Hallvard Fosso

Thermal VSLAM for autonomous ferry

Master's thesis in cybernetics and robotics
Supervisor: Edmund Brekke, Trym Haavardsholm
February 2021

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Hallvard Fosso

Thermal VSLAM for autonomous ferry

Master’s thesis in cybernetics and robotics
Supervisor: Edmund Brekke, Trym Haavardsholm
February 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Preface

This report was written as part of the course TTK4900 - Engineering Cybernetics, Master's Thesis, and concludes my master's degree in cybernetics at NTNU Trondheim with specialization in autonomous systems. The thesis was completed during 2020 late autumn and 2021 after New Year, and is a continuation of the author's specialization project with the same title completed during the spring of 2020 of which some parts are re-used. The topic of visual simultaneous localization and mapping sparked an interest in me, having experience with outdoor land navigation from the Norwegian Trekking Association (DNT) and the Norwegian Army. I also hold a certificate of apprenticeship in electronics (Norwegian: fagbrev i elektronikk) and have worked with maintenance and repairs of cameras and electronics, as well as having an interest in optical systems and cameras as a hobby. When I heard of VSLAM I was fascinated about the fact that computers could be able to perform visual based navigation for us, and started wondering how this worked. I hope this thesis can contribute to the further understanding and use of VSLAM both within cybernetics and other fields. Cameras are in abundance, and thermal cameras have properties which make them interesting for safety critical applications. I can hardly think of any more exciting applications for computer vision currently than VSLAM.

I would like to thank my supervisors Edmund Brekke and Trym Haavardsholm for their advice and contribution to this project. Especially I would like to thank Trym for his code for saving frame trajectories in ORB-SLAM2 and for providing example IMU noise parameters. I would also like to thank Erik Wilthil for providing datasets and Øystein Kaarstad Helgesen for letting me borrow a thermal camera for testing.

Hallvard Fosso
Stavanger, 1 February 2021

Abstract

The thesis presents an overview of ORB-SLAM1, 2 and 3, and shows the implementation and adaptations to ORB-SLAM2 and ORB-SLAM3 for thermal video. Thermal VSLAM on autonomous vehicles is motivated by making the navigation more robust for example in nighttime, fog, or in GNSS-denied environments, resulting in better overall situational awareness and increased safety. However, in the literature there has not been much research related to the adaptation of visual spectrum based VSLAM methods with thermal video, but rather with thermal video specific SLAM methods. To investigate its feasibility, ORB-SLAM2 was implemented with thermal video in the authors' specialization project, and based on results from the specialization project it was decided to look at three key areas for improvement: initialization, place recognition and tracking. A focus was put on creation of a vocabulary based on thermal images in an attempt to improve place recognition. In addition to creation and testing of various vocabularies, different camera parameters were tested, and a simple solution leaving out radial and tangential distortion showed the best results. The newly released ORB-SLAM3 (2020) was implemented, both with and without IMU, with the goal to further improve performance in regards to initialization, place recognition and tracking with thermal video. The built-in ORB-SLAM vocabulary was replaced with new vocabularies trained on the thermal images taken both from indoor and outdoor. The new vocabularies based on thermal images however only achieved similar performance on thermal video as the vocabulary bundled with ORB-SLAM based on visual spectrum images. On our data, ORB-SLAM3 showed an improvement with the ability to use multi-maps in cases where track was lost, allowing continued mapping in difficult situations. However, in a realistic case study for a ferry, ORB-SLAM2 performed better in regards of keeping the problem bounded and not creating too many new keyframes. ORB-SLAM2 performed well in a realistic case study. The hope is that these findings can be useful towards the goal of achieving navigation using multi sensor fusion with other sensors like lidar, radar and optical camera.

Sammendrag

Oppgaven presenterer en oversikt over ORB-SLAM1, 2 og 3, og viser implementering og tilpasninger av ORB-SLAM2 og ORB-SLAM3 for termisk video. Termisk VSLAM på autonome fartøy er motivert av et ønske om å gjøre navigasjonen mer robust for eksempel i natt, tåke eller miljøer uten GNSS-tilgang, i håp om å gi økt sikkerhet og bedre generell situasjonsforståelse. Imidlertid har det ikke vært publisert mye forskning i litteraturen om tilpasning av VSLAM-metoder laget for video i det synlige spektrumet til termisk video, men istedet noe forskning på termisk-spesifikke SLAM-metoder. For å undersøke muligheten for dette ble ORB-SLAM2 implementert med termisk video i forfatternes spesialiseringsprosjekt, og basert på resultatene fra spesialiseringsprosjektet ble det besluttet å se på tre viktige områder for forbedring: initialisering, stedsgjenkjenning og sporing. Fokus ble lagt på å generere et vokabular (ordforråd) basert på termiske bilder i et forsøk på å forbedre stedsgjenkjenning. I tillegg til generering og testing av forskjellige vokabularer ble forskjellige kameraparametre testet, og en enkel løsning som utelater radiell og tangentiell forvrengning gav best resultat. Nylig publiserte ORB-SLAM3 (2020) ble implementert, både med og uten IMU, med fomål å forbedre ytelsen ytterligere med hensyn til initialisering, stedgjenkjenning og sporing med termisk video. Det innebygde ORB-SLAM-vokabularet ble erstattet med nye vokabularer trent på termiskbilder tatt både innendørs og utendørs. De nye vokabularene basert på termiske bilder oppnådde imidlertid bare tilsvarende ytelse på termisk video som vokabularet som følger med ORB-SLAM og er basert på bilder i det visuelle spektrumet. På våre data viste ORB-SLAM3 en forbedring med muligheten til å bruke multikart i tilfeller der sporingen gikk tapt, noe som muliggjorde fortsatt kartlegging i vanskelige situasjoner. I et realistisk casestudie for en ferge presterte imidlertid ORB-SLAM2 bedre med hensyn på å holde problemet begrenset og ikke skape for mange nye nøkkelbilder. ORB-SLAM2 presterte bra i en realistisk casestudie. Håpet er at disse funnene kan være nyttige mot målet om å oppnå navigering ved hjelp av multi-sensorfusjon med andre sensorer som lidar, radar og visuelt-spektrum-kamera.

Table of Contents

Abbreviations

BA	=	Bundle Adjustment
BoW	=	Bag of Words
BRIEF	=	Binary Robust Independent Elementary Features
DBoW2	=	Dorian Galvez-Lopez' Bag of Words 2
FAST	=	Features from Accelerated Segment Test
FPS	=	Frames per second
IR	=	Infrared
IRT	=	Infrared thermography, also called thermal imaging
KF	=	Keyframe
LM	=	Levenberg-Marquardt
ORB	=	Oriented FAST Rotated BRIEF
RGB-D	=	RGB color image with Depth information
rBRIEF	=	Rotation-aware Binary Robust Independent Elementary Features
ROS	=	Robot Operating System
SLAM	=	Simultaneous Localization and Mapping
VSLAM	=	Visual Simultaneous Localization and Mapping
VO	=	Visual Odometry

Chapter 1

Introduction

1.1 Motivation and background

Knowing the position of your own vehicle (the agent) and surrounding obstacles is essential for safe and efficient navigation. Simultaneous localization and mapping (SLAM) methods can contribute to gain situational awareness, and can both be an important part of autonomous navigation systems as well as a support tool in manned vehicles.

Some types of SLAM rely on active ranging sensors such as lidar, radar or sonar, which can provide ranging with a relatively high degree of accuracy. VSLAM uses camera sensors, which are passive. Monocular VSLAM requires additional steps to get precise ranging. Camera sensors can however provide rich information about the nearby environment such as colour, texture and lighting. Computer vision algorithms can be used to extract as much information from camera images from a scene as human vision can, or even better than humans, and the information can be used for 3D reconstruction to build a map and to localize the agent within that map.

Usage of thermal infrared cameras is motivated by a desire to compensate for some of the shortcomings with cameras operating in the visual spectrum, and thermal cameras give new capabilities to camera based systems like the ability to see through fog, smoke and dust (Vidas and Sridharan, 2012), continued operation in the absence of light, and resilience to being blinded by bright light and other dynamic lighting conditions, either in the whole or parts of the image. Maybe thermal imaging will be the technology that will give us the ability to see as a hawk during the day and an owl during the night.

Other previous student projects at the NTNU Department of Engineering Cybernetics (ITK) have included the usage of lidar SLAM, while this thesis investigates the use of VSLAM with thermal infrared video. Motivations include future possibility of sensor fusion of thermal VSLAM with sensor data from GNSS, IMU and lidar, as well as other SLAM methods, to provide a more robust system with increased situational awareness.

Milliampere is a 5×2.4 meter research vessel developed by NTNU with the goal of becoming Norway's first passenger carrying autonomous watercraft, and is part of the NTNU Autoferry project. The goal for this electric ferry is to be able to autonomously transport cyclists and pedestrians about 100 meters between the two points of Ravnkloa and Vestre kanalhavn in the Trondheim harbor. Currently the ferry is controlled manually, but it is technically enabled for future autonomous operation by being richly equipped with sensors such as GNSS, IMU, thermal cameras and visual spectrum cameras, and functions as an advanced test bed for applied science.

