data was simulated.

4.2 Simulated data

To test how well big and complex system works, it is often helpful to simulate data. When simulating data, one becomes more aware and more in control of what went well and what did not. It is also easier to debug, because both the true input and output values are known, thus the code can be traversed through and the values at all stages can be compared to the true ones in order to find the bug.

In order to simulate data it was necessary to rewrite some of `EKFmonoSLAM`. A function `getFeatures.m` was implemented that given a time index outputs the coordinates of the features projected into the image plane and an index for each features. Several functions and data structures had to be altered such that the features were initialized, matched, and stored based on their index and not their descriptor. This meant that the image-part of the software could be excluded and the EKF could be tested alone.

It was found that it is a good approach to begin simulating the simplest useful data. In this report, a box of points without noise were simulated (Figure 4.2 (left)). The simulated data points were projected into the image plane shown i Figure 4.2 (right) using the camera matrix of the Tobii Pro Glasses 2.

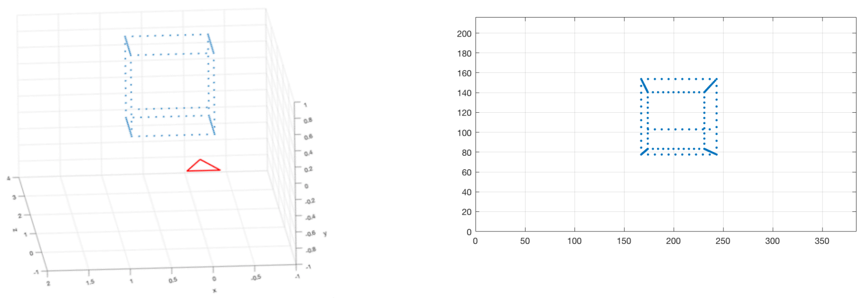


Figure 4.2: The left figure illustrating the simulated data. Data points form a box without noise. The red triangle illustrate the camera position and orientation. The right figure illustrate the simulated data projected into the camera of the Tobii Pro Glasses 2.

By simulating the data without noise, the model should be accurate to machine precision. The constant velocity model was initialized with the same speed as the data was simulated after. Thus, the prediction should be spot on from step one. This insight is valuable for debugging.

This chapter has described my experimental work and the considerations behind. The next chapter will explain how `EKFmonoSLAM` can be adapted to the data described in this chapter.

CHAPTER 5

Implementation

The goal of this chapter is to explain how `EKFmonoSLAM` is adapted to the image data collected with the Tobii Pro Glasses 2. The first Section 5.1 outlines the `EKFmonoSLAM` implementation. This section will give readers a better understanding of the iterative steps of the algorithm. In Section 5.2, the camera calibration of the Tobii Pro Glasses 2 is described. Following in Section 5.3 a constant for downsizing the recorded images is found to reduce the computational cost. Then, in Section 5.4 thresholds for the feature detector algorithms, FAST and the non maximum suppression, that provide a sufficient number of features from the images recorded by the Tobii Pro Glasses 2 are established. As a form of validation of the adaptation, Civera et al.'s [JD06] results are replicated in Section 5.5 using hand held recordings from the Tobii Pro Glasses 2.

5.1 Software

A big part of this project has been to understand and analyse `EKFmonoSLAM`. `EKFmonoSLAM` is a huge program consisting of 131 files. Therefore, it is chosen to dedicated a section to give an overview of the software. It is believed that this overview will make it easier for the reader to understand which parts of the implementation that are altered in order to adapt the software to the data from the glasses. In appendix E a more detailed and language specific manual of the software is provided. It is recommended that readers interested in working with the software visit this manual.

Figure 5.1 shows a flowchart of the algorithm. To begin with, illustrated as the purple box, the EKF is initialized. This includes initialization of the state vector $\boldsymbol{x}$, the uncertainty matrix $\mathbf{P}$, and the error operator $\boldsymbol{\Gamma}$. It also covers initialization of the internal camera parameters.

Secondly, in the first green box, features are found in the first frame using FAST-9 with non-max suppression and stored in inverse depth coordinates in the state vector.

Thirdly, the algorithm enters its recursive stage. In the grey box, the found features are managed. This includes testing if the features can be converted to an Euclidean representation based on an linearity index in order to reduce the computational cost,

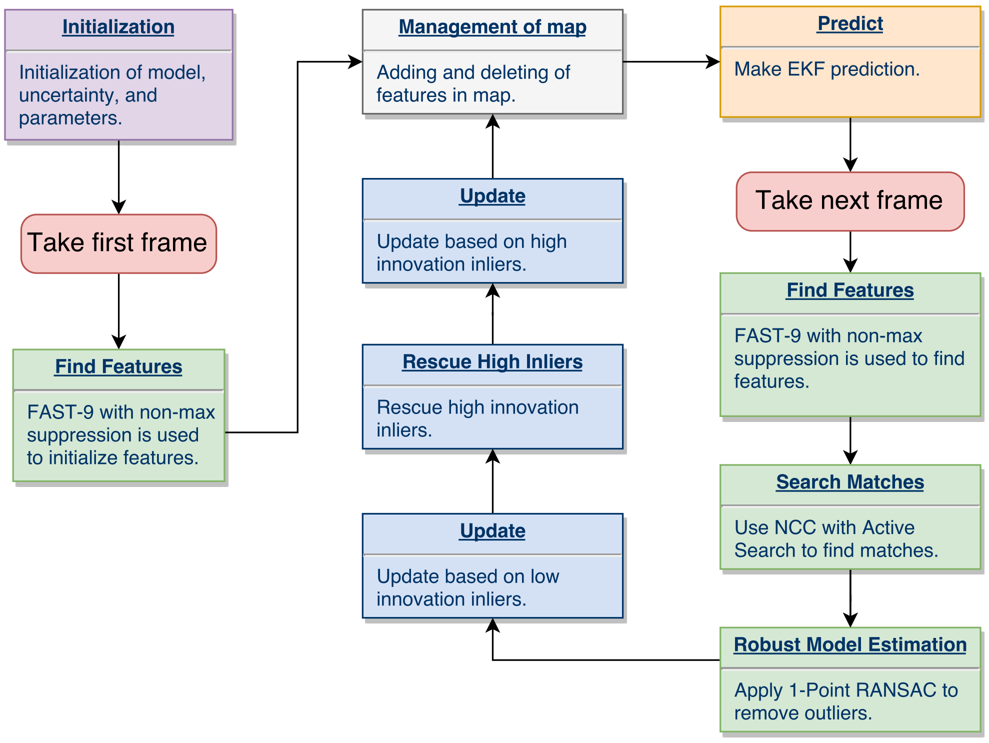

Initialization
Initialization of model, uncertainty, and parameters.
Take first frame
Find Features
FAST-9 with non-max suppression is used to initialize features.
Management of map
Adding and deleting of features in map.
Update
Update based on high innovation inliers.
Rescue High Inliers
Rescue high innovation inliers.
Update
Update based on low innovation inliers.
Predict
Make EKF prediction.
Take next frame
Find Features
FAST-9 with non-max suppression is used to find features.
Search Matches
Use NCC with Active Search to find matches.
Robust Model Estimation
Apply 1-Point RANSAC to remove outliers.

Figure 5.1: The figure shows a flowchart of the `EKFmonoSLAM` algorithm. The purple box indicated initialization and preallocation of variables. The red boxes indicated fetching of images. The green boxes indicated image operations. The grey box indicate storage and deletion of features. The orange box shows the prediction step, and the blue boxes shows the update step of the EKF.

and deletion of features that are not measured half as many times as they are predicted.

In the orange box, a prediction of the next state for both the features and the camera is made using the transition equations described in the Section 3.4.

Following the prediction, a new image is fetched (red box) and three image operations are applied (the green boxes) to this image. First, features are found in this new image. Then, the uncertainties about the landmarks which are already in the state vector are used to perform Active Search. Features from the previous frame and the new frame are matched with NCC. Finally, the 1-point RANSAC is applied to exclude outliers. Thus, these three steps find the matches that represents the correspondence between the the new image and the previous.

Subsequently, the EKF is updated based on this correspondence between the frames. This process is also divided into three steps (the blue boxes). First, the filter is updated based on the low innovation inliers (the features with small distance to the most supported hypothesis). These low innovation inliers are often distant points as they are often viewed more stationary. This updating greatly reduced the uncertainty about the model. The reduction in uncertainty is used to rescue high innovation inliers. These high innovation inliers are often closer points where the translation has greater influence on their position. Finally, the filter is updated based on these high innovation inliers. It is possible to divide the updating into two steps because of the linearity of the filter. According to Civera et al. [Civ+10], splitting the updating into two steps does not have noticeable effect on the computational cost as long as the state vector is significantly larger than the measurement vector which is usually the case in SLAM.

This recursive structure continues till the last frame is reached. In appendix E, a manual on getting started and a more detailed Matlab specific outline of the algorithm is presented.

This section has presented an overview of the recursive structure of the `EKFmonoSLAM`. Hopefully, this overview will help the reader understand which parts of the implementations are altered in order to adapt the software to data from the Tobii Pro Glasses 2. The next section will describe how the initialization, depicted as the purple box 5.1, is altered to the front camera of the glasses.

5.2 Camera calibration

The goal of camera calibration is to determine the internal camera parameters **A** described in Section 3.1. These parameters have a huge influence on how the world is perceived in the images and therefore it is very important to find a good estimation of these parameters. This section will describe how the Matlab calibration application is used to find the internal camera parameters of the front camera on the glasses.

The `EKFmonoSLAM` implementation uses the following parameters (Table 5.1).

Parameter	Value
n_{row}	Number of pixel in row
n_{col}	Number of pixel in column
u_0	The principal offset x direction
v_0	The principal offset y direction
κ_1	1. distortion constant
κ_2	2. distortion constant
f	The focal length
d	The ratio between the cameras' digital sensor and the image size.

Table 5.1: The figure shows the necessary internal camera parameters used by the `EKFmonoSLAM` implementation.

Matlab has a camera calibration application which was used for the camera calibration. This application was given 13 full HD images (1920x1080) of a checkerboard taken from different angles and distances. The area of each square was known to be 15 mm^2 and all the corners of these squares was known to lie in the same plane, thus the world coordinate could be reduced to a plane where the relative positions between the points were known. The Matlab calibration application automatically found all the corners $\boldsymbol{q}_j$ of the squares in the checkerboard in each of the 13 images. Thus, both $\boldsymbol{q}_i$ and $\boldsymbol{q}_j$ were known. It was wished to determine the operator **H** which is possible based of the homographical relationship described in Equation 3.2.

The Matlab calibration application minimized the distance between the points mapped to world coordinates via the homography and the points on a checkerboard (Equation 3.3). The application found the internal parameters presented in Table 5.2.

From Table 5.2 it is seen that not all the necessary parameters are explicitly given in Table 5.1. However, using Equation 5.1, is is possible to find d.

$$d = \frac{1}{2}(d_x + d_y) = \frac{1}{2}\left(\frac{W}{n_{col}} + \frac{H}{n_{row}}\right) \tag{5.1}$$

Parameter	Value
n_{row}	1080 pixels
n_{col}	1920 pixels
u_0	968.9 pixels
v_0	499.8 pixels
κ_1	0.0286 mm
κ_2	-0.0669 mm
f_x	1139.1 pixels
f_y	1143.8 pixels

Table 5.2: The estimated internal parameters found by the Matlab calibration application. Note that not all the necessary parameters are provided compared to Table 5.1.

where W and H are respectively the width and height of the camera's digital sensor in millimeters[1].

And the focal length can be converted into millimeters by multiplying with d.

$$f = fd = \frac{1}{2}(f_x + f_y)d \tag{5.2}$$

Thus, Table 5.3 shows the estimated internal parameters for the front camera of the Tobii Pro Glasses 2.

Parameter	Value
n_{row}	1080 pixels
n_{col}	1920 pixels
u_0	968.9 pixels
v_0	499.8 pixels
κ_1	0.0286 mm
κ_2	-0.0699 mm
f	1.6232 mm
d	0.0014 mm/pixels

Table 5.3: The figure shows the estimations for the necessary internal camera parameters used by `EKFmonoSLAM`.

In order to make sure that these internal camera parameters were correctly estimated, the camera model was validated by selecting one corner on two random squares on an image of a checkerboard.

[1] Values of W = 2.7328 mm and H = 1.5344 mm are from a e-mail correspondence with Tobii's Technical Support Engineer Lea Kümper 29/9/2017

These two 2D points were projected onto the 3D checkerboard plane through the homography with the found internal camera parameters. This resulted in a distance between the selected two points of $[75.03; 29.67]^T$. As mentioned, each side of the squares was 15 mm, thus the true distance between the points is [75; 30]. The small error is most likely due to manual selection of points or small errors in the camera matrix.

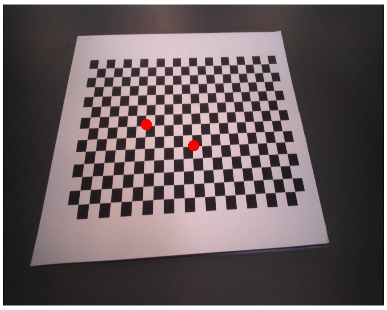


Figure 5.2: Two corners selected on the checkerboard

Finally, the selected points were projected back onto the 2D image plane through the inverse homography. The coordinates of the back projected points were compared to the originally selected points. It was found that the distance between the original points and the back projected points were respectively $[0.11 \cdot 10^{-12}; 0]^T$ and $[0.11 \cdot 10^{-12}; 0.05 \cdot 10^{-12}]^T$.

This procedure was repeated for points further from each other and similar results were found. Thus, the estimation of the camera matrix is correct.

In this section, the Matlab calibration application was used to find the internal camera parameters of the front camera of the Tobii Pro Glasses 2 (Table 5.3). It is important to note that these parameters were found from full HD images (1080x1920). However, because of the linearity of the homography, the parameters can be scaled by the same factor as the images. These parameters will be used to initialize the camera in `EKFmonoSLAM`. The next section will investigate how the full HD images can be downsized to decrease the computational cost.

5.3 Scale Comparison

The images recorded by the Tobii Pro Glasses 2 are full HD (1920 x 1080), and because of the big size of the images operations become very computational expensive. They become less expensive if the images are downsized. However, downsizing decreases the information content of the image. This section will investigate how much the images should be downsized.

Civera et al. use images of size 320x240 in [JM08]. It is assessed that it will provide better results to downscale with a constant that gives approximately the same image size as used in `EKFmonoSLAM` as many hard coded values throughout the code (e.g. in feature detection and feature matching) are chosen accordingly to the size of the images. Furthermore, it is preferred to scale both dimensions by a common constant without cropping the image, because then the corresponding scaled internal camera parameters can easily be obtained by multiplying with this common constant. Thus, it has been chosen to test scaling by 1/5 (image size: 384x216) and scaling by 1/6 (image size: 320x180).

Table 5.4 shows that the elapsed time is 9 % shorter when downsizing the images by 1/6 than the elapsed time when downsizing by 1/5.

Scale	Time
1/5	479.70
1/6	439.45

Table 5.4: The table shows the elapsed time for the two experiments in seconds. It is seen that the experiment with scale 1/6 runs 9 % faster than the experiment with scale 1/5.

The increased speed advocates downsizing the images by 1/6. However, also the image information (map size) has been investigated. Figure 5.3 shows the map size and the number of matched features for each frame for the experiments with different image sizes.

Figure 5.3 shows that both more features and matches are found in the experiment with scaling 1/5 (red) than in the experiment with scaling 1/6 (blue). The images become increasingly blurred when they are downsized. This increased blurriness makes it harder to detect features and increase the measurement error of the detected features. This advocates that higher resolution images results in more accurate and stable SLAM solutions.

This section has investigated how the scaling of the images effects the SLAM problem. Two scaling factors were investigated: 1/6 and 1/5, because these scaling factors

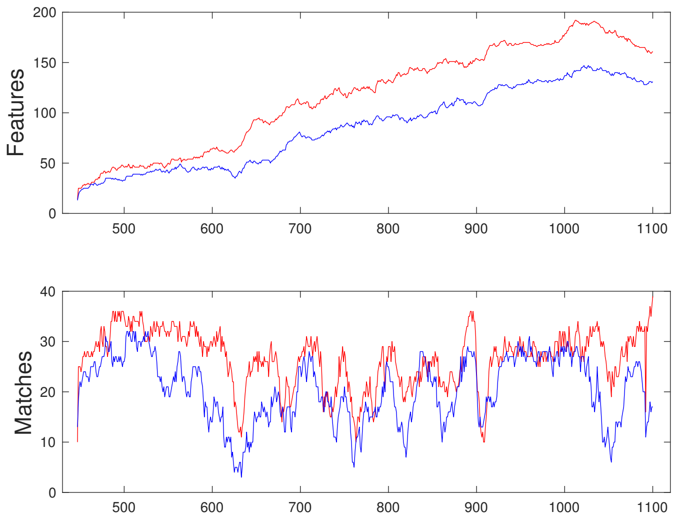


Figure 5.3: The upper subfigure shows the map size (number of landmarks) for the experiment with scale 1/5 (red) and 1/6 (blue). The lower graphs shows the number of matched features in the experiment with scale 1/6 (red) and the experiment with scale 1/5 (blue).

made the image size correspond approximately to the image size used by Civera et al.[JM08]. On the one hand, it was found that the run time of the experiment with scaling 1/6 was 9 % faster. On the other hand, it was found that more features and matches were found in the experiment with scale 1/5. It is assessed that the increased information, and thus accuracy, is more important to the objective of this thesis than an improvement in the run time. Therefore, it is chosen to use a scaling factor of 1/5 (image size: 384x216). This scaling constant will be used in the rest of the thesis for analysis. The next section will investigate how the thresholds in the feature detector have to be lowered in order to find a sufficient number of features.