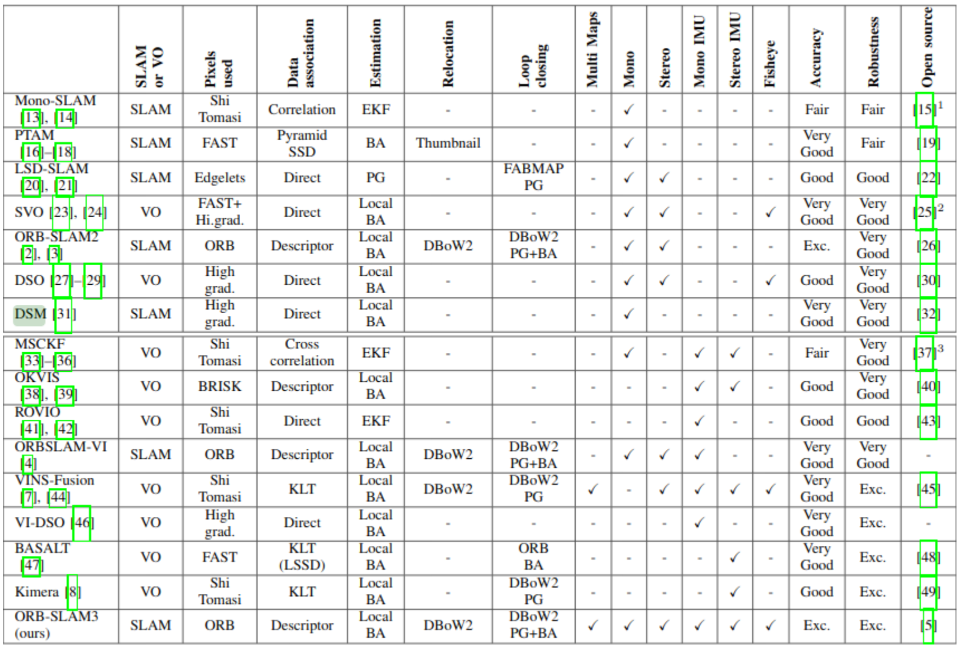
It is highly likely that many of the results on Milliampere can be transferred to the general boat traffic, both in autonomous vessels as well as in manned vessels in the form of support tools for crew which can increase safety onboard. Thermal VSLAM in particular can also be a valuable tool for human ship navigators faced with poor sight due to fog or the dark of the night, even when GNSS is lost.

1.2 Literature review and previous work

In the recent years, VSLAM has seen large research interest and advancements in terms of accuracy and effectiveness (Ma et al., 2019). ORB-SLAM is a popular and well documented VSLAM method with over 2300 paper citations combined according to IEEE Xplore. Table 1.1 from the makers of ORB-SLAM3 (Campos et al., 2020) lists some notable examples of recent, influential and mostly open-source VSLAM methods. Most of these methods have also been published in several versions with added features. For example, LSD-SLAM has been published in one version for mono camera input (Engel et al., 2014) and one for stereo camera input (Engel et al., 2015).

Older methods tend to be particle filter based (like for example Mono-SLAM in the table). Parallel tracking and mapping (PTAM) was influential in the development of ORB-SLAM, particularly when comes to some of the basic ideas behind the framework for tracking and relocalization. Most of the other methods listed in Table 1.1 have been published during the last 5 to 10 years. As we can see from the table, most of the recent VSLAM methods are graph based and use local bundle adjustment.

Some of the methods listed in Table 1.1 are visual odometry (VO) methods rather than SLAM methods, but can be good candidates for being developed into SLAM methods. The first part of the table lists methods which do not readily accept input from an inertial measurement unit (IMU), while the second part lists IMU-ready methods. ORB-SLAM3 has added the support for an inertial measurement unit. The fusion of visual and inertial cues has been a popular topic in the robotics community for some years due to these two sensing capabilities being somewhat complementary in nature (Leutenegger et al., 2013). The table also shows support for multi maps, monocular and stereo images, fisheye lens model, as well as the ORB-SLAM makers' view on their accuracy and robustness.



	SLAM or VO	Pixels used	Data association	Estimation	Relocation	Loop closing	Multi Maps	Mono	Stereo	Mono IMU	Stereo IMU	Fisheye	Accuracy	Robustness	Open source
Mono-SLAM [13], [14]	SLAM	Shi Tomasi	Correlation	EKF	-	-	-	✓	-	-	-	-	Fair	Fair	[15][1]
PTAM [16]–[18]	SLAM	FAST	Pyramid SSD	BA	Thumbnail	-	-	✓	-	-	-	-	Very Good	Fair	[19]
LSD-SLAM [20], [21]	SLAM	Edgelets	Direct	PG	-	FABMAP PG	-	✓	✓	-	-	-	Good	Good	[22]
SVO [23], [24]	VO	FAST+ Hi.grad.	Direct	Local BA	-	-	-	✓	✓	-	-	✓	Very Good	Very Good	[25][2]
ORB-SLAM2 [2], [3]	SLAM	ORB	Descriptor	Local BA	DBoW2	DBoW2 PG+BA	-	✓	✓	-	-	-	Exc.	Very Good	[26]
DSO [27]–[29]	VO	High grad.	Direct	Local BA	-	-	-	✓	✓	-	-	✓	Good	Very Good	[30]
DSM [31]	SLAM	High grad.	Direct	Local BA	-	-	-	✓	-	-	-	-	Very Good	Very Good	[32]
MSCKF [33]–[36]	VO	Shi Tomasi	Cross correlation	EKF	-	-	-	✓	-	✓	✓	-	Fair	Very Good	[37][3]
OKVIS [38], [39]	VO	BRISK	Descriptor	Local BA	-	-	-	-	-	✓	✓	-	Good	Very Good	[40]
ROVIO [41], [42]	VO	Shi Tomasi	Direct	EKF	-	-	-	-	-	✓	-	-	Good	Good	[43]
ORBSLAM-VI [4]	SLAM	ORB	Descriptor	Local BA	DBoW2	DBoW2 PG+BA	-	✓	✓	✓	-	-	Very Good	Very Good	-
VINS-Fusion [7], [44]	VO	Shi Tomasi	KLT	Local BA	DBoW2	DBoW2 PG	✓	-	✓	✓	✓	✓	Very Good	Exc.	[45]
VI-DSO [46]	VO	High grad.	Direct	Local BA	-	-	-	-	-	✓	-	-	Very Good	Exc.	-
BASALT [47]	VO	FAST	KLT (LSSD)	Local BA	-	ORB BA	-	-	-	-	✓	-	Very Good	Exc.	[48]
Kimera [8]	VO	Shi Tomasi	KLT	Local BA	-	DBoW2 PG	-	-	-	-	✓	-	Good	Exc.	[49]
ORB-SLAM3 (ours)	SLAM	ORB	Descriptor	Local BA	DBoW2	DBoW2 PG+BA	✓	✓	✓	✓	✓	✓	Exc.	Exc.	[5]

Figure 1.1: From the makers of ORB-SLAM3 (Campos et al., 2020): A selection of recent influential VSLAM methods (all but two of those listed are open-source).

One example of a VO method which has been developed into SLAM is Direct sparse odometry (DSO) (Engel et al., 2016), which has been extended into the SLAM method Direct sparse odometry with loop closure (LDSO) (Gao et al., 2018).

Another example of a recognized visual odometry method is Fast semi-direct monocular visual odometry (SVO) (Forster et al., 2014), of which CNN-SVO is a recent advancement which uses the addition of convolutional neural networks (CNN) to estimate depth from a single image (Loo et al., 2019). Semi-direct monocular visual odometry using fixed maps (FSVO) (Fu et al., 2017) is another derivative of SVO which uses a fixed map leading to less computational cost, and has shown improved performance on the EuRoC and KITTI datasets.

During the finalizing of this master's thesis, in January 2021, Real-time dynamic SLAM using semantic segmentation methods (RDS-SLAM) was published which is a real-time dynamic SLAM method built on ORB-SLAM3, but adds a novel semantic thread and a semantic-based optimization thread running in parallel with the other threads in order to detect dynamic objects and remove outliers (Liu and Miura, 2021). In addition to adding semantic meaning, RDS-SLAM also uses fiducial markers, which means that known objects can be used as a point of reference or for measuring. RDS-SLAM has currently not yet been published as open-source, and currently has currently only been developed for RGD-D video. According to its makers, RDS-SLAM is planned to also be developed for mono and stereo camera input, and to be published as open-source.

1.2.1 ORB-SLAM

Released in 2015, ORB-SLAM1 was a complete SLAM system which used the same features for all SLAM tasks such as tracking, mapping, relocalization and loop closing. It had a survival fittest strategy which used only selected keyframes and points in order to keep the map size growth controllable, used several parallel threads and operated in real time in large environments. According to its authors, ORB-SLAM1 achieved good performance on well-known datasets. ORB-SLAM has been further developed over the years with ORB-SLAM2 adding full bundle adjiustment, as well as RGB-D and stereo input capabilities, while ORB-SLAM3 added support for IMU, several camera models, an atlas which can contain several maps, and improved place recognition. Due to its popularity, its many features, as well as both ORB-SLAM1, 2 and 3 being available as open-source, this makes ORB-SLAM a good candidate for a framework for researchers who want to implement VSLAM.

1.2.2 Thermal SLAM

Thermal images often appear to show large surfaces with gradual transitions of object's temperatures, often leading to not less sharp and clearly defined corners and edges in some situations compared to visual images. In a similar fashion, since being able to image surface temperatures, thermal images can show features which are invisible in visual images.

Vidas and Sridharan (2012) designed a thermal monocular SLAM which achieved robust performance on high-noise low-texture images. Borges and Vidas (2016) noted that it was "not straight forward to apply standard visual odometry algorithms to thermal imainging data", and proposed another thermal monocular VO system. Khattak et al. (2019) proposed a VO method for fusing radiometric thermal imagery with inertial measurements for robust navigation in GNSS-denied and visually-denied environments. Shin and Kim (2019) also noted that applying 14-bit radiometric thermal imagery to visual based methods was difficult due to modality differences, and proposed a thermal SLAM system for 6 degrees of freedom motion, enhanced by sparse depth measurements from lidar, which was used to successfully overcome the scale problem of a monocular camera. Saputra et al. (2019) tried to get around the problem of thermal imagery having few robust visual features by overlaying a "hallucination" of a visual image with the help of a neural network, to predict fake visual features from thermal images by using a Huber loss function. NASA plans to send robots out in space to map areas beneath ice, like for example Europa (one of Jupiter's moons). For this purpose, (Gonzalez et al., 2019) investigated the use of a thermal camera to augment VSLAM using visible light cameras. Their solution using the cameras combined gave better results than using the visible light camera alone.