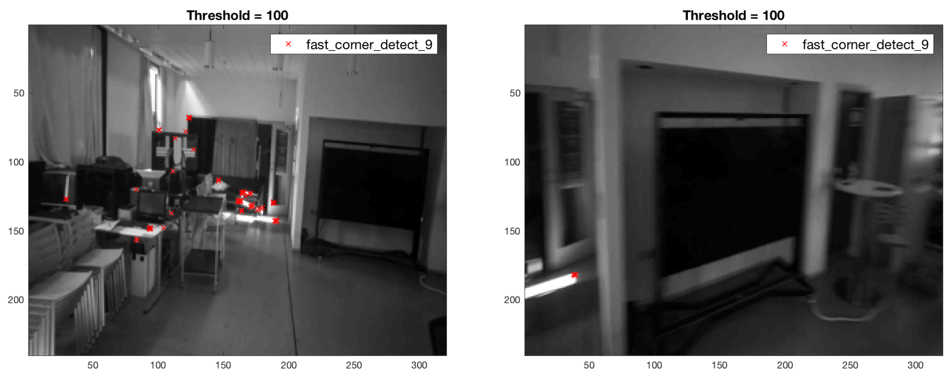
5.4 Thresholds for the Feature Extraction

In computer vision, points are often used to find correspondences between images. To do so, it is important to find points with intrinsic meaning such that the 'same' point can be found in two images. This section will describe the trade-off between including too many or too few features and determine a reasonable thresholds for the feature detector FAST-9 with non maximum suppression used in `EKFmonoSLAM`.

It is a trade-off how many features should be included when finding the image correspondence. Selecting a high threshold will include few features of high quality (more intrinsic features), and thus it will be easier to find the 'same' feature in two images. However, each match will bear a high weight, thus an incorrect match can be rather significant for the model estimate. In addition, when just a few number of features are detected the probability of not finding any correspondences between two frames increases.
On the other hand, selecting a low threshold will include many, but less intrinsic features, which results in more incorrect matches. However, the model will trust each match less, and thus the influence of each incorrect match is reduced. Also, including more features is more computational expensive.
As this trade-off has been established, the rest of this section will seek to find thresholds that neither include too many or too few features.

As mentioned in the Section 3.7, the `EKFmonoSLAM` implementation uses the FAST-9 algorithm followed by a non maximum suppression to detect features. Both methods use a threshold to determine how many features should be included. First, the threshold for the FAST-9 will be investigated. Then, the barrier for the non maximum suppression.

Figure 5.4 shows the number of features detected by the FAST-9 for different thresholds.



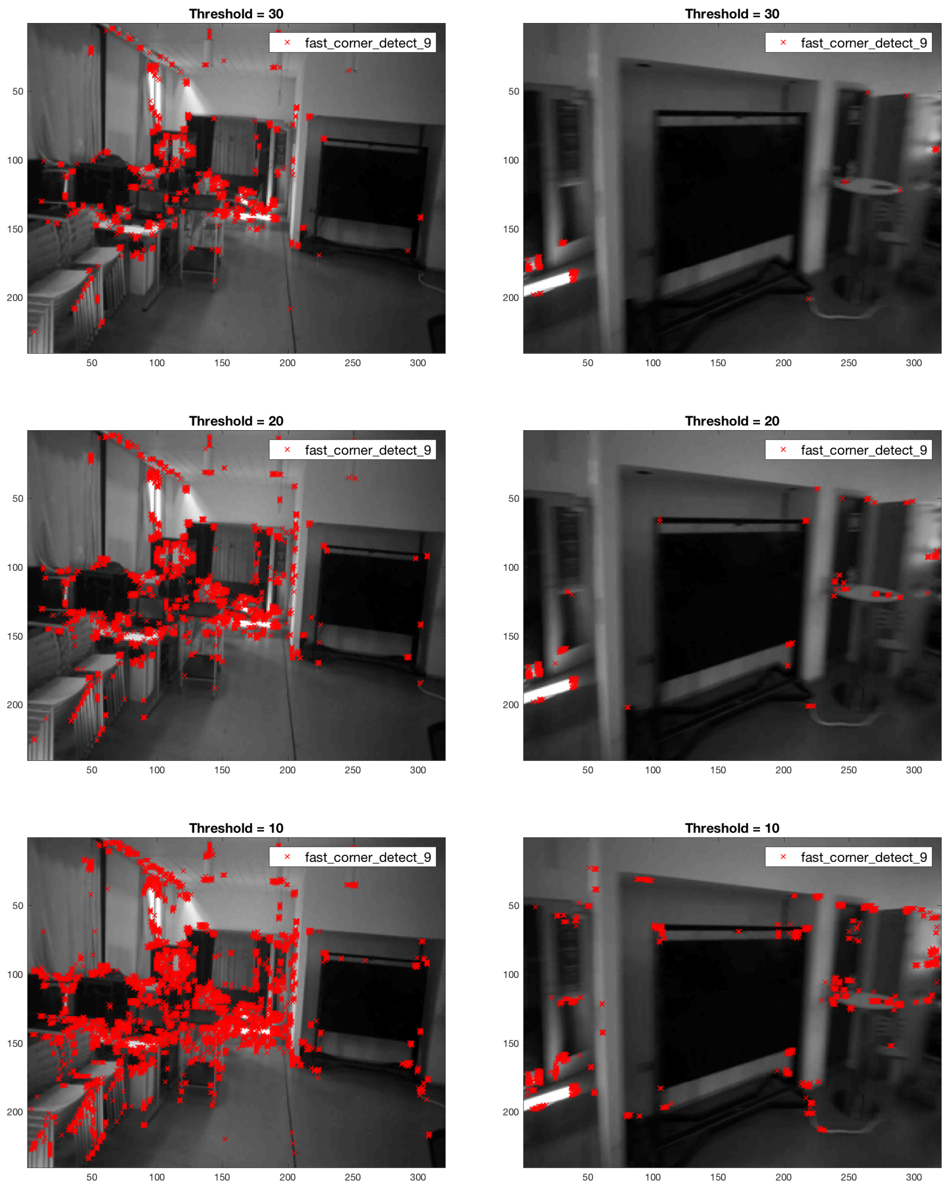


Figure 5.4: The figure shows which features are detected by the FAST-9 when adjusting the threshold for both a blurred image (right column) and a non-blurred image (left column). The default threshold is 100. From the figures, it is seen that the number of detected features increase as the threshold decreases. It is chosen to use a threshold of 20.

Figure 5.4 shows that lowering the threshold results in more detected features. Based on the figure, it is assessed that the default threshold of 100 is too high because only one feature is detected in the blurred image. By looking at the right column of images, one would expect that the corners of the TV were intrinsic features. This is the case with a threshold less or equal to 20. However, it is assessed that too many features are detected with a threshold of 10. Thus, it is found that a threshold of 20 gives a sufficiently amount of features and that these features still are sufficiently intrinsic.

It was also investigated what the barrier of the non maximum suppression should be. By using a threshold at 20 for the FAST-9, and trying different barrier values for the non maximum suppression the barrier was tested. Figure 5.5 shows a zoom of a cluster of points detected in the upper left corner of the TV in the blurred image.

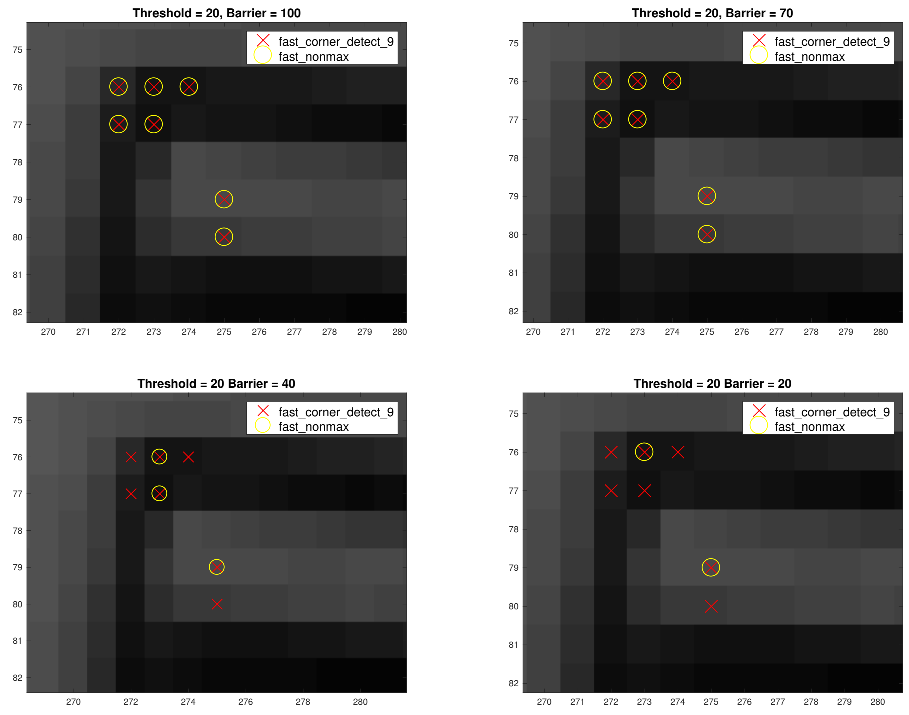


Figure 5.5: The figure shows a zoom of the left corner of the TV in the blurred image from Figure 5.4. Different barriers of the non maximum suppression have been used. In `EKFmonoSLAM`, the default barrier by Civera et al. is 100, however it has been assessed that in this case, it is better to use a barrier of 20.

Figure 5.5 shows that by decreasing the barrier of the non maximum suppression, only the most intrinsic point in a 3x3 region is selected. From the figure, it is seen that the barrier has to be decreased to 20 in order for only one point in each cluster to be selected. Thus, based on Figure 5.5 it is chosen to use a barrier of 20.

Figure 5.6 summarizes how the threshold of the FAST-9 algorithm and the barrier of the non-maximum suppression effects the number of features. The figure shows that there is a huge difference in features detected by the FAST-9 whether the image is blurred or not blurred. The figure supports the lowering of both thresholds.

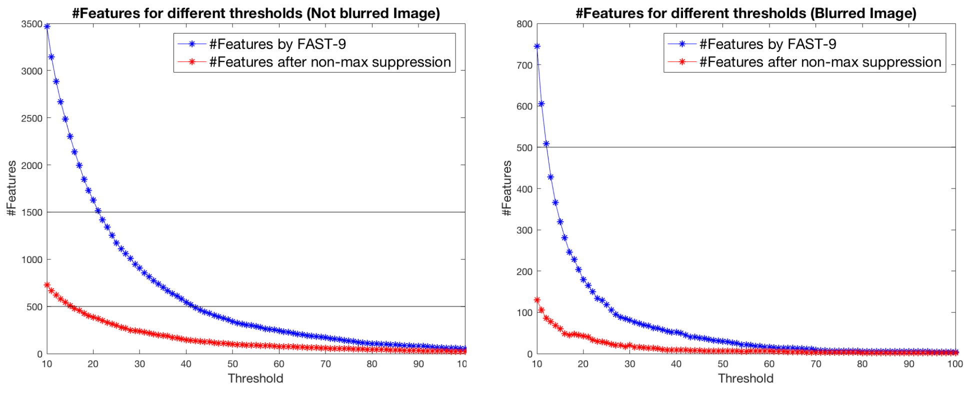


Figure 5.6: The figure shows the number of features detected in a blurred image (right) and a not blurred image (left). The blue points are the features detected by FAST-9, and the red points are the features after performing the non maximum suppression. The black lines indicate the suggested number of features by [Ros+08]. Note that the scale on the y-axis is different in the two plots.

This section has investigated the lowering of the thresholds for the FAST-9 and the barrier for the non maximum suppression. It has been assessed that the default values should be lowered in order to obtain a sufficient amount of features - especially in blurred images. Thus, a threshold of 20 has been applied for the feature detector and a barrier of 20 is used for the non maximum suppression. These thresholds will be used in the rest of this thesis unless otherwise stated.

Subsequently, the internal camera parameters have been found and used to initialize the `EKFmonoSLAM`. It has been found that the best results compared to the computational cost are obtained by scaling the images by a factor of 1/5. And it has been found that the threshold and barrier of the FAST-9 and non max suppression should be 20. By establishing these parameters, it is possible to replicate Civera et al.'s experiments using data recorded by the Tobii Pro Glasses 2.

5.5 Validation of implementation

This section will, as a form of validation for the implementation, demonstrate how Civera et al.'s two experiments described in [JD06] can be replicated using `EKFmonoSLAM` with the previously described internal camera parameters, thresholds, and downsized images collected with the Tobii Pro Glasses 2. Civera et al.'s experiments are in this thesis referred to as the outdoor experiment and the indoor experiment and can be found at the following links: http://webdiis.unizar.es/%7Ejosemari/in.avi and http://webdiis.unizar.es/%7Ejosemari/out.avi.

In the outdoor experiment Civera et al. have placed several boxes on top of a bench. A small black square has been marked as the front of the box. The figure below compares the setting of the Civera et al.'s outdoor experiment by a replication of the experiment.

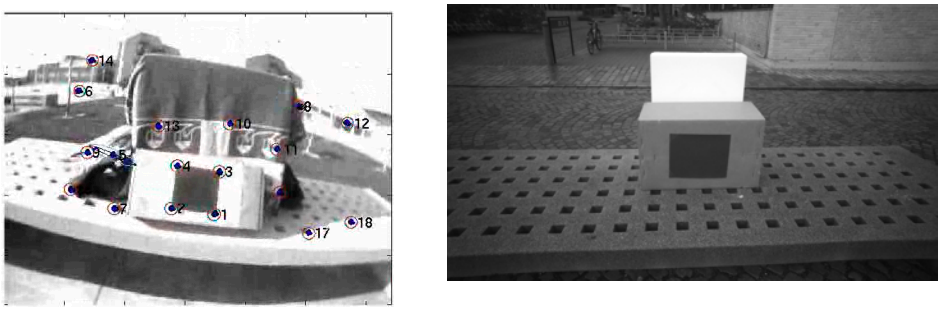


Figure 5.7: The figure shows a comparison between the landscape used by Civera (left) and the one used to replicate his results (right).

In Civera et al.'s outdoor experiment, a hand held camera is maneuvered such that it starts out facing the box from the front (as shown on Figure 5.7). Then, the camera is moved a quarter of a circle such that the box is seen from the right side, and then half and circle back such that the box is seen from the left side, and finally the camera is moved to the original position. Note that the movement is very slow.

Like the settings has been tried replicated, the slow movement has also been tried replicated. To obtain a more stable recording the Tobii Pro Glasses 2 are hand held (watch video: http://elers.dk/Videos/halfcircle.avi). Using the previous described alterations of the software, similar results as Civera et al.'s were obtained.

From Figure 5.8, it is seen that the final path of both Civera et al.'s experiment and the replicated experiment shows the correct movement. It is seen that both features close to the camera and more distant features are located with very low uncertainty

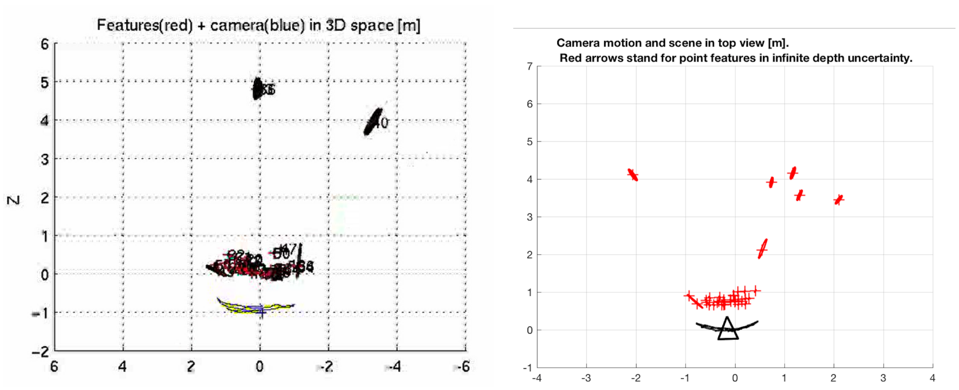


Figure 5.8: The figure shows a comparison between the final path in Civera et al.'s outdoor experiment (left) and the final path of the replicated experiments (right).