There have also been done works with other types of affordable infrared distance sensors, but these are active sensors which measure distance using reflections of infrared light.

1.2.3 Related student projects at NTNU

Previous student projects at the NTNU Department of Engineering Cybernetics (ITK) using lidar SLAM includes Even Skjellaug (2020) and Marius Strand Ødven (2019). Ødven created an overview of lidar based SLAM methods and tested three of them. Ødven also recorded a dataset from the Milliampere ferry in the Trondheim harbor environment focusing on docking scenarios with lidar, IMU, GNSS and Real Time Kinematics (RTK) data as a fundament for others to build on. Based on Ødven's work, Skjellaug proposed a system built on the iSAM2 framework and fusion of lidar, IMU and GNSS. Using lidar, IMU and RTK, this system performed better than a standard GNSS receiver used as a reference. Loop closure was also shown to be consistent, and the system proposed was therefore one of the first feature-based lidar SLAM systems using keypoints and descriptors in order to perform loop closure. Other currently ongoing student projects at ITK includes fiducial SLAM by Martin Eek Gerhardsen, multi-sensor SLAM by Thomas Hellum, and VSLAM for automatic docking by Dinosshan Thiagarajah.

1.3 Problem description

The goal of this project is to evaluate the feasibility of thermal infrared VSLAM using ORB-SLAM. The feasibility of using thermal-infrared video with ORB-SLAM is investigated experimentally in a marine environment, with regard to issues such as feature detection, place recognition and initialization. Necessary adaptations should be identified and performed. One research topic of particular interest is bag of words. An overview of the architecture and components of ORB-SLAM will be demonstrated.

1.4 Outline

- In chapter 2, general computer vision concepts necessary for understanding ORB-SLAM is presented.
- In chapter 3, theoretical background about ORB-SLAM is explained. The original ORB-SLAM1 as well as additions implemented in ORB-SLAM2 and ORB-SLAM3 are presented.
- In chapter 4, an overview and theoretical background about infrared imaging (including thermal imaging) and IMU readings is presented.
- In chapter 5, the implementation steps are shown for ORB-SLAM with ROS using thermal video, as well as creation of a dataset from a thermal camera, steps to train a vocabulary based on an image set, and steps for trajectory plotting.
- In chapter 6, results are presented and discussed. Results include camera data collection and performance of self-trained vocabularies. The ORB-SLAM2 method is also compared with ORB-SLAM3 (with and without IMU).

- In chapter 7, a conclusion and suggestions for further work are presented.

Chapter 2

Computer vision theory

In this chapter some computer vision concepts are included as they are relevant in terms of understanding ORB-SLAM.

2.1 Pinhole camera model and intrinsics

ORB-SLAM uses the pinhole camera model, which is the simplest camera model. Camera coordinates $\mathbf{x^c} = \begin{bmatrix} x^c \\ y^c \\ z^c \end{bmatrix}$ can be found from image coordinates $\mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix}$ by:

$$\mathbf{x^c} = \pi_p^{-1}(\mathbf{u}, z; K) = z \begin{bmatrix} \dfrac{u - c_u}{f_u} \\ \dfrac{v - c_v}{f_v} \\ 1 \end{bmatrix} \tag{2.1}$$

where K is the intrinsic matrix. The intrinsic matrix is constructed in the following manner:

$$K = \begin{bmatrix} f_u & s_\theta & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

where c_u and c_v represents the principal point. f_u and f_v is the focal length.

2.1.1 Distortion coefficients

The radial distortion in x and y direction can be described as:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.3}$$
$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.4}$$

The tangential distortion in x and y direction can be described as:

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{2.5}$$
$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{2.6}$$

Distortion can be represented by five parameters in ORB-SLAM using OpenCV style camera parameters. These are known as the distortion coefficients given by $(k_1, k_2, p_1, p_2, k_3)$

2.1.2 Scale

When using a monocular camera the scale is undetermined, which means that we can not accurately determine the size of objects. For monocular VSLAM this means that the scale of the reconstruction and the estimated trajectory will be unknown, and the scale will therefore be set to an arbitrary value. The ORB feature extraction implements a scale pyramid to be able to detect features again at different scales. A more in depth presentation of how this works will be presented in the next chapter. Due to the fact that a single camera itself doesn't provide scale information, a monocular SLAM system will drift in scale after some time. Some methods for obtaining scale can be from the use of external sources like IMU or GNSS readings, or by using a stereo camera arrangement instead of a monocular camera. ORB-SLAM2 introduced support for stereo camera, while ORB-SLAM3 also provides the possibility to add IMU readings and fuse them with monocular SLAM. This capability is discussed more in the next chapter. Other suggestions for methods which can provide scale are to use information about the surroundings, like for example the real size of an actual object in the scene, or by knowing the real distance between two objects in the scene. Additionally, the loop closing is a part of the SLAM algorithm that is able to correct for drift to some extent as well. In order to compare monocular trajectories, one can align and scale them with a reference to make them more comparable.

2.2 Hamming distance

The Hamming distance is a distance measurement which can be defined as the difference between two binary strings with an equal number of digits. The resulting can be calculated by counting how many digits that needs to be flipped in order to make the two strings similar. As such, the distance increases by one for every corresponding decimal place with a different value. Mathematically, the Hamming distance can be computed by XOR operations. For example, the values in equation 2.7 and 2.8 have a Hamming distance of three (the non-matching digits have been indicated with a bold face):

$$1\,0\,\mathbf{0}\,0\,\mathbf{1}\,1\,0\,0\,\mathbf{0}\,1\,0\,0 \tag{2.7}$$

$$1\,0\,\mathbf{1}\,0\,\mathbf{0}\,1\,0\,0\,\mathbf{1}\,1\,0\,0 \tag{2.8}$$

2.3 Rotation matrices

The transformation matrices below are needed to represent the transformation between the IMU and camera centers. The principal rotation matrices can be seen below:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{2.9}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.10}$$

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.11}$$

Thus, conversion from camera body coordinates to camera fixed coordinates can be done by the following rotations:

$$R = R_z(\psi)R_y(\theta)R_x(\phi). \tag{2.12}$$

Transformation matrix is constructed by a rotation matrix and translation vector like this:

$$T = \begin{bmatrix} R11 & R12 & R13 & t1 \\ R21 & R22 & R23 & t2 \\ R31 & R32 & R33 & t3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

where t is the components of the translation vector and R denotes the components of the rotation matrix.

2.4 Perceptual aliasing

Perceptual aliasing is when different places are incorrectly perceived as the same. One potential situation where perceptual aliasing may occur is shown in Figure 2.1 which consists of three images from the harbor environment in Brattøra quay. The pier has a long and relatively monotone shape to it, and the images seem to have an almost repeating pattern. Even though similar shaped land is visible in the background of all these images, the camera (or the agent) had been moving a lot between these images. The upper left image was taken first, followed by the upper right image 27 seconds afterwards, and lastly the third image at the bottom was taken after another 31 seconds. For clarity, Figure 2.2 contains the same three images, but with bounding boxes applied to show that these images obviously were taken at different parts of the pier. This example shows a potentially extremely difficult environment to navigate using unaided monocular SLAM.

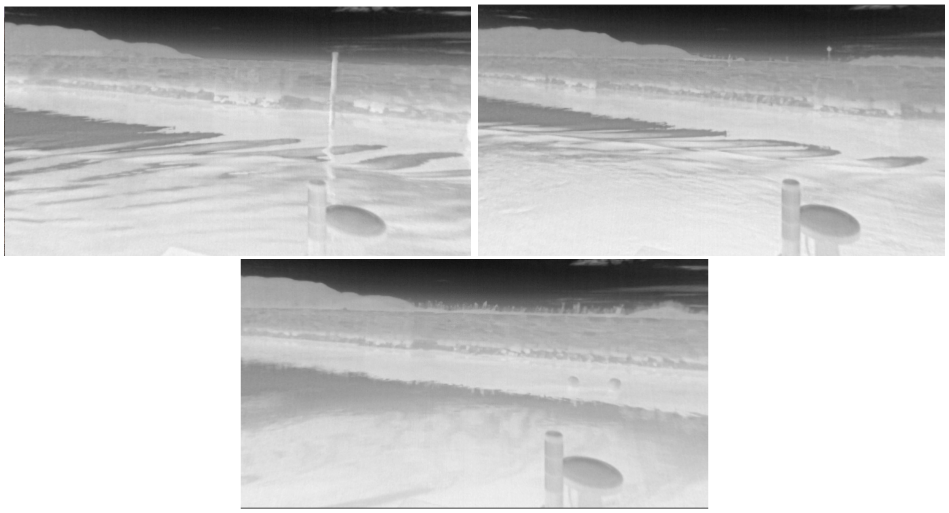


Figure 2.1: Potential perceptual aliasing example from the harbor.

Other potential situations for perceptual aliasing may for instance be in apartment and office buildings which are built nearly identically from floor to floor, or apartment to apartment, or even within a corridor consisting of a lot of flat and ”clean” surfaces on walls, floors, doors and windows, etc. Figure 2.3a shows how two similar doors at different locations can give a very similar appearance. In Figure 2.3b, a similar situation is shown for two different windows. In general, similar looking image frames may be taken close to each other or at random and completely different locations.