(the ellipse around the features are very small). Thus, very similar results are obtained.

In the indoor experiment Civera et al. have filmed an auditorium. Again, the setting has been tried replicated (Figure 5.9).

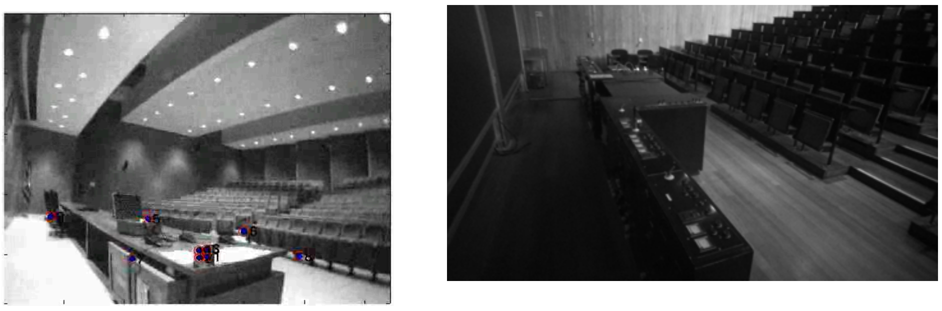


Figure 5.9: The figure shows a comparison between the landscape used by Civera (left) and the one used to replicated his results (right).

In this experiment, a hand held camera set out from behind the counter, it is moved up through the stands, and then back across the scene. This movement has been tried replicated. The glasses are also hand held in this experiment (watch video: http://elers.dk/Videos/auditorium.avi). The final paths are showed in Figure 5.10.

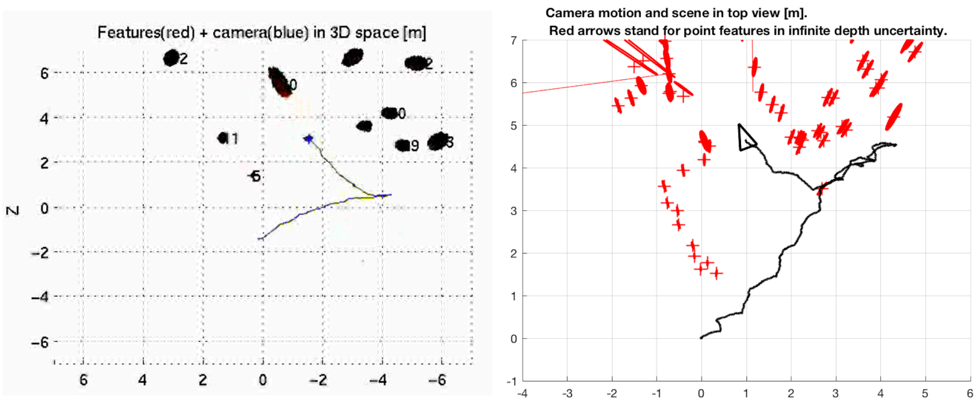


Figure 5.10: The figure shows a comparison between the landscape used by Civera (left) and the one used to replicated his results (right).

Again the path of the replicated experiment does very well estimating the actual path. Also, there is very little uncertainty about most of the features like in Civera et al.'s experiment. It is important to note that the data and the actual path for the two experiments are not identical, thus it is not expected that the final paths are either.

The results Civera et al. present in [JD06] were replicated with data recorded with the Tobii Pro Glasses 2. Note that the glasses was hand held in these experiment to get more steady recordings and imitate Civera et al. most accurately. Using the parameters found in the Sections 5.2, 5.3, and 5.4 very similar results to those presented in [JD06] were obtained. Thus, the adaptation of the `EKFmonoSLAM` implementation to the image data collected by the Tobii Pro Glasses 2 has been successfully validated.

Of course it is not of much interest how the `EKFmonoSLAM` performs when the glasses are hand held. Therefore, the obvious next step is to test how well the `EKFmonoSLAM` performs when the Tobii Pro Glasses 2 are worn as glasses. The next chapter will present and discuss the obtained results when the glasses were head-worn.

CHAPTER 6

Results & Discussion

In this chapter the results of the thesis will be presented and discussed. The structure of the chapter will represent the iterative process of hypothesizing, testing, and evaluating. In Section 6.1, it is presented how the `EKFmonoSLAM` performs on data recorded by a person wearing the glasses in an indoor experiment. In this section, it is found that the software is both very dependent on the initial inverse depth distribution and it is sensitive to fast rotations. It is investigated in Sections 6.2 and 6.3 how a good prior inverse Gaussian distribution to initialize features can be found. It is hypothesized that the sensitivity to fast rotations is due to the non-invariant feature matching algorithm. Thus, in Sections 6.4 and 6.5 it is investigated how sensitive NCC with Active Search is towards rotations. It is hypothesized that an outdoor experiment, where there are more distant features, will decrease `EKFmonoSLAM`'s sensitivity towards rotation. This is investigated in Section 6.6.

6.1 Wearing the Tobii Pro Glasses 2

This section will present the results obtained from experiments 1 (described in Section 4.1) using the previously described adaptation of the software. As stated previously, the data collected wearing the Tobii Pro Glasses 2 varies in difficulty in the three recordings in experiment 1. In the first recording, the camera has little movement and slow rotations, in the second recording, the camera is manoeuvred in a loop around a table, and in the last recording, a longer distance is covered with faster movement and rotations.

6.1.1 Recording with little movement

Figure 6.1 shows the first two images in the recording with little movement. It is seen from the top left image that some features are already initialized in the first iteration. This undelayed initialization is possible because of the inverse depth parametrization with the prior knowledge that the observed features are in front of the camera [JM08].

It can be seen from the size of the red circles in the top right and bottom image in Figure 6.1 that the uncertainty decreases from the first to the second iteration as more points are initialized and recognized. The right column in Figure 6.1 shows

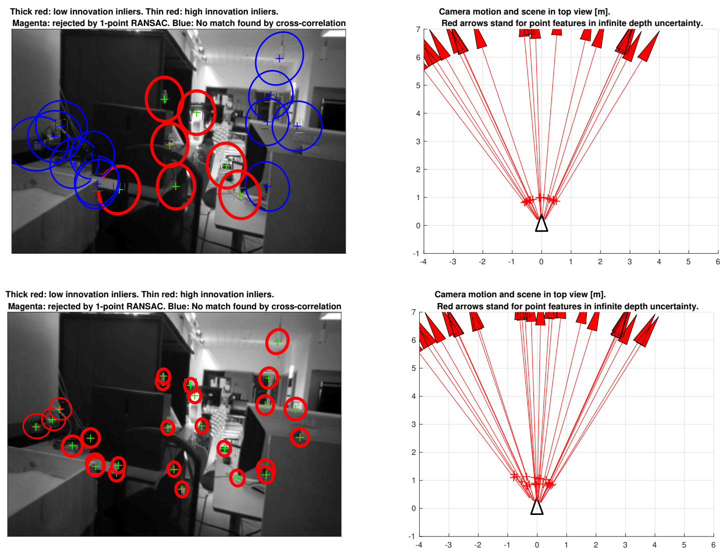


Figure 6.1: The upper row shows the first step in the experiment. The lower row shows the second step. The red circles are high innovation inliers and the blue circles show features where a match is not found. The size of the circles illustrate the uncertainty estimate. The initialized points are to begin with set to infinity (red arrows in right plot)

that the distance to most features are estimated to infinity to begin with. The reason is that the features are observed with low parallax, thus the uncertainty about their position is high, and therefore it is chosen to estimate them at infinity.

The implementation solves the proposed SLAM problem. It estimates the 3D position and movement through the experiment. Figure 6.2 shows the last iteration which coincides with the expected path (Figure 4.1). A video of the solution can be found at http://elers.dk/Videos/atDesk.avi.

Figure 6.2 shows the last frame of the first recording. It can be seen from the left image that the scene (and thus the position) is roughly the same as the first image in Figure 6.1. The map verifies that the starting and ending positions are very close (right plot in Figure 6.2). Furthermore, it is seen from the estimated map that the position uncertainty of the features have decreased and only the position of few features are estimated to infinity.

Thus, satisfactory results were obtained using the `EKFmonoSLAM` on the data recorded with little movement.

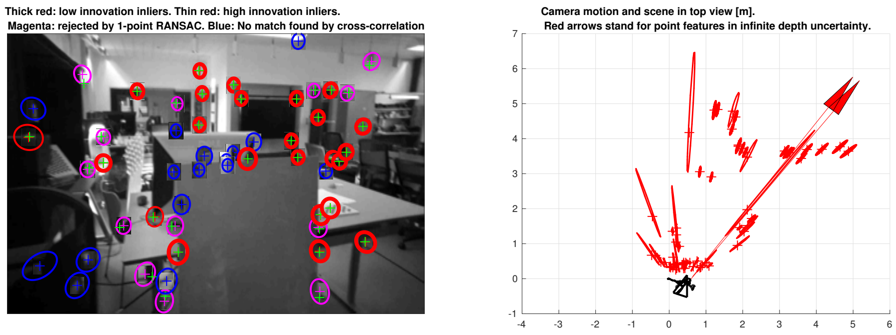


Figure 6.2: The figure shows the last iteration of the first experiment. The right plot shows the estimated map and path. The features with a magenta circle are features that have been rejected by 1-Point RANSAC.

6.1.2 Recording with loop around table

The aim of the second recording was to estimate how well the `EKFmonoSLAM` could estimate the closing of a loop. This experiment had more movement and more rotation than the first experiment. Figure 6.3 shows the final path.

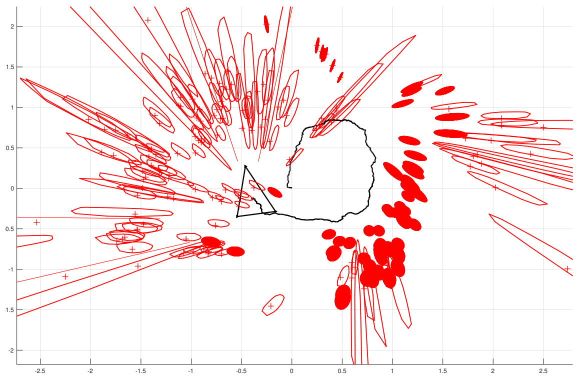


Figure 6.3: The final path of the experiment where the camera was manoeuvred around a table in a loop.

Figure 6.3 shows the final path of the recording where the camera was manoeuvred in a loop as illustrated in Figure 4.1. From the estimated path the structure of the loop is recognized. Features are detected at all sides. However, the estimated path is drifted such that the starting point and end point do not meet as expected. Also, the scale seems to be wrong. The radius of the circle is estimated to approximately 0.5 meters whereas it should be approximately 1.5 meters. A video of the solution can be found at http://elers.dk/Videos/loop.avi.

An even more challenging recording was tested. This recording consisted of a longer path with faster rotations and movements (Figure 4.1 (C)).

6.1.3 Recording with random walk in the laboratory

Figure 6.4 shows the estimated final path from the recording that covered the longest distance and had the fastest rotations. Comparing the estimated path in Figure 6.4 with the expected path in Figure 4.1 C, it is seen that the number and direction of the turns coincides. However, it is expected that the estimated path starts and ends approximately the same place which is not the case in Figure 6.4. From this figure, it is seen that a rather big drift occurred. Again, the scaling is off as the size of the laboratory is significantly larger than 2 m^2. A video of the solution can be found at http://elers.dk/Videos/aroundImageLab.avi.

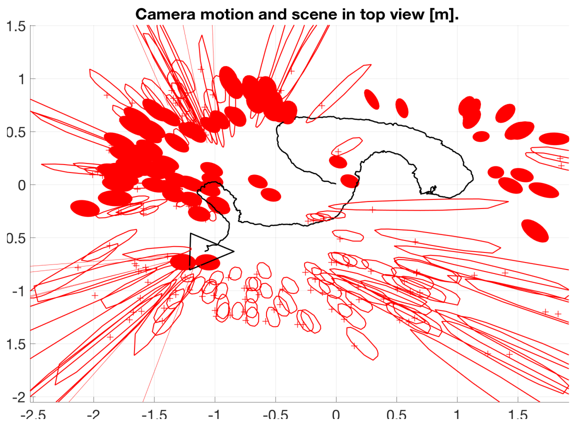


Figure 6.4: The final path of the experiment where the camera was manoeuvred randomly around the laboratory.

In this section the `EKFmonoSLAM` implementation was tested on three videos that had been recorded wearing the glasses. The software performed well in the first recording with little movement. However, it was observed in the two last recordings that the software could not estimate the correct scale of the map and that the estimated path had a tendency to be drifted when there were fast rotations.

The next two sections will investigate how a more accurate scaling can be selected. First by simulating data and then by using recorded data. Following this investigation, the robustness of the algorithm will be tested in order to explain why the drifts in the estimated path happens during the fast rotations.

6.2 The importance of a good prior

This section will use simulated data to investigate how the inverse depth initialization of features effects the scaling of the estimated path. The simulation setup described in Figure 4.2 will be used to examine these effects. The main advantage of using this simulated data compared to some recorded data is that the true position and the true model is known.

When observing point features through a camera, the distance to these features can only be determined up to scale - meaning that the distance between the camera and the features could be anything between zero and infinity. In order to initialize features after just one image Civera et al. make an educated guess about a distribution describing the distances to the observed features. Because of the inverse parameterization in `EKFmonoSLAM`, this initial guess is an inverse Gaussian distribution. By default in `EKFmonoSLAM`, Civera et al. have chosen the mean and standard deviation of this distribution to one meter. Civera et al. state that this initial distribution must be "derived heuristically ... to cover the 95% confidence intervals" [JM08] of the features. If this is not the case, like it was to begin with in the simulated data, where the initial distances between the simulated points and the camera were greater than three meters, the scaling will be off. In this example it meant that the EKF thought that the points were approximately three timers closer than they actually were. And because the translation of closer point requires less velocity, the model thought that the velocity of the camera was approximately a third of the true velocity.

The plot to the right in Figure 6.5 shows the relative errors after 30 iterations as a function of the initial distance.

From Figure 6.5, it is seen that the relative error in position decreases as the initialization distance comes closer to the true initial distance to the features (three meters). This figure shows the importance of good prior knowledge to get the correct scaling. The right plot shows the inverse Gaussian distribution with the minimizer at $d_0 = 2.9$ and fixed $\sigma = 1$. It is seen that the 95 % confidence intervals are very wide. Also, it is seen that the median of the distribution is 1.2. The median is defined as the middle value, thus it is expected that approximately equally many observations have greater and shorter distance to the camera. This is not the case, as all features are located between 3 and 4.12 meters, and thus the median distance being 3.46. Based on the wide confidence intervals and the too small median, it does not seem ideal to fix the standard deviation.

Instead of fixing the standard deviation, Civera et al. suggest that the standard deviation is dependent on the initial distance[JD06]. They suggest the following relationship.

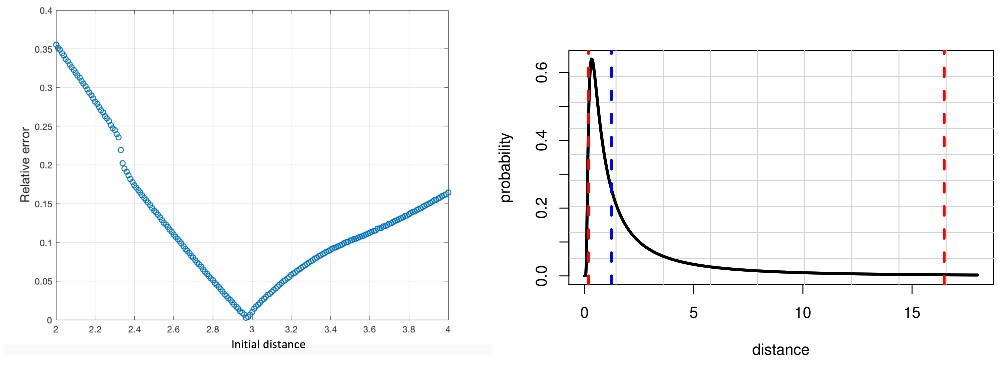


Figure 6.5: The left plot shows relative error after 30 iteration with different initialization depths and standard deviation fixed at 1. The right plot shows the density function for the inverse Gaussian distribution using the minimizer 2.9 as mean. The red lines indicate the 95 % confidence intervals and the blue line indicate the median.

$$\hat{\rho}_0 = \frac{\rho_0}{2} \qquad \sigma_\rho = \frac{\rho_0}{4} \qquad \rho_0 = \frac{1}{d_0} \tag{6.1}$$

Even though this relationship is not applied in `EKFmonoSLAM`, it is tested how this dependency effects the inverse Gaussian distribution and the error function.