2.5 *k*-median clustering

k-median clustering is used for creating vocabulary trees which form an important part of the place recognition module to ORB-SLAM. *k*-median clustering is an unsupervised method for grouping data into a *k* number of groups. It works by first selecting *k* points randomly from the dataset, and these selected points will be the cluster centroids initially. Then the method calculates the Manhattan distance (1-norm) between each data point and the centroids, and then assigns the data point to the cluster with the closest centroid. When all the data points have been assigned to clusters, the centroids are recalculated such that

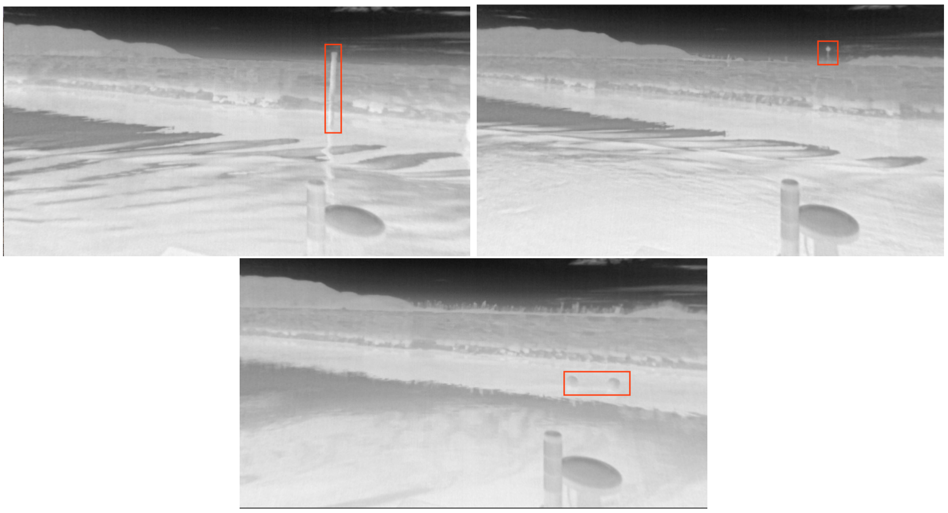
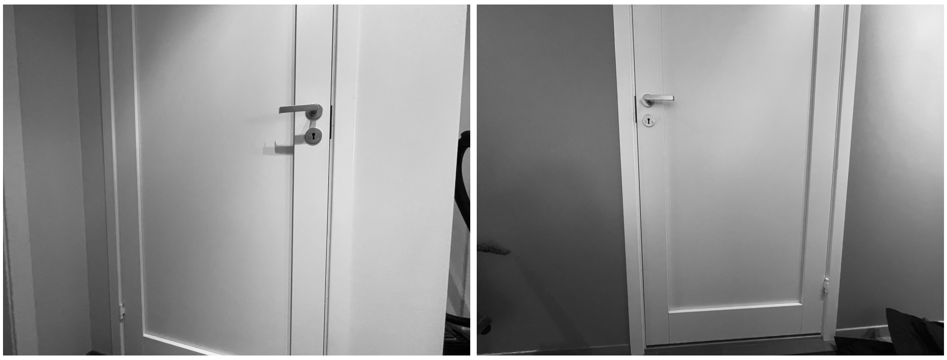


Figure 2.2: Same images as above, here with bounding boxes applied which clearly indicates that the locations are different.

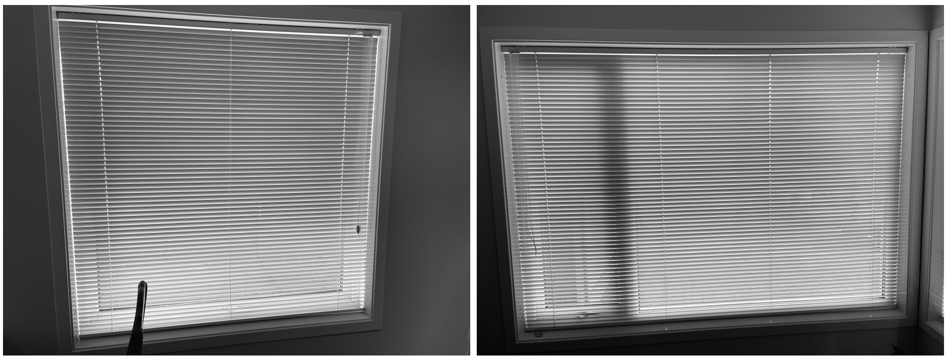
they become the medians of each cluster. This series of steps is then repeated until convergence or the maximum iterations has been reached. The *k*-median clustering algorithm is guaranteed to converge, but not necessarily to the optimal solution. *k*-means clustering is a very similar method, but uses the Euclidean distance and, as the name indicates, the mean instead of the median when calculating the clusters. An example for creating clusters in this manner with *k*=4 is show in Figure 2.4.

An alternative to randomly select the cluster centroids initially are to use the kmeans++ seeding proposed by (Arthur and Vassilvitskii, 2007). The kmeans++ seeding results in points farther away from the already chosen centers having a higher probability of being chosen as a new center. The kmeans++ method works by the following steps:

1. Choose one center randomly from the dataset.
2. The probability for choosing a data point as another center x is $\frac{D(x)^2}{\Sigma_x D(x)^2}$, where $D(x)$ is the shortest distance from a data point to a center already chosen.
3. Repeat step 2 until all k centers have been chosen.



(a) Different doors (at different locations), but with similar appearance.



(b) Different windows (at different locations), but with similar appearance.

Figure 2.3: Perceptual aliasing example 2.

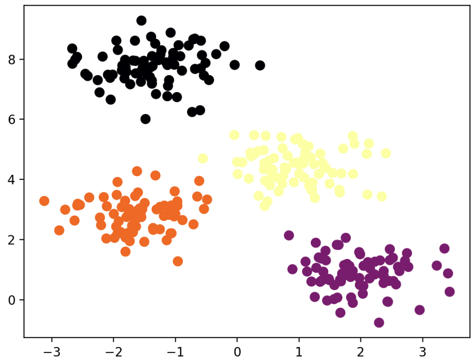


Figure 2.4: *k*-means clustering with *k*=4

Chapter 3

ORB-SLAM

Simultaneous localization and mapping (SLAM) is about finding the movement of an agent and at the same time building a 3D map of the environment. In the case when the input data is obtained from a camera sensor the task is referred to as visual SLAM (VSLAM). Two common ways to classify SLAM systems is (1) by their method of interpreting the camera input (either direct or indirect) and (2) their method of how the map is drawn and how much information is retained in the map (dense or sparse). Indirect means that the method uses features to recover the camera motion and the map, while a direct method would use pixel intensities directly. Sparse means that the method only uses a subset of the pixels in an image to create a map, while a dense method attempts to use all of the pixels in an image which results in a finer map. ORB-SLAM is an example of a sparse and indirect VSLAM algorithm.

Different SLAM approaches can also broadly be categorized as either filtering based methods or graph based methods (also called optimization based methods). Filter based methods is the classical approach, and works by representing the environment and the state of the agent as a probability density function, on which prediction and update steps are performed recursively. Extended Kalman filter and particle filter based methods are examples of filter based methods. Drawbacks of filtering based methods includes slow computation times as the dataset grows. Graph based methods are based on keyframes and uses bundle adjustment on an underlying graph structure to estimate the motion and structure. ORB-SLAM is an example of an optimization based method.Furthermore, ORB-SLAM2 and ORB-SLAM3 both uses full bundle adjustment and they are therefore global methods.

Some of the main contributions from ORB-SLAM is that it uses the same features for all of its tasks, provides real-time operation in large environments, and uses a survival of the fittest strategy when it comes to inserting new keyframes.

This chapter presents an overview of the ORB-SLAM algorithm. More details can be found in the ORB-SLAM papers (Mur-Artal et al., 2015), (Mur-Artal and Tardós, 2016) and (Campos et al., 2020), corresponding to ORB-SLAM1, ORB-SLAM2 and ORB-SLAM3 respectively. Note that a choice has been done in this thesis to refer to the first version of ORB-SLAM as ORB-SLAM1, even though it was referred to as ”ORB-SLAM” in original the paper. In this thesis, ”ORB-SLAM” (without any numbers) will refer to all three ORB-SLAM versions in common for simplicity. This chapter is outlined based on the three respective papers. First the ORB-SLAM1 algorithm is presented in section 3.1 with all the building blocks and concepts for ORB-SLAM. Then, additions that came with ORB-SLAM2 are presented in Section 3.2, and lastly additions from ORB-SLAM3 are presented in section 3.3.

3.1 ORB-SLAM1

In this section ORB-SLAM1 is presented, and the later versions of ORB-SLAM presented in Chapter 3.2 and 3.3, respectively, build on these main concepts. The outline of this ORB-SLAM1 section is based on the diagram in Figure 3.1. First, the map and place recognition modules in the center of the figure are presented. Then the three main parallel threads are presented, which are tracking, local mapping and loop closing.

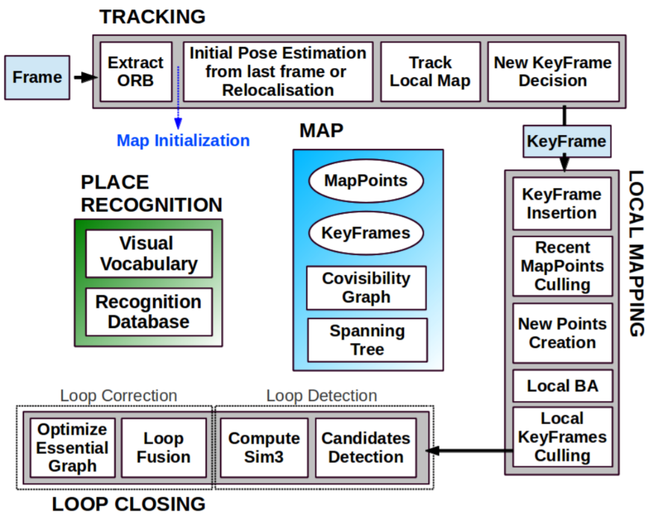


Figure 3.1: ORB-SLAM1 system threads and modules. Image from (Mur-Artal et al., 2015)

3.1.1 Map

The map module contains map points, keyframes, the covisibility graph and the spanning tree. The essential graph, which is also an important part of the map module, is explained here as well.

- **Map points block**: Map points contain information about their 3D position in a world coordinate frame. The world coordinate frame is such that the origin is set in the center of the first camera pose successfully initialized. Furthermore, the axes of the new world coordinate system will be aligned with the axes of the initial camera pose. Information about the mean viewing direction of the keyframes observing the map point is included, which can be useful for selecting new keyframes with larger differences in viewing angles. Additionally, the ORB descriptor is connected to the map point with the smallest Hamming distance (for a definition on the Hamming distance, see Section 2.2). Finally, maximum and minimum distances where it is possible to observe the map point, based on scale invariance limits for the feature, is also stored in the map points block.

- **Keyframes block**: Keyframes contain information about all ORB features detected in their image (their frame), and whether these features belong to any map points (and if so which ones). The keyframes also provide information about camera intrinsics and extrinsics. Every frame is not considered a keyframe, only certain frames satisfying certain requirements described in section 3.1.4.

- **Covisibility graph block**: In the covisibility graph the nodes are made up of the keyframes. Two keyframes are connected if they observe at least 15 of the same map points.

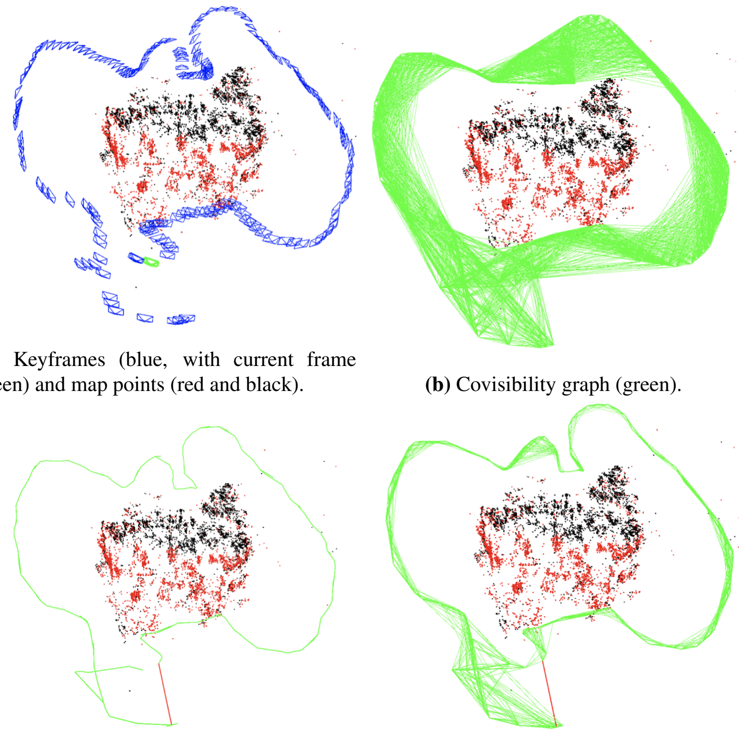
- **Spanning tree block**: The spanning tree is built by connecting a keyframe, when it is inserted, to the keyframe it has most map points in common with.

- **The essential graph**: In the essential graph the keyframes are the nodes as well, but here the edges are only formed if they are in the spanning tree, represent a loop closure, or if the two keyframes the edge is connecting have at least 100 map points in common. The essential graph is optimized in order to perform loop closing. More details on the essential graph is given in subsection 3.1.5.

The different map components are illustrated in figure 3.2. The keyframes can be seen as blue rectangles in Figure 3.2a, and the current keyframe is colored green. The map points can be seen in all subfigures, with the map points visible from the current frame being colored red and the other map points being colored black. In Figure 3.2b the covisibility graph is shown together with map points, while the keyframes here are removed. Only the green lines remain, showing the edges in the covisibility graph. The edges in the spanning tree is shown in green in Figure 3.2c. The red line indicates a loop closure connection between two keyframes. Lastly, the essential graph is shown in green in Figure 3.2d.

3.1.2 Place recognition

ORB-SLAM uses a visual vocabulary and a recognition database in order to recognize previously visited areas. The place recognition module is useful for relocalizing when track is lost and for performing loop closure.



(a) Keyframes (blue, with current frame green) and map points (red and black).

(b) Covisibility graph (green).

(c) Spanning tree (green).

(d) Essential graph (green).

Figure 3.2: Illustration of map components used in ORB-SLAM from (Mur-Artal et al., 2015)

Visual vocabulary

Bag of words (BoW) is a technique that uses a visual vocabulary to convert an image into a sparse numerical vector. A text can be represented by a bag of words by simply counting the number of times a word is mentioned. For example the text ”It was warm. It was sunny” can be represented with bag of words as ”It:2, sunny:1, warm:1, was:2”. For images however it is somewhat more difficult to understand the concept of bag of words. An intuitive explanation of this process can be shown by considering the four images in Figure 3.3 of four different types of objects. A feature detector and feature descriptor, for example ORB, can then be applied on each of these images. Examples of features and the surrounding regions can then be extracted in a sense as shown in Figure 3.4. Note that the patches shown in Figure 3.4 are not made around features obtained from a real feature detector, they are neither made in any standard size, and are only meant as an illustrative example. The next step is then to apply a feature descriptor to each patch, in the case of ORB each patch will be described with a 256 bit string. *k*-means clustering is then performed on the descriptors and each cluster will form a (visual) word. The clustering could for example result in patches being grouped together like in Figure 3.5. In this way a small vocabulary of four words is created. An image like the one shown in Figure 3.6 would then probably contain most words of type number 1 and some words of type number 3. In this way place recognition is possible as the visual words representation of one image will likely have high correspondence with an image taken at the same location.

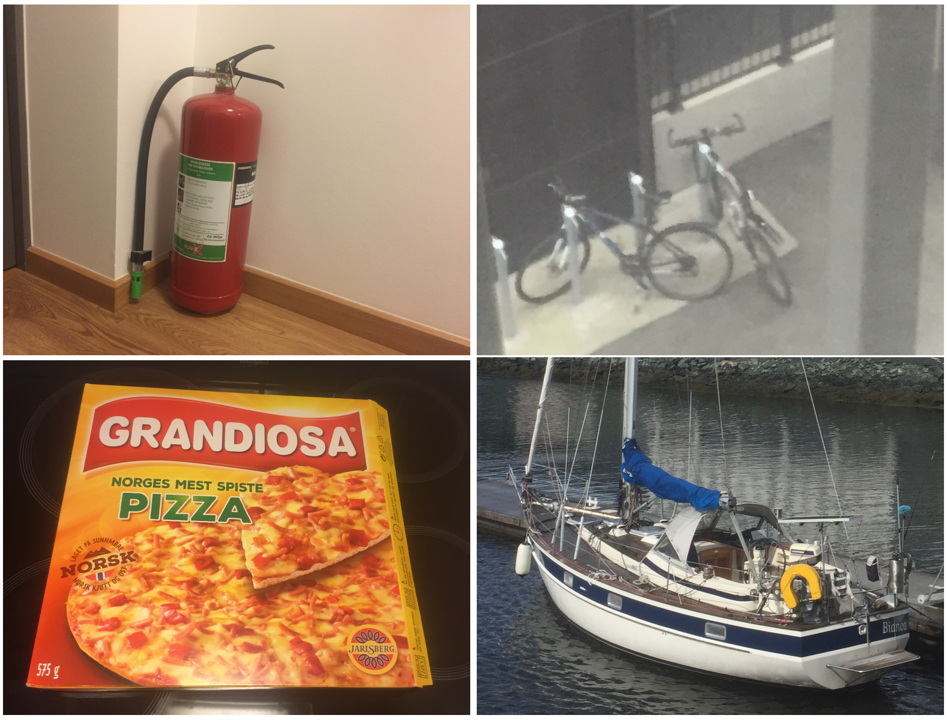


Figure 3.3: Four example images used to illustrate the visual words concept.

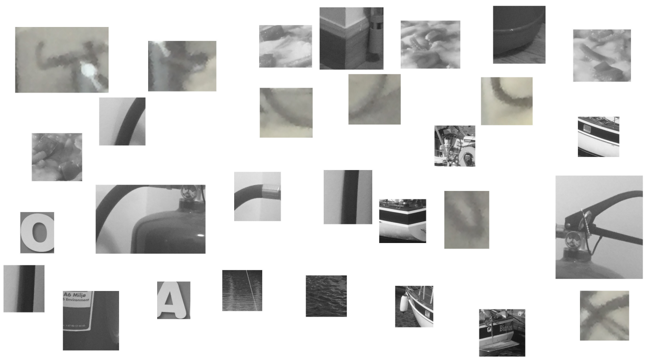
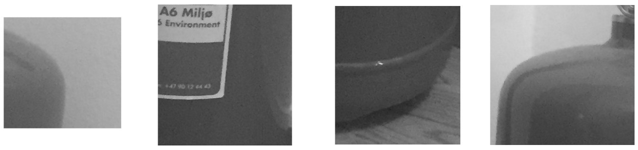
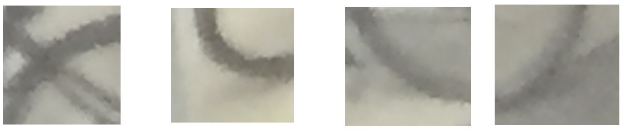


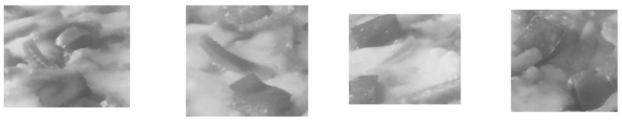
Figure 3.4: Image patches used to illustrate the visual words concept.



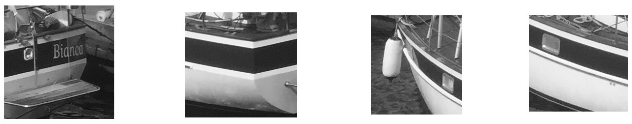
(a) Visual word 1.



(b) Visual word 2.



(c) Visual word 3.