From Figure 6.6 a similar error curve is seen as in Figure 6.5. The minimizer is found to $d_0 = 3.4$. Using the relations described in Equation 6.1, this gives $\rho_0 = 0.29$, $\hat{\rho}_0 = 0.145$, and $\sigma_\rho = 0.073$. These parameters are used to obtain the inverse Gaussian density distribution showed in the right plot in Figure 6.6. From this plot, it is seen that the confidence intervals are more narrow. And all features are still included in the 95 % confidence interval. Furthermore, it is seen that the median is 3. This is not the middle distance of the features, however it is significantly closer than in Figure 6.5. Therefore, based on the more narrow confidence intervals and the more appropriate median, it is chosen to incorporated the relations described in Equation 6.1 in `EKFmonoSLAM`.

This section has described how homogeneous 3D world coordinates are only described up to scale. The importance of this scale as a form of prior information was investigated through simulated data. It was found that when initializing a feature in inverse depth coordinates, the initial heuristic derived distribution describing the distance from the camera to the features should cover the 95 % confidence region in order to provide good position and velocity estimations. Furthermore, it was chosen

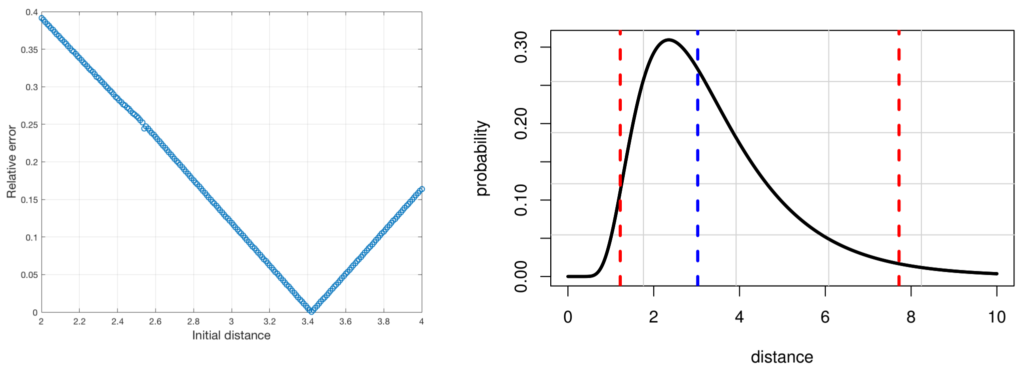


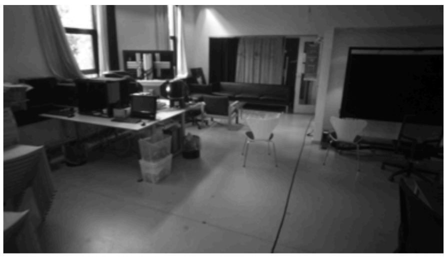
Figure 6.6: The left plot shows the relative error after 30 iterations. The initial standard deviation is dependent on the initial distance by the relationship described in 6.1. The right plot shows the density function of the inverse Gaussian distribution using the minimizer. The red lines indicate the 95 % confidence intervals and the blue line indicate the median.

to incorporate the relation described in Equation 6.1 in `EKFmonoSLAM`.

The next section will investigate how the initial depth distribution effects real data and how a good initial guess can be heuristically derived from image data.

6.3 Finding a good prior depth

The purpose of this section is to investigate the influence of the initial depth distribution on real data and to describe how a good prior can be found. These tests are performed on the indoor recordings from Experiment 6 described in Section 4.1. In this experiment the camera was manoeuvred in an ellipse where the size of the ellipse was known. The initial image in this recording is shown as the left image in Figure 6.7.



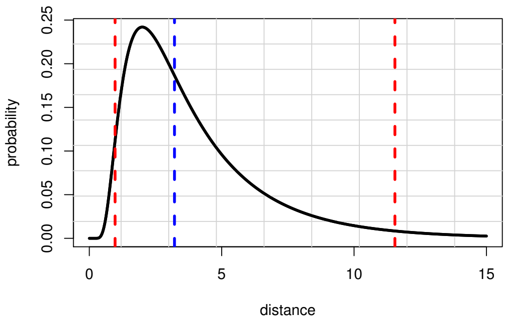


Figure 6.7: The left image shows the initial frame from the indoor recording from Experiment 6. The right plot shows the hypothesis of an initial inverse Gaussian depth distribution with $\hat{\rho}_0 = 0.125$ and $\sigma_p = 0.0625$.

Because of humans' prior knowledge about the size of the furniture in the image, we rather intuitively get an understanding of the depth in the image. Thus, from the image it is expected that the closest objects are about 1 meter from the camera and the most distant features are approximately 10 meters away from the camera. Also, it is expected that most of the objects such as the chairs, the desk, and the TV are approximately 4 meters from the camera. Using the correspondence between the mean and the shape of the inverse Gaussian distribution described in Equation 6.1 an initial hypothesis could be $\hat{\rho}_0 = 0.125$ and $\sigma_p = 0.0625$ which provide the inverse Gaussian distribution seen in the right plot in Figure 6.7. Note that both the confidence intervals and median coincides with the expectations about the depth. This hypothesis will be investigated in this section.

Figure 6.8 shows three subfigures which each illustrate the estimated path of 10 tests using an initial depth of respectively 1, 4, and 10 meters. The known true path is illustrated by the black dotted line.

From Figure 6.8 it is seen that the initial depth distribution has a huge influence on the estimate. In the top left subfigure a too small ellipse is estimated compared to

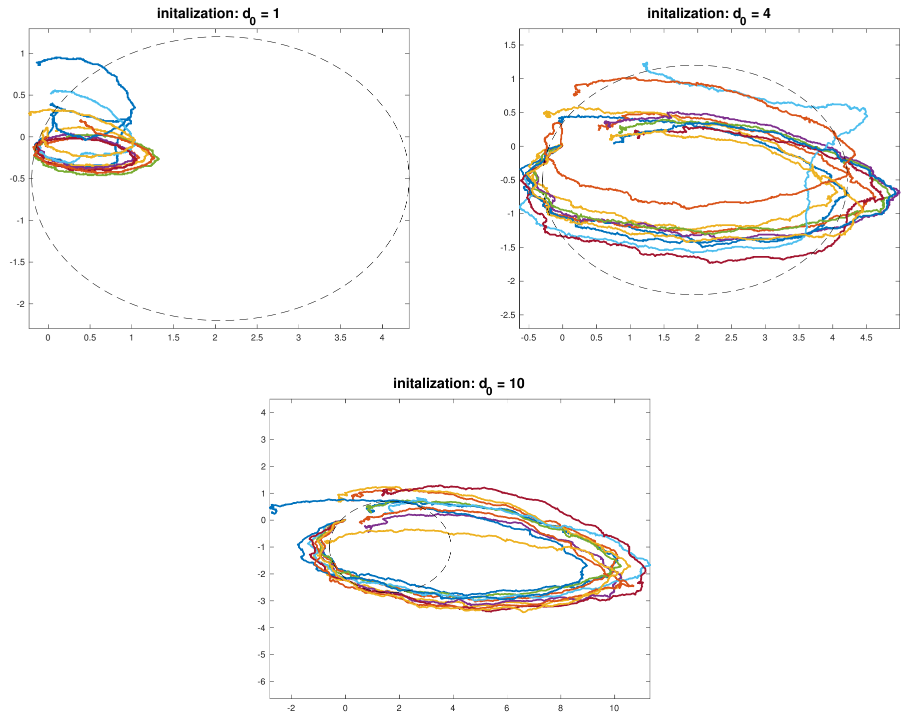


Figure 6.8: Each subfigure have 10 estimated path with the same initial depth distribution. The initial mean depth in the top left subfigure is $d_0 = 1$, in the top right subfigure it is $d_0 = 4$, and in the lower central subfigure it is $d_0 = 10$. From the figure it becomes very clear that the initial depth distribution has huge influence on the scaling. The true path is illustrated by the black dotted line and has the same size in all three subfigures

the true path (the black dotted ellipse) and in the lower central plot a too big elliptic path is estimated. This is caused by a choosing respectively a too short and too long initial distance. The path in the last figure, with an initial distance of $d_0 = 4$, seems to fit the true path well.

It is also seen from Figure 6.8 that different estimated paths are obtained each time `EKFmonoSLAM` is run even though both the data and the parameters do not change. The reason for this randomness lies in 1-Point RANSAC that randomly chooses a model with high support from data. Note that 1-Point RANSAC does not choose the best model, but merely a good model. This results in the random structures seen in the figure. It is also seen that the distances between the estimated paths propagate as time progresses. This is a common issue for many types of SLAM because the errors accumulate over time.

Furthermore, the absolute error in meters and the relative cumulative error is calculated for the $d_0 = 1, \dots, 10$. An average of six runs have been taken to reduce the influence of the randomness. The average estimated paths can be found in Appendix B.1 together with an explanation on how the error functions are calculated. Figure 6.9 shows these error functions.

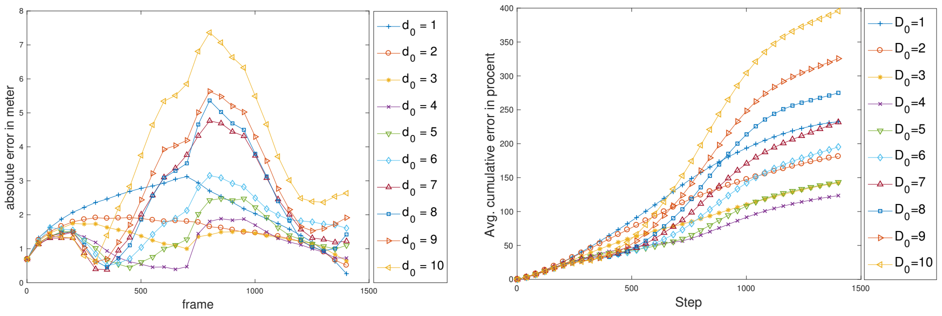


Figure 6.9: The right subfigure shows the absolute error in meters when choosing different initial depths. The left subfigure shows the relative cumulative error when choosing different initial depths. In both subfigures, each line is an average of six runs to reduce the influence of randomness.

From Figure 6.9 the same trend is seen as described in Figure 6.8: Using a too small prior depth results in a high error and choosing a too big prior depth also results in a high error. From Figure 6.9, it is seen that choosing $d_0 = 4$ provides the best estimate. Choosing $d_0 = 4$ provides a relative cumulative error of 125.4835%. From these observations it is seen that the `EKFmonoSLAM` is dependent on the initial depth distribution to provide the correct scaling. Also, it was found that a rather good initial guess of the scaling can be obtained by human intuition from looking at the images and by investigation of the inverse Gaussian distribution. However, even with a good initial guess very high cumulative errors are obtained.

As described in Section 6.1, two issues arose after testing `EKFmonoSLAM` on data collected wearing the glasses: The scaling was off and a lot of drift occurred in recordings with fast rotations. In this and the previous section the importance of the prior knowledge for proper scaling of the estimated path and environment have been established. It has been investigated how a good initial depth can be chosen to reduce the error. The second issue, the drifting that occurred with fast rotation, will be investigated in the following sections. When the filter is estimated on basis of just a few feature correspondences, the estimated model will be more sensitive to outliers and noisy observations. During fast turns the images are often more blurred and therefore the features are associated with higher error. Previously, in Section 5.4, it was found that significantly less features were found in images with rotation. Furthermore, it is hypothesized that even less features are matched during fast turns. This hypothesis will be tested by investigating the matching algorithm NCC with Active Search in the following two sections.

6.4 Investigation of thresholds for Active Search

In this section the search region in Active Search is investigated. The purpose is to determine if Active Search with a threshold of 100 limits the number of correctly matches features.

As described in Section 3.8, the advantage with Active Search is that only feature matches within a limited search region are considered. Figure 6.10 illustrates which features are excluded because of a too high uncertainty (red), which features are included but without finding a match with NCC (yellow), and which features that are included and where a match has been found (green).

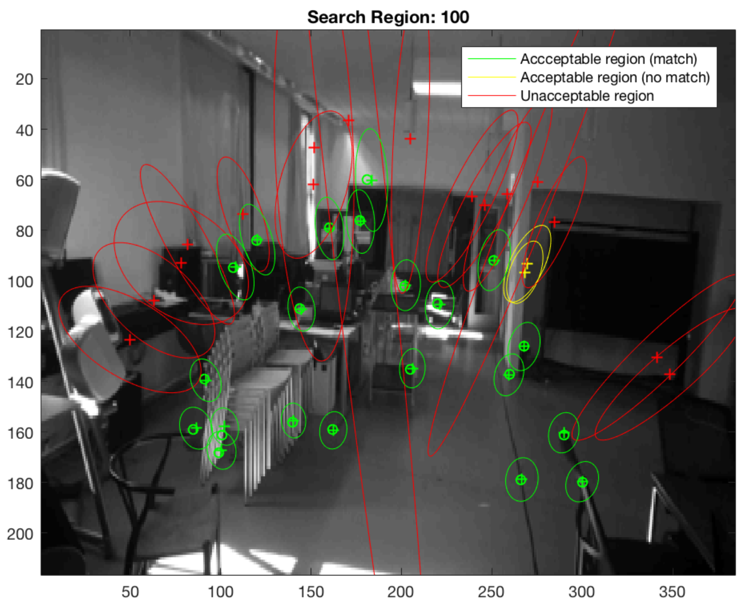


Figure 6.10: The figure shows the unacceptable search region (red), the acceptable search region where no match has been found by NCC (yellow), and the acceptable search region where a match has been found (green). The default threshold of 100 has been chosen. Note that very little rotation is happening in this frame.

The figure shows that features with high uncertainty (big ellipses) are not tried being matched. There is only little rotation between this image and the previous, thus most of the feature uncertainty is rather small. It is seen that NCC finds a match in nearly all the included search regions. And it seems as a sufficient number of features are matched. Thus, the search region seems reasonable in this setting. However, when looking at a sequence of blurred images during a turn (Figure 6.11) the uncertainty increases and much less features are matched.

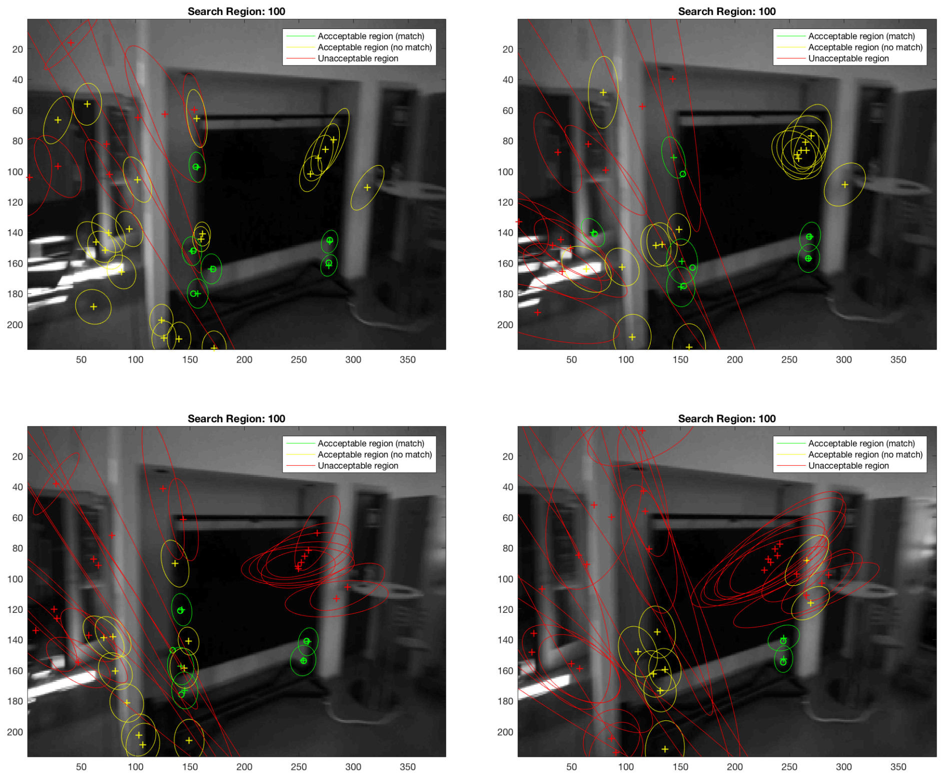


Figure 6.11: The figure shows four sequential images in a turn. Features within the unacceptable search regions are colored red, features within acceptable search regions where no match has been found by NCC are colored yellow, and the features within acceptable search region where a match has been found are colored green. The default threshold of 100 has been chosen.