(d) Visual word 4.

Figure 3.5: Four examples of visual words

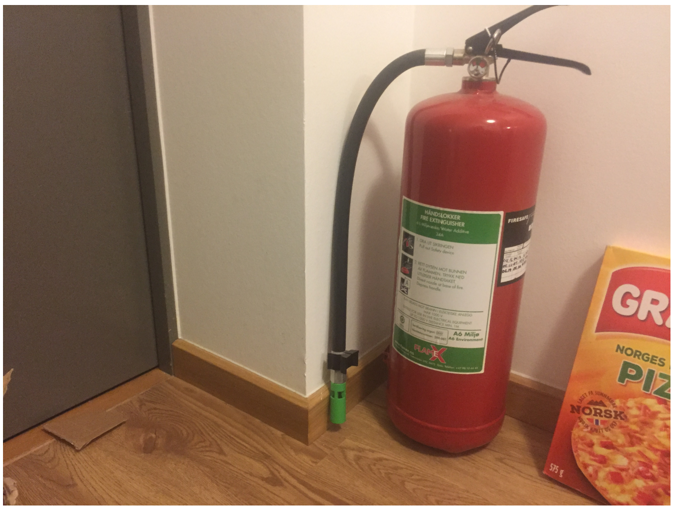


Figure 3.6: Image containing visual word 1 and visual word 3 (i.e fire extinguisher and pizza).

The vocabulary tree is structured in an hierarchical manner like shown in Figure 3.7. The example tree shown has $L_w = 2$ depth levels and a branching factor $k = 3$. The depth of a tree is the number of edges from the root node to any leaf node that maximizes this number. The branching factor is the number of children each parent node can have. In a vocabulary tree the leaf nodes will represent the words. The number of words generated will be given by $W \approx k^{L_w}$. It holds for the example shown in Figure 3.7 as $3^2 = 9$ and there are nine words generated. The inverse index is commonly used in the context of place recognition with bag of words. As shown in Figure 3.7 the inverse indexes are the words. So by looking up a word one will receive information about which images has this word and also the weight of the word in this image. DBoW2 augmented the bag of words solution to also include the use of a direct index. In the direct index, one can look up an image and find information about which nodes at level l that are ancestors for the words in the image and also local features belonging to that node.

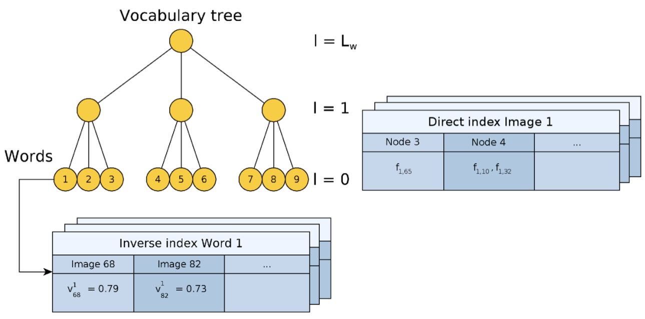


Figure 3.7: Vocabulary tree with direct and indirect indexes. Image from (Gálvez-López and Tardós, 2012).

The DBoW2 paper states that the method identifies no false positives when tested on different datasets (Gálvez-López and Tardós, 2012). To handle the case of perceptual aliasing, which is a scenario were the bag of words alone would in many situations give false positives therefore the method includes a verification step to check geometric consistency. The geometric consistency check ORB-SLAM uses for loop detection is presented in 3.1.5. Even though ORB-SLAM uses most parts of the DBoW2 repository as it is, some changes are made. For one, the distance function in ”FORB.cpp” that computes the distance between two descriptors (bit strings) is modified. ORB-SLAM has modified the way of counting bits set to ”1” to be counted in parallel instead of the Brian Kernighan’s way which DBoW2 uses. Counting bits set methods available in Anderson. The other major difference is that the vocabulary is saved on another format, the format ORB-SLAM uses. This format includes information for each node about its parent, whether it is a leaf node, the descriptor and its weight. The leaf nodes are the words in the vocabulary. The weight will be zero for nodes that not are words, the weight will be low for common words and high for words that can describe images well, those words that are more distinct.

Additionally their words are structured in a hierarchical tree for faster matching. The custom BoWs generated as part of this report are also structured as hierarchical trees. The vocabulary proposed by the ORB-SLAM developers is created offline and based on a large set of images.

Recognition Database

The recognition database uses an inverted index, which means that by looking up a visual word in the database one will find in which keyframes the word has been seen. The recognition database is maintained in real time as keyframes are inserted and removed. The visual words will stay the same, as they have been trained in an offline manner as mentioned above. Querying the database will return keyframe matches with scores higher than 75% of the best score. These matches are good candidates to use for place recognition.

3.1.3 Tracking

As mentioned in chapter 1, ORB-SLAM runs three main threads in parallel. The first of these is the tracking thread. The tracking thread is responsible for extracting ORB features, localizing the camera and deciding if the frame should be a keyframe.

Inserting an image frame

The first step is to insert an image frame into the tracking thread. The images can be either:

- Normal color images (RGB) or black and white (BW).
- Monocular, stereo or RGB-D.

Monocular thermal infrared black and white images are the frames used in this thesis.

Extracting ORB features

After a new image frame is inserted, the next step in the tracking thread is to extract the ORB features from this new frame. One of the main contributions of ORB-SLAM is as mentioned presenting a SLAM algorithm which uses the same features for all tasks: Tracking, mapping, relocalization and loop closing. The features used for all these tasks are the ORB features extracted here. ORB features are known for being very fast to compute, faster than SIFT and SURF for example. The ORB features are also fast to match and uses a binary descriptor that provides good invariance to both viewpoint and illumination (Mur-Artal et al., 2015).

ORB is based on the features from accelerated segment test (FAST) feature detector (Rosten et al., 2010) and the BRIEF feature descriptor (Calonder et al., 2010), but with many modifications in order to enhance performance (Mordvintsev and Abid, 2013).

The ORB feature detector

The ORB feature detector is based on the FAST feature detector for finding the candidates for features in an image. The features are in this cases corners. The FAST corner detector considers each pixel in the image as a candidate. Figure 3.8 shows how for each pixel a Bresenham circle of radius 3 is created around the current pixel p. The pixel p is detected as a corner if 12 contiguous pixels in the circle of 16 are all brighter than the intensity of p plus some threshold, or darker than the intensity of p minus some threshold. In figure 3.8 the pixel p is detected as a corner, due to the fact that there are 12 contiguous pixels that are brighter than p plus some threshold. The dashed line in the figure represents these contiguous pixels that meet the requirement.

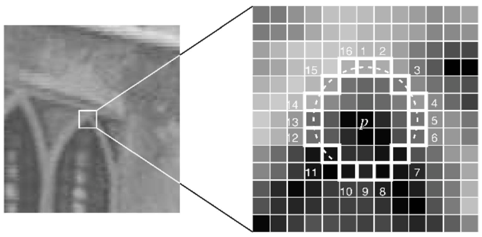


Figure 3.8: The FAST corner detector applied on a pixel p. Image from (Rosten et al., 2010).

After the FAST algorithm has found the candidates, ORB then uses the Harris corner detector to pick the best n corners among them. The Harris corner detector takes into account intensity changes in all directions when determining a score of how likely a pixel is a corner, see the paper (Harris and Stephens, 1988) for more details on how these calculations are done. Another important trait of the ORB feature detector is that it uses a pyramid multi-scale representation which provides robustness in terms of scale invariance. The multi-scale pyramid is obtained by iterations of smoothing and subsampling. The developers of the ORB algorithm also made a modification in order to achieve rotation invariance. The idea is to consider an area around the feature and compute the intensity weighted centroid for this area. The vector from the feature point to the centroid determines the orientation.

The ORB detector detects more corners in higher resolution images compared to in lower resolution images, and the algorithm also divides the images into a grid and tries to extract a certain amount of corners per cell in order to get more distributed features.

The ORB feature descriptor

The ORB feature descriptor is based on the BRIEF descriptor, but since BRIEF performs poorly with rotation (Mordvintsev and Abid, 2013), the developers made a modification to BRIEF in order to steer BRIEF in the orientation of the keypoints. The orientation of the keypoints is the vector between the feature and the intensity weighted centroid mentioned above. The modification is called rBRIEF, which is short for rotation aware binary robust independent elementary features.

The BRIEF descriptor describes an image patch as a bit string based on a set of binary intensity tests. A binary test τ is defined by:

$$\tau(p; x, y) := \begin{cases} 1 & I(p(x)) < I(p(y)) \\ 0 & I(p(x)) \geq I(p(y)) \end{cases} \tag{3.1}$$

where $I(p(x))$ is the intensity of the image patch p at a point x. The feature is then described by a vector of n binary tests:

$$f_n(p) := \Sigma_{1 \leq i \leq n} \ 2^{i-1} \tau(p; x_i, y_i) \tag{3.2}$$

The tests are distributed according to the Gaussian distribution around the feature it aims to describe. The BRIEF descriptor has a length of 256 bit, which means that $n = 256$ in equation 3.2.

The points $\mathbf{x}$ and $\mathbf{y}$ that are used in the n binary tests performed, are collected in the matrix S written below:

$$S = \begin{bmatrix} x_1 & ... & x_n \\ y_1 & ... & y_n \end{bmatrix} \tag{3.3}$$

The steered version of S, S_θ, is then found by:

$$S_\theta = R_\theta S \tag{3.4}$$

where θ is the orientation of the patch and R_θ is the corresponding rotation matrix. A lookup table with precomputed values at increments of 12 degrees (0.21 rad) are made which increases the speed when computing the descriptors.

The BRIEF descriptor has the nice property of having a large variance, which means that the features are easier to distinguish from each other. However, using the steered version of the BRIEF descriptor reduces this variance as it is steering BRIEF in the feature direction, making the descriptions more similar. To solve this the rBRIEF algorithm runs a greedy search to find the binary tests with highest variance, additionally the greedy search looks for the binary tests with means closest to 0.5 and most uncorrelated binary tests, as these are desirable properties as well.

Map initialization

The algorithm ORB-SLAM proposes in order to initialize and get an initial set of map points is presented in this section.

- The first step is to look for feature matches between current frame and reference frame. The algorithm only proceeds if a sufficient amount of matches is found, if not the reference frame is reset.

- The second step is completed by computing two different models in parallel. One computes the homography H between current and reference frame as a model, here the underlying assumption is a planar scene. The other computes fundamental matrix F between the current and reference frame as the model, in this case the underlying assumption is a non-planar scene. The homography is calculated by:

  $$x_c = H \cdot x_r \tag{3.5}$$

  where c refers to the current frame, r refers ti the reference frame and H to the homography. The calculation is done by using 4 point correspondences and the normalized DLT algorithm combined with RANSAC. The fundamental matrix is computed by the following equation:

  $$x_c^\top \cdot F \cdot x_r = 0. \tag{3.6}$$

  The fundamental matrix F is computed with 8 point correspondences and the 8-point algorithm. The number of iterations RANSAC uses to compute a model is fixed and the model obtained at each iteration is evaluated by the following score:

  $$\begin{aligned} S = \Sigma_i(\rho(d^2_{cr}(\mathbf{x}^i_c, \mathbf{x}^i_r, model)) + \\ \rho(d^2_{rc}(\mathbf{x}^i_c, \mathbf{x}^i_r, model))) \end{aligned} \tag{3.7}$$

  where i represents each one of the feature matches between the current frame and the reference frame from step 1, model is either the homography or fundamental matrix, and ρ is defined as the following function:

  $$\rho(d^2) = \begin{cases} \Gamma - d^2, & d^2 < T \\ 0, & d^2 \geq T. \end{cases} \tag{3.8}$$

  The T denotes the threshold for whether the feature match is considered an outlier with respect to the proposed model. For the homography ORB-SLAM authors set $T = 4.99$, correspondingly for the fundamental matrix it is set $T = 3.84$. In both cases the value is set based on a 95% confidence χ^2 test. Γ in the equation above is set equal to the threshold for the homography. If not enough feature matches are inliners, in other words if not enough feature matches confirms the model, then the algorithm starts at step 1.

- The next step is then to decide which of the two models to use, homography or fundamental matrix. The homography works best for, as mentioned, planar scenes, while the fundamental matrix works best for non-planar scenes. The ORB-SLAM method uses the following heuristic in order to select the best model:

  $$R = \frac{S_{homograhpy}}{S_{homograhpy} + S_{fundamental}}. \tag{3.9}$$

  The homography is selected if R is greater than 0.45, if not the fundamental matrix model is selected.

- The fourth step is to estimate the camera motion. For the homography case, 8 hypotheses are found by method proposed in Faugeras and Lustman (1988). The eight camera motion hypothesis are then triangulated and if one solution clearly better than the others see points with more parallax in front of both cameras then this camera motion hypothesis will be used. If not, the algorithm will start over from the first step. In the case of the fundamental matrix, the essential matrix E is estimated:

  $$E = K^{\top} \cdot F \cdot K \tag{3.10}$$

  where K is the intrinsics matrix. From the essential matrix, it is possible to find 4 motion hypothesis by the use of singular value decomposition. Triangulation and selecting is the done in the same manner as for the homography.

- Fifth and final step is to perform full bundle adjustment to optimize the solution.

Initial pose estimation from last frame or relocalization

There are two possible cases for further processing depending on whether tracking in the previous frame was successful or not. The ORB-SLAM algorithm will, if tracking is lost, convert the image into a bag of words and search the recognition database in order to attempt to find keyframes similar to the current in order to perform relocalization. If a good camera pose candidate is found then the tracking procedure is able to start again. If on the other hand the tracking was successful, a velocity motion model is used to estimate the position and orientation of the camera. Feature matches are then searched for in the area predicted by the motion model, and if not found the area is expanded. If any feature matches are found they will be used in order to optimize the rotation and the translation estimates of the camera. This optimization of pose is performed with motion only bundle adjustment as explained below:

Motion-only bundle adjustment Motion-only bundle adjustment is performed in the tracking thread to find the rotation, $\mathbf{R}$, and the translation, $\mathbf{t}$, that minimizes the reprojection error. The motion-only BA equation which ORB-SLAM uses is expressed in equation 3.11:

$$\{\mathbf{R}, \mathbf{t}\} = arg \min_{\mathbf{R},\mathbf{t}} \sum_{i \in \mathcal{X}} \rho(||\mathbf{x}^i - \pi(\mathbf{R}\mathbf{X}^i + \mathbf{t})||^2_{\Sigma}) \tag{3.11}$$

where $\mathbf{X}^i$ is the 3D points, $\mathbf{x}^i$ is the 2D feature points, $\mathcal{X}$ is the set of all matches, ρ is the robust Huber cost function, Σ is the covariance matrix associated to the scale of keypoints and π is the projection function defined to be:

$$\pi(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}) = \begin{bmatrix} f_x \dfrac{X}{Z} + c_x \\ f_y \dfrac{Y}{Z} + c_y \end{bmatrix}. \tag{3.12}$$

Local map tracking

The next step in the tracking pipeline is to track the local map. The local map consists of the map points seen in the set of keyframes with features in common with the current frame. The local map also consists of map points seen in keyframes connected to the set of keyframes above in the covisibility graph. For each map point in this local map, the map point is first projected onto the image. Three reasons can then result in the map point being skipped: The first is if the projection is outside the image, second if the dot product between the current viewing direction and the map point's average viewing direction is less than 0.5, and third if the distance between the camera center and the map point is outside the scale region for the map point. If none of the reasons to skip the map point is satisfied then a descriptor of the map point is compared with unmatched ORB features in the current frame.

New keyframe decision

The final step in the tracking pipeline is to decide if the keyframe should be inserted. In order to prevent having a lot of similar keyframes, it is required in order to be a keyframe to have more than 10% unique points compared to the most similar keyframe, also at least 20 frames must have passed since a keyframe was inserted. Another requirement for the current frame to be a keyframe is that at least 20 frames have passed after relocalization, to ensure some correctness for the relocalization process. The final requirement for the keyframe insertion is that at least 50 points are tracked, to ensure some minimum number of information from the frame. Note that ORB-SLAM attempts to detect 1000 corners in an image with resolution 512×384 and even more in images with higher resolution. So for the 512 x 384 resolution 5% of the corners are required to be tracked. ORB-SLAM is known for accepting a lot of keyframes initially, using the so called survival of the fittest strategy. This strategy is very advantageous in the situation when the camera moves quickly, in particular when rotating. The ORB-SLAM method could in such a case lose track if the keyframe insertion is too restrictive as then the previous keyframe and the new keyframe would have no or very little overlap. However as the ORB-SLAM method inserts more keyframes compared to other SLAM methods it is more robust in cases of abrupt changes in scenery.

3.1.4 Local mapping

The local mapping thread is the second main thread in ORB SLAM and contains the steps keyframe insertion, recent map points culling, new points creation, local BA and local keyframe culling like shown in Figure 3.9.

Keyframe insertion

The first step in the local mapping thread is to insert the keyframes that satisfies the criteria presented in the final step of the tracking thread. The covisibility graph is updated to include the new keyframe and connect edges between the new keyframe and keyframes with a sufficient amount of shared map points. The spanning tree is also updated, and the new keyframe is here connected to the keyframe with the most shared map points. Additionally a bag of words representation of the keyframe is created which can be used for place recognition later on if the same area is revisited.

Culling recent map points

The map points that are recently added need to meet some requirements in order to not be removed. This step is performed as an attempt to remove wrongly estimated map points. The first requirement is that the map point must be visible in 25% of the frames where the map point should be visible until three keyframes are created. Additionally after one new keyframe is inserted, the map point is required to be observed in minimum three keyframes. If a map point is observed in less than three keyframes it will be removed, and this requirement also applies to map points that has been present in the map for a longer amount of time.

Creating new points

The next step in the local mapping thread is concerned with creating new map points. The new map point candidates are the keypoints in the current keyframe which are unmatched. For every one of these unmatched keypoints, it is attempted to find a match among the unmatched keypoints in the keyframes connected to the current keyframe. A match is searched for by using a constrained brute force matcher. If a match is found a 3D point will be triangulated and added to the map, but it is possible that it will be removed later on as explained above in the previous subsection.

Local bundle adjustment

Local bundle adjustment is performed in the local mapping thread to find the 3D points, $\mathbf{X}^i$, the rotation $\mathbf{R}$, and the translation, $\mathbf{t}$, that minimizes the reprojection error. The local BA equation ORB-SLAM uses is expressed as:

$$\{\mathbf{X}^i, \mathbf{R}_l, \mathbf{t}_l \mid i \in \mathcal{P}_L, l \in \mathcal{K}_L\} = arg \min_{\mathbf{X}^i, \mathbf{R}, \mathbf{t}} \sum_{k \in \mathcal{K}_L \cup \mathcal{K}_F} \sum_{j \in \mathcal{X}_k} \rho(||\mathbf{x}^j - \pi(\mathbf{R}_k \mathbf{X}^j + \mathbf{t}_k)||^2_\Sigma) \quad (3.13)$$

where $\mathcal{K}_L$ is the set of covisible keyframes the local BA optimizes and $\mathcal{P}_L$ is all the points seen in those keyframes. The set of keyframes not in $\mathcal{K}_L$ is denoted $\mathcal{K}_F$ and these will contribute to the cost function if a point in $\mathcal{P}_L$ is observed in the set of those keyframes. $\mathcal{X}_k$ denotes matches between points in $\mathcal{P}_L$ and keypoints in a keyframe k.