From Figure 6.11 it is seen that in many of the acceptable regions NCC is not able to find any matches. A reason for this is that NCC is not rotation invariant, thus the rotation between the images reduce the odds of getting a true match by NCC. From this figure, it is also seen that the uncertainty grows for features that are not matched over a series of images. E.g. the features in the top right corner of the TV are not being matched in any of the images and thus the uncertainty of their position grows. In the lower left image in Figure 6.11, the uncertainty of these features have become too big, thus they are exclude by the Active Search algorithm. From these points it is also seen that the uncertainty is greater in the horizontal direction which corresponds to the turning direction.

From Figure 6.11 it is seen that there are many acceptable search regions where a match has not been found. Based on these observations, it is assessed that NCC is the limiting factor to achieve more correct matches during turns. Therefore, the threshold that determine the size of an acceptable search region is kept as the value suggested by Civera et al. in the `EKFmonoSLAM` implementation[JM08].

This section has tested the influence of the size of the search region of Active Search on data from the Tobii Pro Glasses 2. It is found that the NCC is the limiting factor to achieve more correct matches especially between images with high rotation. Therefore, it is chosen to keep a search region of 100. In the following section the robustness of NCC will be examined.

6.5 Robustness of matching

When there is fast rotation between images it is important to have rotation invariant features to obtain good matches. When walking around wearing the glasses, the camera movement and rotation will be rapid, because the person wearing the glasses tend to make quick head movements. It is known that NCC is neither scale nor rotation invariant [Aan15]. However, in this section, it is tested how much rotation this NCC implementation can handle.

NCC is not rotation invariant because it is calculated based on a patch from the feature (Section 3.8). If the image is rotated, the pixels in this patch will not be the same nor will they be in the same order, and thus lower correlation will be found between the two patches.

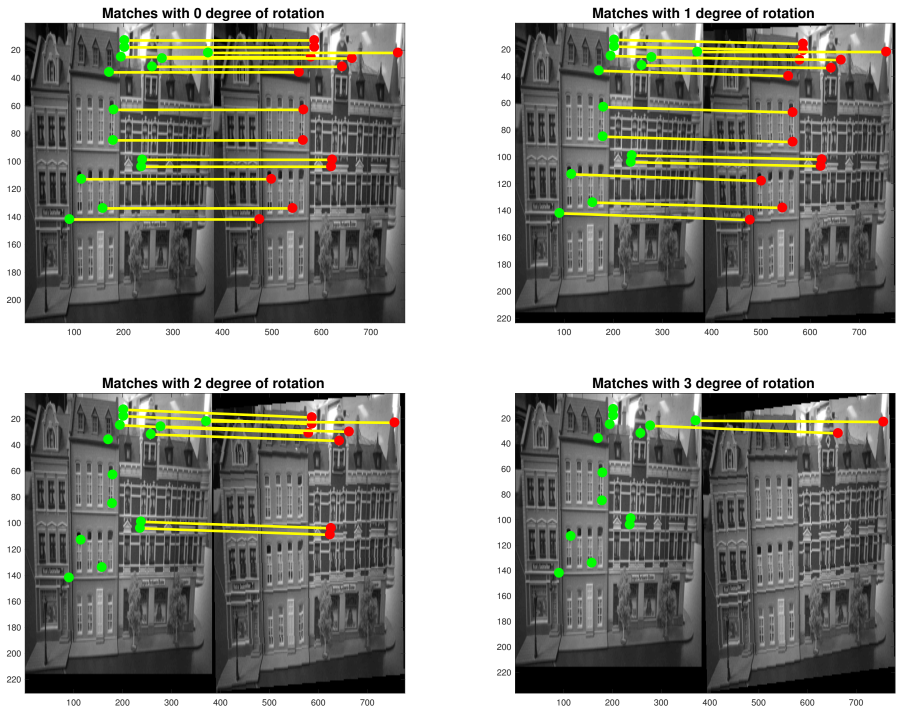


Figure 6.12: The figures shows the number of features that are matched when rotating an image with respectively 0,1,2, and 3 degrees. From the figure, it is is found that the NCC is very sensitive to rotation.

In this scenario, it is chosen to use a threshold of 100 for the feature detectors to provide an easier and better illustration. The detected features are matched with features in a rotated version of the same image. This matching is repeated with increased rotation in Figure 6.12.

From Figure 6.12 it was found that the NCC is very sensitive to rotation. A rotation with just 3 degrees removed nearly every match. Additionally, it was found that with these settings and this data, a rotation of 15 degrees removed all matches.

In this section it was confirmed that NCC is very sensitive to rotation and only a few degrees of rotation can mess up the matching. This is an issue when using NCC for feature matching between images with high rotation which is typically the case for a head mounted camera.

The second issue that arose in Section 6.1 when the glasses were head-worn was that a drift occurred in the estimate path in recordings with a lot of rotation. The assumption was that bad or few feature correspondences between frames with high rotation caused this error. Already in Section 5.4 it was found that significantly fewer features were discovered by FAST-9 during turns. Furthermore, it has been found in this section that NCC dramatically limits the number of matches when exposed to just a few degrees of rotation. The combination of the poor performance of FAST-9 and NCC when exposed to rotation results in very few feature matches during turns. The model estimate becomes sensitive to outliers when it is based on a very little sample. Thus, these few matches are likely to cause the drift of the estimated path observed in Section 6.1.

Previously it was mentioned that close features are good at estimating translation and distant features are good at estimating rotation. Thus, it is hypothesized that `EKFmonoSLAM` will be less sensitive to rotation if more distant features are included and that the relative drift would be smaller in a bigger scale. Thus, it was decided to test how well the `EKFmonoSLAM` behaved in an outdoor experiment with bigger scale and thus more distant features.

6.6 Increased distances

This section presents the results form the outdoor experiment 6 (Section 4.1). The purpose of this experiment was to test how well the `EKFmonoSLAM` performed in an outdoor setting where the scale was significantly larger than in the indoor experiments. Figure 6.13 shows five heuristically derived inverse Gaussian distributions that describe the depth of the features.

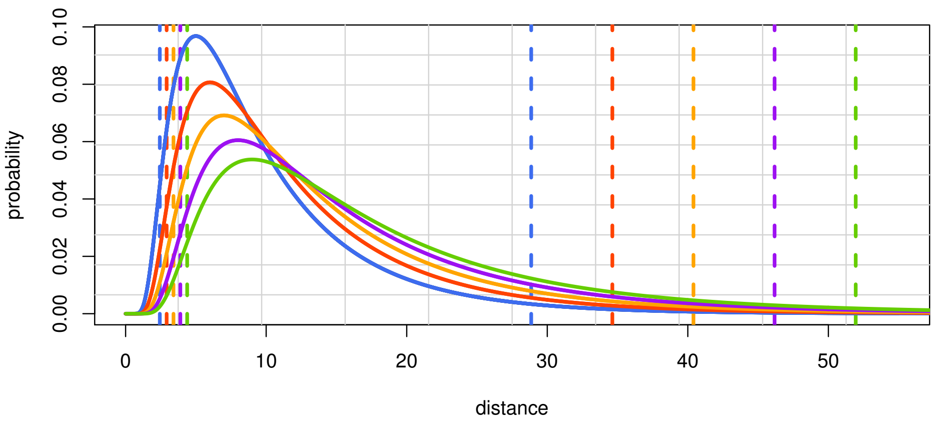


Figure 6.13: The figure shows the five heuristically derived inverse Gaussian distribution which will be tested in the outdoor experiment. d_0 is respectively 10, 12, 14, 16, 18 meters.

Figure 6.14 shows the estimated paths for the outdoor experiment with the initial inverse depth distributions shown in Figure 6.13. In the experiment, I walked around an basketball court. The total distances covered is estimated to 104 meters based on GPS data.
It is important to note that the that correspondences between the estimate satellite image, the GPS data, and the estimate paths are unknown. The scale has been estimated by dividing the length and width of the basketball court measured in pixels on the image with the length and width of the court measured in meters via Google Maps. The rotation and off-set are heuristically derived to get a reasonable fit. Furthermore, also the GPS data is associated with errors up to 5 meters[Cor]. However, in order to evaluate the accuracy of the estimated paths, it is assumed that these estimations of the scale and rotation are precise enough to use the GPS path as the

true path.

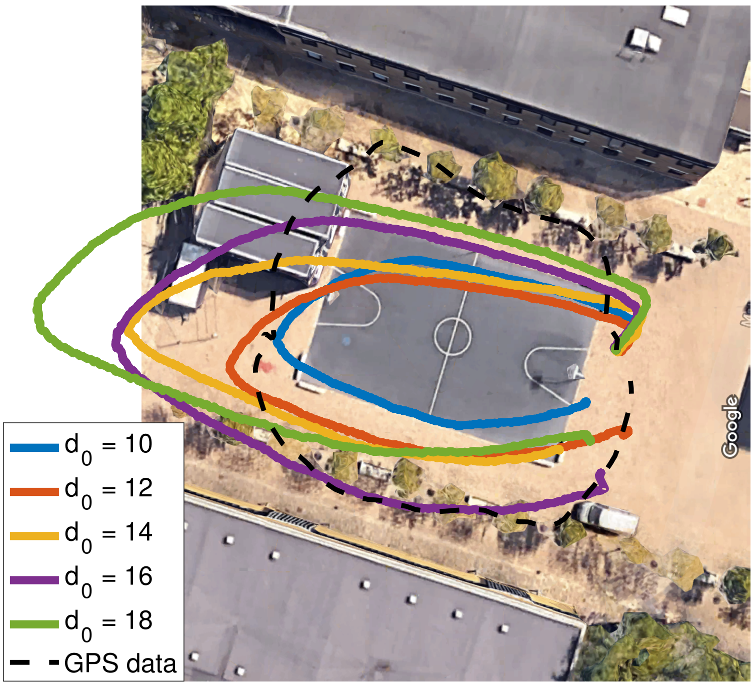


Figure 6.14: The figure shows the estimated paths of the outdoor experiments with different initial paths. It is seen that a higher d_0 result in longer estimated distances. The background image is from Google Maps[Goo].

From Figure 6.14 it is seen that increasing d_0 will increase the length of the estimate path. Based on the figure, it seems that either the estimated paths with $d_0 = 10$ or $d_0 = 12$ provides the best scaling.
It is expected that the path of the GPS and the estimate paths should be very similar to begin with. However from the figure, it is seen that this is not the case. Thus either the chosen rotation is imprecise or the GPS path is associate with high errors in the beginning.
Furthermore, it is seen from the figure that the estimated paths seems be more narrow (higher ratio between the major and the minor axis) than the GPS path. This

trend is supported by the left subplot in Figure 6.15 that shows the absolute errors in meters for each frame. From this figure it is seen that two maxima are found at frame 750 and 2250 corresponding to the two more narrow sides.

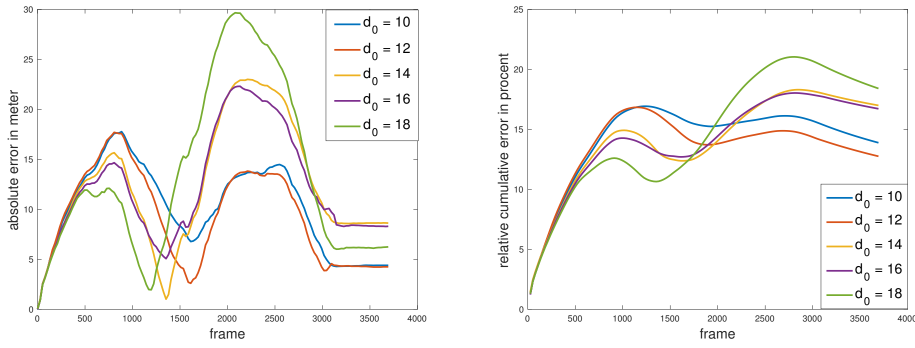


Figure 6.15: The left subfigure shows the absolute error measured calculated as the 2-norm distances between the GPS-data and each of the estimated paths. The right subfigure shows the cumulative error relative to the distance of the estimated path[1].

From Figure 6.15, it is seen that estimated path with the initial inverse Gaussian distribution with $d_0 = 12$ has the lowest relative cumulative error. It has a relative cumulative error of 13% over a distance of 104 meters. This error is a significantly lower than the error in the indoor experiment. This is properly due to the larger scale of this experiment. In this experiment, `EKFmonoSLAM` had long distances with walking in a straight line (which are rather easy to estimate), and it had more distant features, which made it easier for the system to estimate the rotation.
However, even by choosing a good initial inverse Gaussian distribution to describe the depth of the features, the estimates are not accurate enough to be useful for consumer analysis.

In this chapter it was decided to test how well `EKFmonoSLAM` performed with more challenging data from a person wearing the glasses. In this investigation two issues occurred: The scaling was incorrect and the software had difficulties handling fast rotations.
It is only possible to determine the distance to features up to scale from a single image. Therefore, Civera et al. use an inverse Gaussian distribution as a prior knowledge when initializing features. The importance of this initial distribution was investigated.

[1]The GPS data consist of 143 observations whereas the estimated paths consist of 3420 frames. Thus, the error is calculated as the 2-norm between every GPS observation and the estimate from every 3420/143 = 24 frame.

It was found that a good prior knowledge is required in order to obtain the correct scaling. Furthermore, it was found that the standard deviation could advantageously be dependent on the depth, and that the depth could be estimated rather accurately by human intuition. The relative cumulative error of the indoor experiment was found to 125 %. Based on this high error it was concluded that more research and development has to be done in order for `EKFmonoSLAM` to solve the SLAM problem for the Tobii Pro Glasses 2. Furthermore, providing the EKF with an prior distribution to initialize features have several drawbacks. One being that it is expected that the distance to new features follow the same distribution throughout the entire recording. Thus, this approach would be very unlikely to work well in a recording that started indoor and ended outdoor. Also, this approach require a lot of fine tuning and a good prior knowledge of the recording.
Secondly, it was investigated why the `EKFmonoSLAM` had difficulties handling fast rotations. It was found that NCC is the limiting factor to obtain more matches and that this algorithm is very sensitive to rotation. A rotation of just a 3 degrees removed nearly all matches in a non blurred image. This could be dealt with by using scale and rotation invariant features such as SIFT or SURF. These algorithms are computational expensive which is why the are not used in `EKFmonoSLAM`. However, it could be interesting to use SIFT features in every 10th or 20th frame as a form of a more stable anchor points.
Finally, it was tested how well the `EKFmonoSLAM` did with data from an outdoor recording with more distant features. Again, it was found that the prior knowledge of the depth is important to obtain an accurate scaling. It was found that choosing a good prior initial inverse depth distribution led to rather good path estimations. In this test, the relative cumulative error was found to 13 %.

The next section will come with some suggestions for future work. In this section, emphasis is put on sensor fusion.

CHAPTER 7

Future work

Building on the results of the thesis this chapter points to future work that possibly could improve the accuracy and robustness of the implementation, while reducing the importance of manually selecting a good prior to obtain the correct scale. In this chapter a great emphasis is put on sensor fusion.

It was found that the `EKFmonoSLAM` implementation had to be given a prior initial inverse Gaussian distribution to estimate the correct scale and that the implementation was sensitive towards fast rotations. These two issues could potentially be coped with by using the sensor data from the gyroscope and the accelerometer in the Tobii Pro Glasses 2. Appendix F.1 provides a manual for extracting the accelerometer and gyroscope data in an workable format.

As mentioned, it was found that `EKFmonoSLAM` had a tendency to drift during fast turns. The gyroscope measures the angular velocity, thus using the gyroscope data it is expected to obtain better measurements and estimations during fast turns.
To incorporate the gyroscope data into the EKF only the measurement equation needs to be changed as the angular velocity is already modelled in the state space representation. In appendix F.2, there is a theoretical outline of how this could be done. This approach has already been integrated into the implementation and the results so far are presented in the same appendix. However, based on these results there are still need of significant parameter optimization in order to make it work in practice.

The accelerometer could be used to give an indication of the scales. The person wearing the glasses would be more likely to have high acceleration if he or she is not in risk of bumping into objects. Thus the scale would be rather big. On the other hand, if the person has little acceleration, it is believed that the scale is rather small.
To incorporate the accelerometer data into the EKF, the acceleration has to be modelled, thus the constant velocity model currently used should be replaced by a constant acceleration model. Furthermore, both the transition operator and the measurement operator have to be changed. This process has theoretical been explained in Appendix F.3. However, it has not been implemented, adapted nor have the parameters been adjusted in order to make the model work in practice.

This chapter has outlined some possible and promising improvements of the `EKFmonoSLAM` to provide higher accuracy when using the Tobii Pro Glasses 2. The next chapter will conclude on the findings in the thesis.

CHAPTER 8

Conclusion

The object of this thesis was to investigate if there are any published open source SLAM project, which - directly or with modifications - can solve the SLAM problem for the Tobii Pro Glasses 2. If so, this implementation could be useful to study consumer behavior in many different settings.

Considering 34 published SLAM projects, it was found that the `EKFmonoSLAM` implementation was the most adequate open source project to the Tobii Pro Glasses 2. The principal arguments for choosing this implementation were the use of an Extended Kalman Filter, which provided the opportunity to use multiple sensors, the well-acknowledged authors, the 3D map modelling, and the fact that it was implemented in Matlab.

The `EKFmonoSLAM` implementation was adapted to the Tobii Pro Glasses 2 by lowering the thresholds for the feature detector to 20, decreasing the image size to 384x216, and calibrating the front camera of the glasses. The implementation was validated by successfully replicating the results obtained by Civera et al. in experiments where the glasses were hand-held.

When testing how well the implementation performed when the glasses were head-worn, it was found that two issues occurred: First, the implementation could not determine the scale of the map without prior knowledge about the depths of the landmarks. Second, a drift occurred in experiments with fast rotations.
As the features were modelled in inverse depth coordinates, an inverse Gaussian distribution was used as a prior to model the depth of the features. It was found that the standard deviation of this distribution could advantageously depend on the depth. Furthermore, it was found that a good prior depth can be obtained by manually studying the recording and investigating the inverse Gaussian distribution.
It was found that the drift was due to few and erroneous feature correspondences during turns. It was established that Active Search was not the limiting factor to obtain more feature matches. However, it was found that the non-rotation invariant NCC was very sensitive to rotation and thus the number of matches was dramatically reduced with just three degrees of rotation.
Conclusively, it was found that by manually choosing a good prior distribution, the relative cumulative error in a 12 meter indoor experiment could be reduced to 125%

which is far too much to be useful in any practical setting. However, better results were obtained in a 104 meters outdoor experiment where the relative cumulative error could be reduced to 13%, which is much better, but not useful for studies of consumer behavior, which mostly takes place indoors.

Thus, based on this empirical investigation, which includes extensive data-gathering, this thesis concludes that even the apparently best open-source SLAM project does not give sufficient precision to provide useful information about consumer behavior in an indoor setting. Further research and development is necessary in order to solve the SLAM problem for the Tobii Pro Glasses 2.

APPENDIX A
Comparison of 34 open-source SLAM implementations

Table A.1: The table shows a description of all 34 of the open source SLAM projects available at OpenSLAM.org. Key stats as which algorithm is used, which requirements, 3-D support etc. are listed. Furthermore, a short description of the projects are also included. Most of the information is collected from [Bac17].

	Hardware and software requirements	Algorithm	Author	Input Data	Type of map	Dato	Description	3D support
2D-I-SLSJF	Matlab	Iterated Sparse Local Submap Joining Filter (SLSJF)	Shoudong Huang and Zhan Wang	A sequence of small loal maps, each local map contain a state vector estimate and the corresponding covariance matrix.	point feature maps	2009	I-SLSJF is a local submap joining algorithm using sparse information filter and least squares optimization. The input of I-SLSJF is a sequence of local maps and the output of I-SLSJF is a global map. The preprocessed Victoria Park data set and the corrected DLR-Spatial-Cognition data set are provided to demonstrate the algorithm.	No
CAS-Toolbox	Matlab	EKF	Kai O. Arras	Sensor and odometry data	Feature maps	2007	This software is a GNU GPL licenced Matlab toolbox for robot localization and mapping. It is made for research and education and independent on the type(s) of feature and type(s) of sensors/ It can import a number of data file formats from any sensor. It allows you to plug in and out your feature extraction, odometry model, data association strategy, etc. and to plug in and out your SLAM or Localization approach. It furthermore comes with a number of useful tools and functions.	No
CEKF-SLAM	MATLAB	Compressed Extended Kalman Filter	Haiqiang Zhang and Lihua Dou	Sensor and odometry data	Feature maps	2007	CEKF-SLAM was originally proposed by Jose Guivant and Eduardo Net. This algorithm reduces the computational complexity by dividing the system state vector into two parts: the active local state vector and the others. Only the local state vector is updated for each step by EKF, and the necessary information for updating the other states is compressed into some auxiliary cofficient matrices. When the local area changes, a full update was executed to get the same estimation results as EKF.	No
COP-SLAM	C++ and Matlab	g2o	Gijs Dubbelman;Brett Browning;	COP-SLAM takes as input g2o pose graph files, which specify nodes and edges of a pose graph. Example data sets with a total length of 60 kilometers are provided with the demo program.	Pose-chains	2012	COP-SLAM is a highly efficient closed-form 3-D SLAM approach, It optimizes pose-chains on-line and it is compatible with g2o. The COP-SLAM demo program comes with 60 kilometers of pose-chain datasets, which are obtained with visual odometry and appearance-based loop detection.	Yes
DP-SLAM	Linux	efficiently maintaining a joint distribution over robot maps and poses.	Austin Eliazar;Ronald Parr;	The approach takes raw laser range data and odometry.	grid maps	2007	DP-SLAM aims to achieve truly simultaneous localization and mapping without landmarks. While DP-SLAM is compatible with techniques that correct maps when a loop is closed, we have found that DP-SLAM is accurate enough that no special loop closing techniqu	

EKFMonoSLAM	Matlab	EKF	Javier Civera;J. M. M. Montiel;	monocular image sequence and its camera calibration	A sparse 3D map of salient point features	2010	EKFmonocularSLAM contains Matlab code for EKF SLAM from a 6 DOF motion monocular image sequence. The algorithm takes as input a monocular image sequence and its camera calibration and outputs the estimated camera motion and a sparse map of salient point features. The code includes state-of-the-art contributions to EKF SLAM from a monocular camera: inverse depth parametrization for 3D points and efficient 1-point RANSAC for spurious rejection.	Yes
FalkoLib	C++	?	Fabjan Kallasi;Dario Lodi Rizzini;Stefano Caselli;	The approach takes raw laser range data. Odometry can be used for testing the association between features.	feature maps	2016	FALKOLib is a library containing keypoint detectors for the stable detection of interest points in laser measurements and two descriptors for robust associations.	No
FLIRTLib	C++	Fast Laser Interest Region Transform	Gian Diego Tipaldi;Kai O. Arras;	?	?	2010	FLIRTLib is a data association library to be used as a front end for graph based mapping, using RANSAC and multi-scale interest point. It implements four different multi-scale feature detectors and two feature descriptors for 2D range data. It is written in C++ and provides an API reference (written using Doxygen) and a set of example binaries to visualize the detector results and the descriptors, as well as perform scan to scan matching.	No
G2O	C++	Graph Optimization	Rainer Kuemmerle;Giorgio Grisetti;Hauke Strasdat;Kurt Konolige;Wolfram Burgard;	Nodes and edges of a graph.	Graphs (nodes and edge)	2011	g2o is an open-source C++ framework for optimizing graph-based nonlinear error functions. g2o has been designed to be easily extensible to a wide range of problems and a new problem typically can be specified in a few lines of code. The current implementation provides solutions to several variants of SLAM and BA.	Yes
GMapping	C++	A Rao-Blackwelliz particle filter	Giorgio Grisetti;Cyrill Stachniss;Wolfram Burgard;	laser range data	grid maps	2007	GMapping is a highly efficient Rao-Blackwellized particle filer to learn grid maps from laser range data.	Yes
GridSLAM	C++	A Rao-Blackwelliz particle filter	Dirk Haehnel;Dieter Fox;Wolfram Burgard;Sebastian Thrun;	laser range data	grid maps	2003	GridSLAM is an easy to use and understand Rao-Blackwellized particle filer to learn grid maps from laser range data.	
HOG-Man	C++	Graph Optimization	Giorgio Grisetti;Rainer Kuemmerle;Cyrill Stachniss;	Nodes and edges of a graph.	Graphs	2010	HOG-Man is an optimization approach for graph-based SLAM. It provides a highly efficient error minimization procedure that considers the the underlying space is a manifold and not a Euclidian space. It furthermore generates a hierarchy of pose-graphs which is used perform the operations during online mapping in a highly efficient way. The approach works in 2D and 3D.	Yes
iSAM	C++	Graph Optimization	Michael Kaess;Hordur Johannsson;John Leonard;	Factor graph: Nodes (variables) and factors (measurements)	Feature-based or pose graph	2009	iSAM is a general optimization library for incremental sparse nonlinear problems as encountered in simultaneous localization and mapping (SLAM).	Yes
Linear SLAM	C++ and Matlab	Graph Optimization	Liang Zhao;Shoudong Huang;Gamini Dissanayake;	The input to the Linear SLAM algorithm is a sequence of local submaps. Each local map contains a state vector estimate and the corresponding information matrix.	Pose feature or pose graph map	2013	Linear SLAM: A Linear Solution to the Feature-based, Pose Graph and D-SLAM based on Submap Joining.	Yes

Max-Mixture	C++	Graph Optimization	Pratik Agarwal;Edwin Olson;Wolfram Burgard;	Nodes and edges of a graph.	Graphs (nodes and edge)	2013	Max-mixture allows handling large number of outliers and multimodal constraints in the least square SLAM formulation. The code here is implemented as a plugin for g2o.	yes
MTK	Matlab	Graph Optimization	Christoph Hertzberg;Rene Wagner;Oliver Birbach;	Input operations have to be implemented by the user.	MTK is only used to represent states. SLoM can operate on arbitrary feature based maps or calibration problems.		MTK is a toolkit that provides easy mechanisms to enable arbitrary algorithms to operate on manifolds. The main application is the use of 3D rotations SO(3), as well as the construction of compound manifolds from arbitrary combinations of sub-manifolds.	
OpenRatSLAM	C++	Graph Optimization	Michael Milford (algorithm);Gordon Wyeth (algorithm);David Ball (code);Scott Heath (code);	OpenRatSLAM takes mono images and odometry as standard ROS messages. The images can be forward facing or omnidirectional and can be high or low quality.	topological maps	2013	RatSLAM is a robot navigation and SLAM system based on computational models of the hippocampus. The approach uses a combination of appearance based visual scene matching, competitive attractor networks, and a semi-metric topological map representation. The approach has been proven in a real tiem 40 hour robot delivery task, mapping an entire Australian suburb and on Oxford's New College dataset. Notably, RatSLAM works well on images obtained from cheap cameras. The RatSLAM system contrasts many of the other SLAM approaches that involve expensive precision laser sensors and occupancy grids.	Yes.
ORB-SLAM	ROS	Graph Optimization	Raul Mur-Artal;J. M. M. Montiel;Juan D. Tardos;	Monochrome/Color Images	Sparse 3D points, keyFrame poses and covisibility graph	2015	ORB-SLAM is a keyframe and feature-based Monocular SLAM. It operates in real-time in large environments, being able to close loops and perform camera relocalisation from very different viewpoints.	Yes
OpenSeqSLAM	Matlab	Sequence-based algorithm	Niko Suenderhauf;	Images	Graphs	2012	OpenSeqSLAM is an open source Matlab implementation of the original SeqSLAM algorithm published by Milford and Wyeth at ICRA12. SeqSLAM performs place recognition by matching sequences of images.	Yes
ParallaxBA	C++	Graph Optimization	Liang Zhao;Shoudong Huang; Yanbiao Sun;Gamini Dissanayake;	The input of ParallaxBA is the matched image points, camera calibration matrix, as well as the initial values of camera poses and features (optional).	3D point features and camera poses		ParallaxBA: Bundle Adjustment using Parallax Angle Feature Parametrization.	
Pkg. of T.Bailey	Matlab	EFK, UKF, FastSLAM 1, and FastSLAM2.	Tim Bailey;		feature maps	2007	This package is a collection of implemented SLAM approaches by Tim Bailey. The code is written in MatLab and performs EFK, UKF, FastSLAM 1, and FastSLAM2.	

RGBDSlam	ROS Diamond-back and HOG-Man.	Graph Optimization	Felix Endres;Juergen Hess; Nikolas Engelhard;Juergen Sturm;Wolfram Burgard;	Monochrome and depth image, colored point cloud	Pose graph with colored point clouds	2012	RGBDSLAM allows to quickly acquire colored 3D models of objects and indoor scenes with a hand-held Kinect-style camera. It provides a SLAM front-end based on visual features s.a. SURF or SIFT to match pairs of acquired images, and uses RANSAC to robustly estimate the 3D transformation between them. The resulting camera pose graph is then optimized with the SLAM back-end HOG-Man.	Yes
Robomap Studio	Matlab	Monto Carlo	Jerry Moravec;	2DLS - sensorial data	Line maps	2011	ROBOMAP Studio is a set of useful programs to processing data from 2DLS, mainly focused to continual and global localization and SLAM.	No.
RobotVision	C++	Graph Optimization	Hauke Strasdat;Steven Lovegrove;Andrew J. Davison;	RobotVision is primarily designed as a library, not as a standalone application. However, it comes with some demo applications.	Feature maps and pose graphs	2010	RobotVision is a library for techniques used on the intersection of robotics and vision. The main focus is visual monocular SLAM. It is written in C++ – partially using object-oriented and template meta programming. Thus, most techniques can be easily adapted to other applications - e.g. range-and-bearing SLAM.	Yes.
ro-slam	C++	Monto Carlo	Jose Luis Blanco;Juan Antonio Fernandez Madrigal;Javier Gonzalez Jimenez;	Pairs of action (movements from odometry) + observation (sensed ranges to a set of static beacons).	Landmarks	2008	This is a C++ implementation for generic RBPF-SLAM with different kinds of maps, including one solution to Range-Only SLAM (RO-SLAM) with landmark maps represented as Sum of Gaussians (SOGs), which are dynamically adapted to represent well the uncertainty of all mapped beacons. There exists a stand-alone executable ready for use and demo configuration files and datasets.	Yes
SLAM6D	C++		Andreas Nuechter; Kai Lingemann; Jochen Sprickerhof; Dorit Borrmann; Jan Elseberg; Peter Schneider; Deyuan Qui;	3D scan data in various file formats.	3D point clouds	2009	This project consists of a software to register 3D point clouds into a common coordinate system, as well as a viewer to display the scene. For the registration, different ICP minimizing algorithms can be chosen, as well as global relaxation methods, aiming at generating an overall globally consistent scene. Several formats for the point clouds are supported, new formats can be implemented easily.	Yes
SLOM	Out of date	Out of date	Out of date	Out of date	Out of date	2008	The SLoM-Framework provides a framework to optimize arbitrary Least-Square Problems on Manifolds. This version of SLoM is out-dated and merely provided for historic reasons. SLoM is now part of the Manifold ToolKit (MTK) which is also available on OpenSLAM.	Out of date
SSA2D	C++	Graph Optimization	Michael Ruhnke;Rainer Kuemmerle;Giorgio Grisetti;Wolfram Burgard;	Nodes and edges of a graph.	Graphs (nodes and edge)	2011	SSA is an open-source C++ tool for post optimization of graph-based 2D SLAM solutions. SSA iteratively refines robot poses and 2D surface points in one global graph optimization system and produces highly accurate 2D laser maps. Laser scans are not treated as rigid body and might be refined during the optimization procedure. This leads to substantially less accumulated noise in the resulting map. SSA uses g2o as optimization back-end.	No.
tinySLAM	C	A particle filter	Bruno Steux;Oussama El Hamzaoui;	The approach takes raw laser range data and odometry.	Grid maps	2010	tinySLAM is Laser-SLAM algorithm which has been programmed in less than 200 lines of C-language code.	No.
TJTF for SLAM	Java	thin junction tree filters	Mark A. Paskin;	feature maps	grid maps	2007	This software package implements a filtering technique that maintains a tractable approximation of the belief state as a thin junction tree. The junction tree grows under filter updates and is periodically “thinned” via efficient maximum likelihood projections so inference remains tractable. When applied to the SLAM problem, these thin junction tree filters have a linear-space belief state and a linear-time filtering operation. Further approximation yields a filtering operation that is often constant-time.	No.

TORO	C++	Graph Optimization	Giorgio Grisetti;Cyrill Stachniss;Slawomir Grzonka;Wolfram Burgard;	Nodes and edges of a graph.	Graphs (nodes and edge)	2009	TORO is an optimization approach for constraint-network. It provides an efficient, gradient descent-based error minimization procedure. There is a 2D and a 3D version of TORO available.	Yes.
TreeMap	C++	Graph Optimization	Udo Frese;	The input to treemap are measurements with covariance and known data-association	The resulting map is a vector of feature positions (2D/3D feature based SLAM) or robot poses (2D/3DOF pose relation SLAM).	2007	Treemap is an algorithm for feature based Gaussian SLAM. Actually it is an algorithm for incremental probabilistic inference in a high dimensional Gaussian defined as the product of many low dimensional Gaussians (incremental least square). Treemap can handle different variants of SLAM. Everything, that's specific to a SLAM variant or even to SLAM as a problem is contained in a small driver layer that can be adapted by the user.	Yes.
UFastSLAM	Matlab	A Rap-Backwellize particle filter	Chanki Kim;	laser range and odometry information	grid maps	2011	Unscented fastslam is a Rao-Backwellized unscented particle filter that uses the unscented filter for both the localization and mapping.	No
Vertigo	C++	Graph Optimization	Niko Suenderhauf;	nodes and edges of a pose graph	pose graphs	2012	Vertigo is a C++ extension for g2o and gtam. It provides an implementation of switchable constraints and enables g2o and gtsam to solve pose graph SLAM problems despite the presence of false positive loop closure constraints.	Yes

APPENDIX B

Extended comments & remarks

The purpose of this appendix is to improve the readers understanding of some concepts by giving more elaborate explanations than it was assessed appropriate in the main text.

B.1 Basis of the calculated error functions

The purpose of this section is to explain how the error function is calculated. Each line in Figure B.1 shows an average of 6 runs using different initial inverse Gaussian distributions.

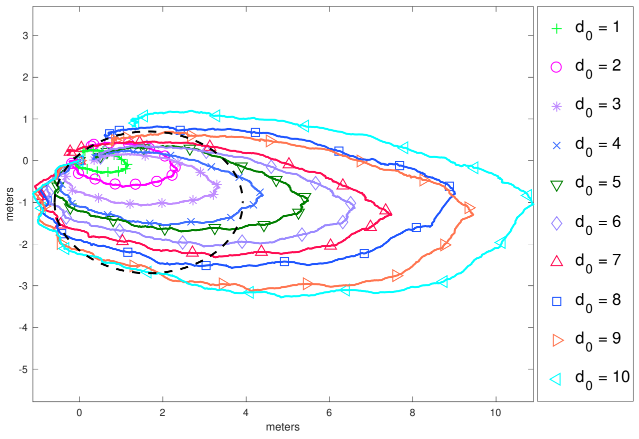


Figure B.1: Each line in the figure shows an average of 6 runs of the `EKFmonoSLAM` with different thresholds.

From Figure B.1 it is seen that the biggest d_0 estimates the biggest path and that the smallest d_0 estimates the smallest path.

The error function is calculated by measuring the distance to the the ellipse (the black dotted line) for each observed point. Thus, the true path is divided into equally distanced points, and the distance can thus be calculated as the 2-norm between these points and the observed points. In doing so, it is assumed that the estimated path has constant velocity. This is not the case, however it is assessed that the calculated error function gives a good approximation of the true error function, and thus is adequate to use as an evaluation of the accuracy.

B.2 Inverse Gaussian distribution

The section strives to give the reader a more intuitive understanding of the inverse Gaussian distribution used to initialize inverse depth coordinates. The inverse Gaussian distribution takes two parameters: The mean and the shape. In Figure B.2 the inverse Gaussian distribution is seen with a fixed shape and varying means.

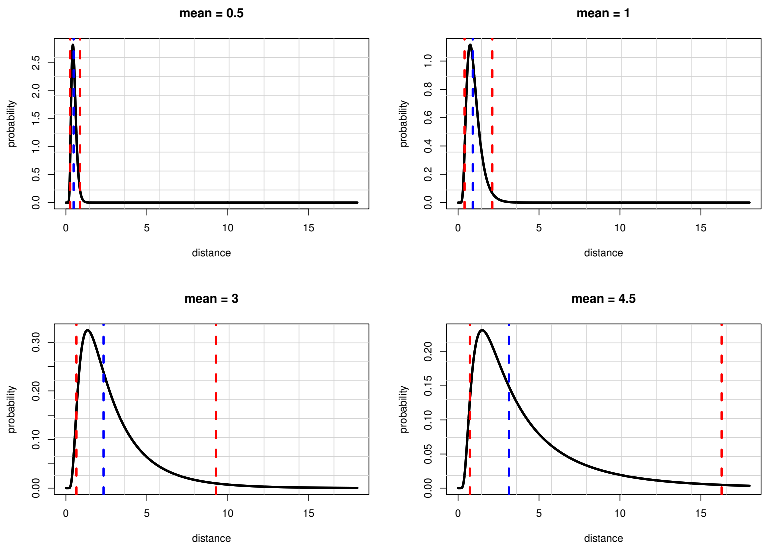


Figure B.2: The figure shows the inverse Gaussian distribution for different means and a fixed shape. The red lines indicate the 95 % confidence intervals whereas the blue line indicate the median.

From Figure B.2 it is seen that the distribution rises to its peak and the goes to zeros slowly. It has a long tail. In practice low parallax features have a high uncertainty (variance). This means that its inverse Gaussian distribution will be very wide and its tail very long which makes its very difficult to determine the depth of a low parallax features. Also, it is seen from Figure B.2 that increasing the mean allows for more distant features as the confidence intervals grows wider.

Figure B.3 shows how the shape effects the distribution.

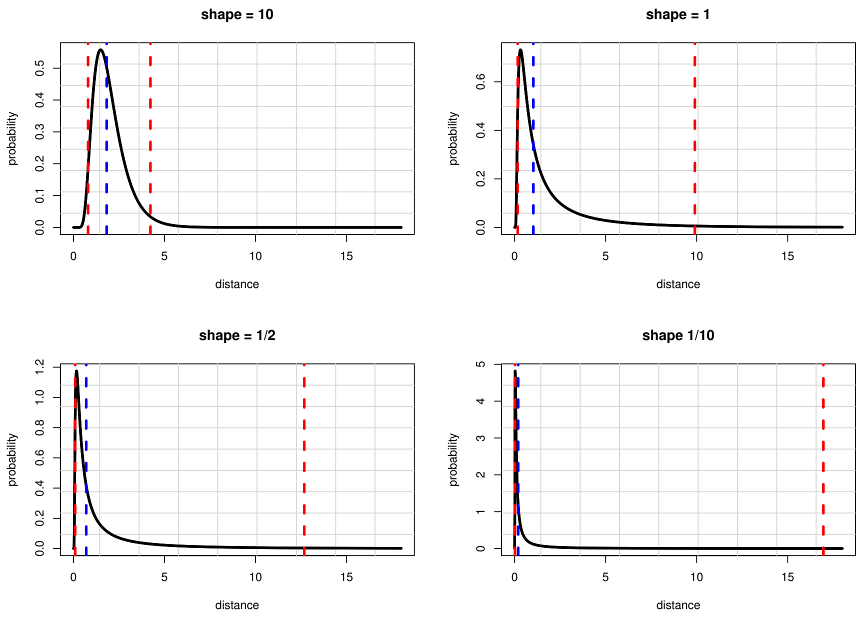


Figure B.3: The figure shows the inverse Gaussian distribution for different shapes and a fixed mean. The red lines indicate the 95 % confidence intervals whereas the blue line indicate the median.

The figure B.3 shows the a high shape will make the distribution more symmetric - looking more like a normal distribution - whereas a low shape will center most of the probability as one point and thus have very wide and strew confidence intervals.

B.3 Spherical to Cartesian Coordinates

Figure B.4 gives an illustrative explanation between spherical and Cartesian coordinates. The purpose of the figure is to illustrate how these two are related.

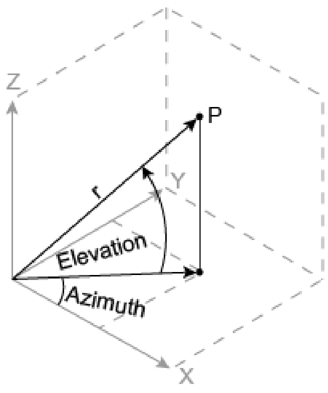


Figure B.4: The figure shows how the point P can be described by spherical coordinates and Cartesian coordinates. [Mat17]

APPENDIX C

Linearizing the transition operator

This section describes how the non-linear transition Equation 3.12 can be linearized by differentition around a point.

The non-linear transition operator F_1 is described by the following formula that maps from step k to step $k+1$.

$$\boldsymbol{x}^g_{k+1} = \begin{bmatrix} \boldsymbol{g}_{k+1} \\ \boldsymbol{q}_{k+1} \\ \boldsymbol{v}_{k+1} \\ \boldsymbol{\omega}_{k+1} \end{bmatrix} = \mathbf{F}_1 \cdot \begin{bmatrix} \boldsymbol{g}_k \\ \boldsymbol{q}_k \\ \boldsymbol{v}_k \\ \boldsymbol{\omega}_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{g}_k + \boldsymbol{v}_k \Delta t \\ \boldsymbol{q}_k \times q(\boldsymbol{\omega}\Delta t) \\ \boldsymbol{v}_k \\ \boldsymbol{\omega_k} \end{bmatrix} \tag{C.1}$$

where q is the quaternion of $\boldsymbol{\omega}\Delta t$. This transition equation assumes constant linear and angular velocities. This $\boldsymbol{v}$ and $\boldsymbol{\omega}$ do not vary over time.

The non-linear part of the transition equation is updating of the orientation $\boldsymbol{q}_{k+1} = \boldsymbol{q}_k \times q(\boldsymbol{\omega}\Delta t)$. By expanding this expression one realize that this is non-linear because the two variables are multiplied.

$$\begin{bmatrix} \Delta t(q_0\ \omega_0 - q_x\ \omega_x - q_y\ \omega_y - q_z\ \omega_z) \\ \Delta t(q_0\ \omega_x + q_x\ \omega_0 + q_y\ \omega_z - q_z\ \omega_y) \\ \Delta t(q_0\ \omega_y - q_x\ \omega_z + q_y\ \omega_0 + q_z\ \omega_x) \\ \Delta t(q_0\ \omega_z + q_x\ \omega_y - q_y\ \omega_x + q_z\ \omega_0) \end{bmatrix} \tag{C.2}$$

This equation needs to be linearize in order to put the system of the form $\boldsymbol{x}_{k+1} = \mathbf{F} \cdot \boldsymbol{x}_k$. To linearize, we differentiate wrt. the state vector, which gives

$$\begin{bmatrix} \Delta t(q_0 - q_x - q_y - q_z + \omega_0 - \omega_x - \omega_y - \omega_z) \\ \Delta t(q_0 + q_x + q_y - q_z + \omega_0 + \omega_x - \omega_y + \omega_z) \\ \Delta t(q_0 - q_x + q_y + q_z + \omega_0 + \omega_x + \omega_y - \omega_z) \\ \Delta t(q_0 + q_x - q_y + q_z + \omega_0 - \omega_x + \omega_y + \omega_z) \end{bmatrix} \tag{C.3}$$

By applying Equation C.3, the system becomes linear and can be expressed put on matrix form as

$$\begin{bmatrix} {g_x}^{k+1} \\ {g_y}^{k+1} \\ {g_z}^{k+1} \\ {q_0}^{k+1} \\ {q_x}^{k+1} \\ {q_y}^{k+1} \\ {q_z}^{k+1} \\ {v_x}^{k+1} \\ {v_y}^{k+1} \\ {v_z}^{k+1} \\ {w_x}^{k+1} \\ {w_y}^{k+1} \\ {w_z}^{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta t & -\Delta t & -\Delta t & -\Delta t & 0 & 0 & 0 & -\Delta t & -\Delta t & -\Delta t \\ 0 & 0 & 0 & \Delta t & \Delta t & \Delta t & -\Delta t & 0 & 0 & 0 & \Delta t & \Delta t & -\Delta t \\ 0 & 0 & 0 & -\Delta t & \Delta t & \Delta t & \Delta t & 0 & 0 & 0 & -\Delta t & \Delta t & \Delta t \\ 0 & 0 & 0 & \Delta t & -\Delta t & \Delta t & \Delta t & 0 & 0 & 0 & \Delta t & -\Delta t & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} {g_x}^{k} \\ {g_y}^{k} \\ {g_z}^{k} \\ {q_0}^{k} \\ {q_x}^{k} \\ {q_y}^{k} \\ {q_z}^{k} \\ {v_x}^{k} \\ {v_y}^{k} \\ {v_z}^{k} \\ {w_x}^{k} \\ {w_y}^{k} \\ {w_z}^{k} \end{bmatrix} \tag{C.4}$$

This section has described how the non linear transition operator can be linerized and how the transition equation (Equation 3.12) can be expressed written on the matrix form.

APPENDIX D

Links for experiments

Experiment 1

.
http://elers.dk/Videos/atDesk.mp4
http://elers.dk/Videos/loop.mp4
http://elers.dk/Videos/aroundImageLab.mp4

Experiment 2

http://elers.dk/Videos/aroundImageLab2.mp4
http://elers.dk/Videos/Recording011.mp4

Experiment 3

http://elers.dk/Videos/outdoor.mp4
http://elers.dk/Videos/Recording013.mp4

Experiment 4

http://elers.dk/Videos/Recording016.mp4
http://elers.dk/Videos/Recording017.mp4

Experiment 5

http://elers.dk/Videos/shortDataRecordings.zip

Experiment 6

http://elers.dk/Videos/recordings.zip

Experiment 7

http://elers.dk/Videos/firkantWalk.mp4
http://elers.dk/Videos/fullstream.mp4
http://elers.dk/Videos/halfcircleandauditorium.mp4
http://elers.dk/Videos/rotationTest.mp4

APPENDIX E
Software Manual

A big part of this project have been to understand and test `EKFmonoSLAM`. `EKFmonoSLAM` is a huge program consisting of 131 files. Therefore, it is chosen to dedicated a section to give an overview of the software and a short introduction to readers who will continue working with the software. Civera et al. [Civ+10] gives a similar outline of the algorithm, however their description is kept in a very general and not implementation specific pseudo-code. This section will give a more low level, language specific outline of the software and the data structure that will hopefully make it easier for future users of the implementation.

`EKFmonoSLAM` is written by Javier Civera and J. M. M. Montiel [Civ+10]. It dates from 2010 and can be downloaded freely from his web page[1]. From this web page, it is also possible to find videos of their results.

E.1 Getting started

To begin with `EKFmonoSLAM` has included a image sequence. Thus, the software can be tested just by running the main script `mono_slam.m`. Note that in Matlab 2017 `update.m` is a reserved file name, thus one needs to change the file name of one of the functions.

If it is wished to use another data from another camera, one needs to calibrate the camera used to record this data and update the file `initial_camera.m` with the appropriate camera parameters. This is described in more detailed in the Camera Calibration section.

E.2 Overview of the software

The core of `EKFmonoSLAM` is the script `mono_slam.m`. This script is very useful to give an overview of the algorithm. In this subsection, some extended comment will be added to the most important functions called from this script. It will be discuss which hard code thresholds can be tuned to obtain better results. Note that some functions

[1] http://webdiis.unizar.es/ jcivera/code/1p-ransac-ekf-monoslam.html

and variable initialization have been excluded from the original script [JM08] to give a better overview.

```
% Camera calibration
cam = initialize_cam;

% Initialize state vector and covariance
[x_k_k, p_k_k] = initialize_x_and_p;

% Initialize EKF filter
sigma_a = 0.007; % standard deviation for linear acceleration noise
sigma_alpha = 0.007; % standard deviation for angular acceleration noise
sigma_image_noise = 1.0; % standard deviation for measurement noise
filter = ekf_filter( x_k_k, p_k_k, sigma_a, sigma_alpha, sigma_image_noise,
    'constant_velocity' );

% variables initialization
features_info = [];
trajectory = zeros( 7, lastIm - initIm );

im = takeImage( sequencePath, initIm );

for step=initIm+1:lastIm

    % Map management (adding and deleting features; and converting inverse
        depth to Euclidean)
    [ filter, features_info ] = map_management( filter, features_info, cam,
        im, min_number_of_features_in_image, step );

    % EKF prediction (state and measurement prediction)
    [ filter, features_info ] = ekf_prediction( filter, features_info );

    % Grab image
    im = takeImage( sequencePath, step );

    % Search for individually compatible matches
    features_info = search_IC_matches( filter, features_info, cam, im );

    % 1-Point RANSAC hypothesis and selection of low-innovation inliers
    features_info = ransac_hypotheses( filter, features_info, cam );

    % Partial update using low-innovation inliers
    filter = ekf_update_li_inliers( filter, features_info );

    % "Rescue" high-innovation inliers
    features_info = rescue_hi_inliers( filter, features_info, cam );

    % Partial update using high-innovation inliers
    filter = ekf_update_hi_inliers( filter, features_info );

end
```

E.2.0.1 Line 1-15

The first 15 lines deals mainly with initialization. The camera parameters are set in `initialize_cam.m`. For details on the calibration please see the section 5.2.

In `initialize_x_and_p`, the initial position, orientation, linear, and angular velocity of the model is set. Also, the initial uncertainty about these quantities are initialized. It is important to note that the size of state vector x_k_k and the matrix p_k_k change during program scope. The first 13 positions of x_k_k are the position, orientation, linear, and angular velocity. The rest of the indexes are features stored in inverse depth or euclidean coordinates. Likewise, the top 13 x 13 matrix is the uncertainty of the camera position orientation, linear, and angular velocity. The rest of p_k_k is the uncertainty about the features.

`sigma_a`, `sigma_alpha`, and `sigma_image_noise` are constant that described the uncertainty of the model and the observations. Thus, with a high `sigma_image_noise` the model will trust the model more than the observations, and on the other hand with a low `sigma_image_noise` it will trust the observations more than the model. Likewise, `sigma_a` and `sigma_alpha` describe the uncertainty about the model.

The `ekf_filter.m` initializes a Matlab struct with the above specified information. The important thing is to note that it initializes a constant velocity model. This can be altered to other models as constant orientation, constant position, etc.

`features_info` is a Matlab struct containing all information about the detected landmarks. This struct is central in keeping track of all the landmarks and their history. The data contained in the struct is used to determine whether to represent the landmarks in inverse depth or euclidean coordinates. The `trajectory` is simply the position and the orientation of the camera through the program scope. This is only used for plotting the results.

E.2.0.2 Line 22

The function `map_management.m` deals with updating the struct `features_info`. First of all, it deletes features if they are not measured half as many times as they are predicted. Then it checks if some features represented by inverse depth coordinates can be converted into euclidean coordinates based on a linearity index. And finally, it deals with adding new features. This is done through the functions `initialize_a_feature.m` that calls `fast_corner_detect_9.m` and `fast_nonmax.m` that both takes a hard coded threshold that determines how many features are detected. To obtain real-time performance a high threshold should be chosen, however to obtain better results the threshold should be decreased (See Section 5.4).

E.2.0.3 Line 25

The ekf_prediction.m makes a prediction about the next state for the features and the camera using the formulas described in the sections about the Transition Equations. In this script it is possible to alter to other models such as 'constant orientation' or 'constant position'.

E.2.0.4 Line 31

The search_IC_matches.m find matches between features ind the features_info and features found in a new image. The functions only updates features_info based on whether the features a matched or not. The search_IC_matches.m uses the uncertainty about the features in features_info to perform Active Search. Based on the uncertainty of ones' parameters it is possible to tune the search region in this function. This can be a good idea if the software have a difficulties initializing, because the greatest uncertainty is in the first step. From a function called matching.m, it is possible to adjust the threshold for correlation to enable more matches.

E.2.0.5 Line 34

The ransac_hypotheses.m function takes a random match and makes a state update based on this one point. The function compute_hypothesis_support_fast.m is used to compute the support of the hypothesis. The sigma_image_noise is used as a threshold to compute the support of a hypothesis. When a hypothesis has enough support, the features_info is updated with the update feature positions. The formulas are described in more detail in the section 3.9.

E.2.0.6 Line 37-43

The rest of the loop deals with updating the filter. This is first done by updating the model based on the low innovation inliers (the features with small distance to the most supported hypothesis) in ekf_update_li_inliers.m. The low innovation inliers are often distant points as they are often viewed more stationary.
This updating greatly reduced the uncertainty about the model. This reduction in uncertainty is used to rescue high innovation inliers in rescue_hi_inliers.m. These high innovation inliers are often closer points where the translation has greater influences on their position. Finally, the filter is updated based on these high innovation inliers in the function ekf_update_li_inliers.m.

This section has presented the EKFmonoSLAM Matlab implementation. It is briefly discussed how to get started using the software. A low level, implementation specific overview of the software is provided (for a more general description [Civ+10]). In this overview, it is highlighted where in the code it is possible to change hard code values and the effect of these changes are briefly discussed. In the next section, it will be explained how data was collected.

APPENDIX F

Sensor Fusion

The IMU that is mounted on the Tobii Pro Glasses 2 provides accelerometer and gyroscope data. Using multiple data sources provides a more nuanced representation of the world and should provide a better estimate of the path. In order to use this extra sensor information to support, it is needed to extend the model. This section will describe how the model is changed in order to support the IMU sensor.

F.1 Extracting gyroscope & accelerometer data

The Tobii Pro Glasses 2 automatically collects eye-tracking, accelerometer, and gyroscope data. The data can be obtained from the memory card, however it is stored in a json format which is rather troublesome for most analytic software besides Tobii's own. [Wul17] has made a detailed investigation of the json file and have also developed a tool that convert the data into three comma separated txt files. The software [1] output three separate file (one containing the the eye-tracking data, one containing the accelerometer data, and one containing the gyroscope data), since these are collected with respectively 50Hz, 100Hz, and 95Hz [Wul17]. The section will focus on further processing of the accelerometer and gyroscope data. For detailed information on the raw data and the eye-tracking data please read [Wul17].

The unit for the accelerometer is meter per second squared and the unit for the gyroscope is degrees per second. The accelerometer data is collected with 100Hz and the gyroscope data is collected with 95Hz [Wul17]. The video feed is collected with 25Hz, thus in order to incorporated the accelerometer and gyroscope data into the EKF, it is chosen to reduce the frequency of the accelerometer and gyroscope data. This is done by taking an average of every fourth observation for the accelerometer data, and an average of every 3.8 observation for the gyroscope data, thus equal time intervals between each observation are obtained.

[1] The software tool can be found on Git: https://github.com/anwul4/Tobii_JsonToTxtConversionTool

F.2 Incorporating the gyroscope

Only the measurement equation has to be altered in order to use the gyroscope data.

$$\begin{bmatrix} u \\ v \\ \boldsymbol{\omega} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{H}}{\partial \boldsymbol{g}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{q}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{v}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{\omega}} \\ \frac{\partial \mathbf{H}}{\partial \boldsymbol{g}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{q}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{v}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{\omega}} \\ 0 & 0 & \mathbf{1} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{g} \\ \boldsymbol{q} \\ \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} \tag{F.1}$$

where **H** is measurement operator described in Section 3.6, thus the two top rows are the linearization of this operator. From this equation it is seen that it is rather simple to add new sensors.

In practice, the additional row can be implemented in a separate function, thus dividing the updating of the filter into three instead of two. Thus, the following function will incorporate the gyroscope measurements.

```matlab
function filter = ekf_update_gyro(filter, gyro)

% mount vectors and matrices for the update

% get measurements
z =  gyro';
% get angular velocity from state vector
x_k_k = filter.x_k_k;
h = x_k_k(11:13);

% initialize the measurement operator H
H = zeros(length(z),length(x_k_k));
H(1:length(z),11:13) = eye(length(z));

% Decide on the error associated with gyroscope measurements.

%R = 0.00001*eye(length(z)); %very low!
%R = 0.0001*eye(length(z)); %low!
%R = 0.001*eye(length(z)); %medium!
%R = 0.01*eye(length(z)); %high!
%R = 0.1*eye(length(z)); %very high!
R =  1*eye(length(z)); %very very high!


% Update filter.
[ filter.x_k_k, filter.p_k_k ] = update_filter( filter.x_k_km1, filter.
    p_k_km1, H,...
    R, z, h );

end
```

Using the above function, it was possible to try out different noise terms. Figure F.1 shows how the angular velocity is modelled using different noise terms.

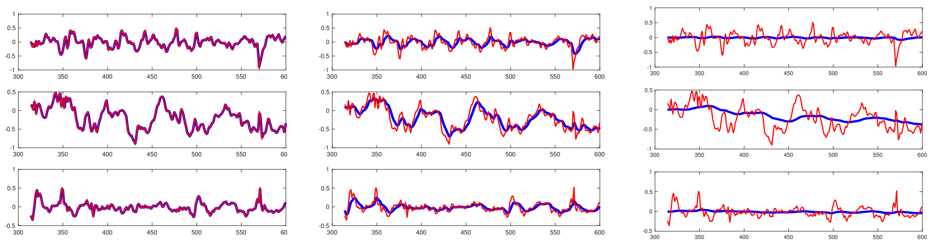


Figure F.1: The figure shows how the angular velocity is dependent on the choice of the error associated with the observations. Producing these plots, the EKF only used gyroscope data to update. The blue line is the angular velocity modelled in the state vector and the red line is the observations from the gyroscope.

The modelled angular velocity effects the orientation through the transition equation. This is shown in Figure F.2, where the orientation without the gyroscope is the blue line and the orientation with different noise terms are the red and green lines.

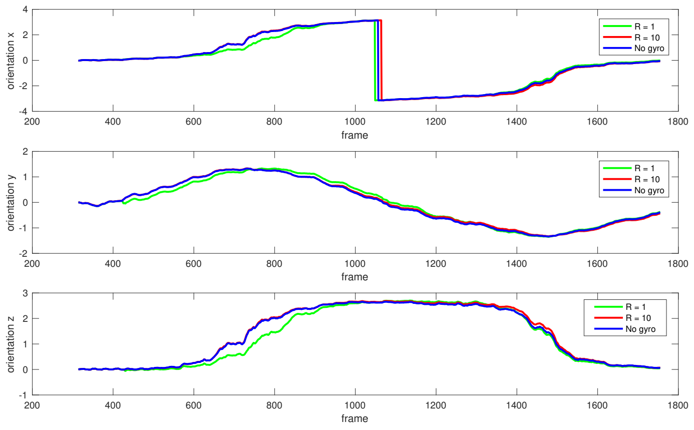


Figure F.2: The figure shows the orientation of the state vector when the gyroscope data is not used and for two difference choices of the error.

Finally, the orientation effects the position through the transition equation. Figure

F.3 shows the final path for the experiment using the data from the gyroscope and for an experiment where this data has not been used. It is seen from this figure that the estimate path without the gyroscope data is best.

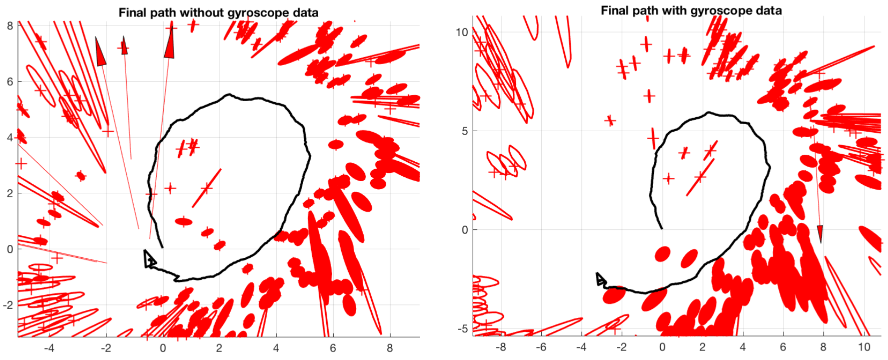


Figure F.3: This figure shows the final path for the loop recording in Experiment 1 Section 4.1. From the figure it is seen that the final path is closer to a loop closing when the gyroscope data is not used.

Based on these observations it seem as the partial update of the EKF works as expected. However, it does not seem to be any improvements right away. However, it is believed that further fine tuning of the associated error in the update function could result in improved accuracy especially during fast turns. However, more research and development has to be put into this implementation.

F.3 Incorporating the gyroscope & accelerometer

Because we want to update the prediction with respect to the measured acceleration, we need to model the acceleration. Thus, the acceleration in the x, y, z direction is added to the state vector. Note that we already model the angular velocity and thus we do not need to further change the state vector in order to use the measurements from the gyroscope sensor.

By modelling the acceleration in the state vector, the we cannot model the acceleration as noise any more. Therefore, the transition of position and velocity also becomes dependent on the acceleration (Equation F.2). This means that we use a constant acceleration model instead of as previous a constant velocity model.

$$\begin{bmatrix} \boldsymbol{g}_{k+1} \\ \boldsymbol{q}_{k+1} \\ \boldsymbol{v}_{k+1} \\ \boldsymbol{\omega}_{k+1} \\ \boldsymbol{a}_{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{g}_k + (\boldsymbol{v}_k + \boldsymbol{a}_k \Delta t)\Delta t \\ \boldsymbol{q}_k \times q(\boldsymbol{\omega}_k) \\ \boldsymbol{v}_k + \boldsymbol{a}_k \Delta t \\ \boldsymbol{\omega}_k \\ \boldsymbol{a}_k \end{bmatrix} \tag{F.2}$$

where a is the acceleration in the x, y, z direction.

Besides changing the transition equation, it is also needed to modify the measurement operator that maps from the model space to the measurement space. The measurements space is no longer 2 dimensions (the image plane) but is now 8 dimensions (image plane, gyroscope measurements, and acceleration measurements). A big advantage with the EKF is that it is rather straightforward to add sensors. Each row in the measurement operator corresponds to a sensor dimension. Thus the measurement equation $\boldsymbol{z} = \mathbf{H}\boldsymbol{x}$ can be formulated as

$$\begin{bmatrix} u \\ v \\ \boldsymbol{\omega} \\ \boldsymbol{a} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{H}}{\partial \boldsymbol{g}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{q}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{v}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{\omega}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{a}} \\ \frac{\partial \mathbf{H}}{\partial \boldsymbol{g}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{q}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{v}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{\omega}} & \frac{\partial \mathbf{H}}{\partial \boldsymbol{a}} \\ 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} \end{bmatrix} \begin{bmatrix} \boldsymbol{g} \\ \boldsymbol{q} \\ \boldsymbol{v} \\ \boldsymbol{\omega} \\ \boldsymbol{a} \end{bmatrix} \tag{F.3}$$

where $\mathbf{H}$ is measurement operator described in Section 3.6, thus the two top rows are the linearization of this operator. From this equation it is seen that it is rather simple to add new sensors. Especially in this case where we already model the acceleration and orientation in the same form as they are measured by the IMU.

In this section, it has been described how to use a IMU sensor to make a bigger, more nuanced model. The modified model is a constant acceleration model where the acceleration is added to the state vector. The measurement operator has also been modified such that it also maps the modelled acceleration and orientation to the measurement space.

Bibliography

[Aan15] Henrik Aanæs. *Lecture Notes on Computer Vision*. DTU informatics, 2015.

[Bac17] Jen Bacher. *LENS DISTORTION: WHAT EVERY PHOTOGRAPHER SHOULD KNOW*. 2017. URL: https://clickitupanotch.com/lens-distortion/.

[BS05] M. Bryson and S. Sukkarieh. “Bearing-only SLAM for an airborne vehicle.” In: *In Australian Conference on Robotics and Automation* (2005).

[CD08] Margarita Chli and Andrew J. Davison. “Active Matching”. In: *Imperial College London, London SW7 2AZ, UK* (2008).

[Civ+10] Javier Civera et al. “1-Point RANSAC for EKF FIltering. Application to Real-Time Structure from Motion and Visual Odometry.” In: *IEEE Transactions On Robotics* (2010).

[Cor] Inc. Corvallis Microtechnology. *Introduction to GPS*. (Accessed on 01/05/2018). URL: http://www.cmtinc.com/gpsbook/index.htm#top.

[Cyr17] Giorgio Grisetti Cyrill Stachniss Udo Frese. *OpenSLAM.org*. 2017. URL: https://openslam.org.

[Dav+04] Andrew J. Davison et al. “Real-Time 3d SLAM with Wide-Angle Vision”. In: *IFAC Proceedings Volumes, vol. 37, no. 8* (2004).

[Dav03] Andrew Davison. “Real-time simultaneous localization and mapping with a single camera”. In: *Proc. International Conference on Computer Vision* (2003).

[EL98] Martin Koch Erik D. Dam and Martin Lillholm. “Quaternions, Interpolation, and Animation”. In: (1998).

[Goo] Google. *Matematiktorvet - Google Maps*. (Accessed on 01/05/2018).

[JD06] Javier Civera J.M.M.Montiel and Andrew J. Davison. “Unified Inverse Depth Parametrization for Monocular SLAM”. In: *Robotics: Science and Systems* (2006).

[JM08] Andrew J. Davison Javier Civera and J.M.M Montiel. “Inverse Depth Parametrization for Monocular SLAM”. In: *IEEE Transactions On Robotics* (2008).

[Mad07] Henrik Madsen. *Time Series Analysis*. Chapman and Hall, 2007.

[Mat17] Mathworks. *Mathworks.com*. 2017. URL: https://www.mathworks.com/help/matlab/ref/math_sphcart.png.

[Nis04] D. Nistér. “An effucuent solution to the five-point relative pose problem”. In: *IEEE Transactions On Pattern Analysis and Machine Intelligence* (2004).

[RD05] Edward Rosten and Tom Drummond. “Fusing Points and Lines for High Performance Tracking”. In: (2005).

[Ros+08] Edward Rosten et al. “Faster and better: a machine learning approach to corner detection”. In: (2008).

[Tob17] Tobii. *Tobii Pro Glasses 2*. 2017. URL: https://www.tobiipro.com/imagevault/publishedmedia/gw66xob79wkirj0720oh/TobiiPro-Glasses2-tech-specs-image-3_1.jpg.

[Vis17] Deepak Geetha Viswanathan. *Features from Accelerated Segment Test (FAST)*. 2017. URL: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV1011/AV1FeaturefromAcceleratedSegmentTest.pdf.

[Wul17] Andreas Wulff-Jensen. “Data Conversion Tool For Tobii Pro Glasses 2 Live Data Files”. In: (2017).