Local keyframe culling

This final step in the local mapping thread is about removing redundant keyframes. As mentioned, ORB-SLAM is using a survival of the fittest strategy so in addition to having low requirements for accepting keyframes initially, a lot of keyframes are at this step removed, so that only the necessary keyframes are kept. The reason for culling keyframes is to keep the problem at a manageable size as the bundle adjustment complexity grows with the number of keyframes. The criteria for removing keyframes is that it will be removed if 90% of its map points have been seen by at least three other keyframes at the same or better accuracy.

3.1.5 Loop closing

The third main thread is the loop closing, and this thread is divided into four blocks as seen in Figure 3.9. The two first are used for loop detection by querying the database and compute the similarity transform, while the two last are used for loop correction by fusing the loop and optimizing the essential graph.

Candidates detection

To find the loop closure candidates, keyframes which are connected to the current keyframe in the covisibility graph are considered. The score to the connected keyframe which is lowest in terms of being similar to the current keyframe is denoted s_{min}. The database is then queried for keyframes with a score higher than s_{min}. If there are found at least three loop closure candidates that are connected in the covisibility graph, then the loop closure candidate can be accepted. Keyframes already connected to the current keyframe are not considered in the query as they are already connected and therefore are not candidates for loop closure.

Compute Sim3

The similarity transform between the current keyframe and the loop closure keyframe is computed in order to correct for the drift that has been accumulated. To compute the similarity transform, the random sample consensus (RANSAC) algorithm is used. To use the loop closure keyframe, the similarity transform must have a sufficient amount of inliers.

Loop fusion

The third step in the loop closing thread is concerned with updating information as a consequence of a loop detection. The covisibility graph is updated to connect the keyframes involved in the loop detection, such that the loop closure is visible in the covisibility graph. All map points from the current keyframe should be merged with other map points from the keyframes involved in the loop detection in the case that they are already accounted for. Also in this step, the similarity transform is applied to the current keyframes and those connected, in order to correct for drift.

Optimize essential graph

The final step in the loop closing thread is to optimize the essential graph by using pose graph optimization. ORB-SLAM uses the g2o framework by Kümmerle et al. (2011) for performing this optimization. The Levenberg-Marquardt (LM) method is used for this purpose. The LM method is a combination of the Gauss Newton method and the gradient descent method. The LM iteration is written:

$$\mathbf{x}_k = \mathbf{x}_{k-1} - ((J_r(\mathbf{x}_{k-1}))^\top (J_r(\mathbf{x}_{k-1})) + \lambda_k I)^{-1} (\nabla f(\mathbf{x}_{k-1})) \quad k = 1, 2, 3, \ldots \tag{3.14}$$

where J_r is the Jacobian matrix of the residuals with respect to x and λ_k is a non-negative scalar. For larger λ the method is more similar to gradient descent while for smaller λ the method is more similar to the Gauss Newton iteration. It is therefore called a trust region method as it controls in which regions it trust the hessian approximation.

3.2 Changes in ORB-SLAM2

This chapter will focus on monocular ORB-SLAM, which was used in this project. There are a lot of similarities between the first and the second version of ORB-SLAM. However, one difference is that ORB-SLAM2 in addition to monocular cameras also provides support for stereo and RGB-D cameras. Another difference is that ORB-SLAM2 performs full bundle adjustment (BA) after the loop closing thread. The system threads and modules for ORB-SLAM2 is presented in Figure 3.9, and how the input is pre-processed is shown in Figure 3.10. Note that Figure 3.10 does not describe the input pre-processing for monocular video. The monocular pre-processing has only one image from which ORB features are extracted, there is no stereo matching or generation of stereo coordinates, and there are only generated mono key points.

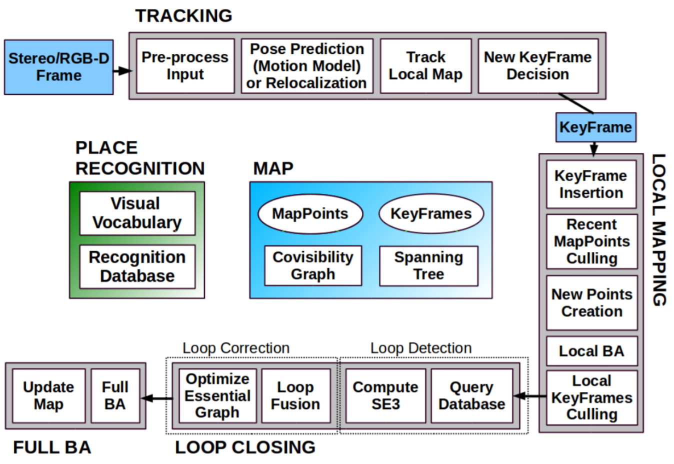


Figure 3.9: ORB-SLAM2 system threads and modules. Image from (Mur-Artal and Tardós, 2016).

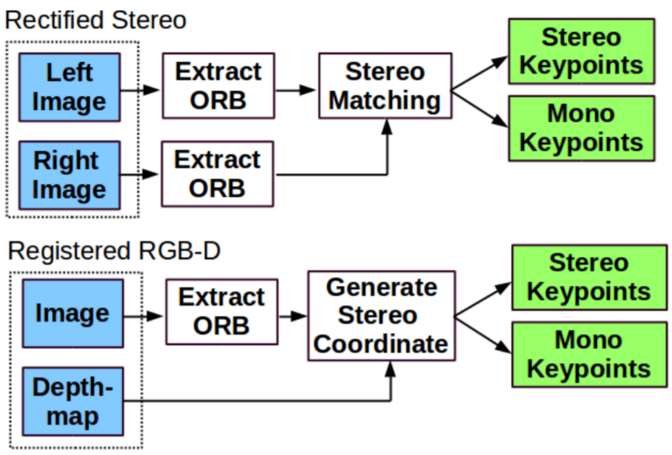


Figure 3.10: ORB-SLAM2 input pre-processing. Image from (Mur-Artal and Tardós, 2016).

3.2.1 Stereo vision

In the input pre-processing step shown in Figure 3.10 there will only be performed extraction of oriented FAST rotated BRIEF (ORB) features and these will be the mono keypoints. In other words there will be no stereo matching and stereo keypoints.

After pre-processing, the next step in the tracking thread is therefore to extract the ORB features from the new image frame inserted

3.2.2 Full bundle adjustment

Full bundle adjustment is used to obtain a global consistent solution as a final step after the loop closure thread. The full bundle adjustment equation is similar to the local bundle adjustment equation, the only difference is that all keyframes and all points are used. The equation is written below:

$$\{\mathbf{X}^i, \mathbf{R}, \mathbf{t} \mid i \in \mathcal{P}\} = arg \min_{\mathbf{X}^i, \mathbf{R}, \mathbf{t}} \sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{X}_k} \rho(||\mathbf{x}^j - \pi(\mathbf{R}_k \mathbf{X}^j + \mathbf{t}_k)||^2_\Sigma) \tag{3.15}$$

where $\mathcal{P}$ is the set of points, $\mathcal{X}$ the set of feature matches and $\mathcal{K}$ is the set of all keyframes.

The full bundle adjustment process runs in its own thread so when it is complete, new keyframes and points have been inserted by the other threads. To update the map and merge these versions the keyframes that are inserted after full BA started will be transformed in the same manner as the keyframe they are connected to in the spanning tree (the keyframe in which they have most map points in common).

3.3 Changes in ORB-SLAM3

The overview of the ORB-SLAM system can be seen in Figure 3.11. The IMU integration and the use of an ATLAS are the main novelties.

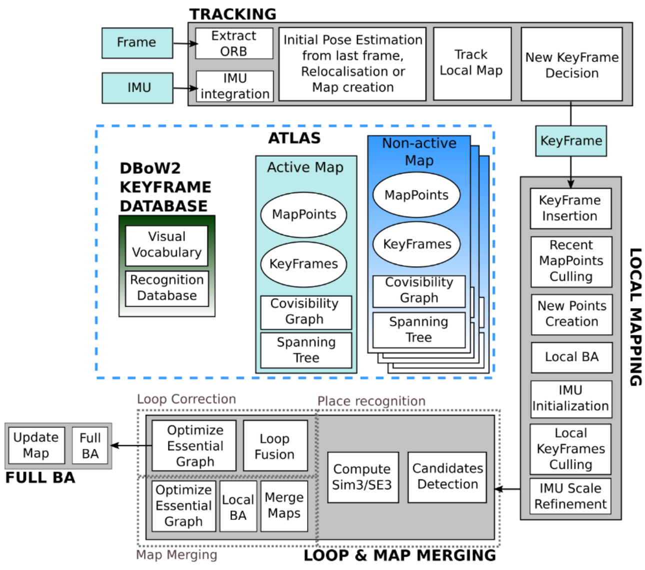


Figure 3.11: ORB-SLAM3 system threads and modules. Image from (Campos et al., 2020)

3.3.1 Novel relocalization

ORB-SLAM3 uses an ATLAS, meaning that it is now possible with several maps. When track is lost a new initialization process starts and if successful, a new map is created. In this way ORB-SLAM3 is able to keep track after difficult tracking scenarios and can merge the map if the place is revisited. ORB-SLAM3 has three types of data association: short-term, mid-term and long-term. ORB-SLAM3 still use the DBoW2 place recognition system as well and the built-in vocabulary is the same as for ORB-SLAM2. However, they have done two novel changes. One is that when a new keyframe is created the place recognition thread looks for matches in all the keyframes in the atlas. If found in the current map, loop closure will be attempted, while if found in another map, merging of maps will be attempted. The second novelty is how ORB-SLAM three finds the relative pose and considers the neighbours to the matching keyframe in order to better perform loop closing and map merging.

3.3.2 IMU integration

The visual inertial system ORB-SLAM3 presents uses the maximum a posteriori (MAP) estimate, which can be written as a minimization problem if one takes the negative logarithm and assumes Gaussian error. The system takes into account the transformation matrix between the visual and the inertial system as well as noise parameters to the accelerometer and the gyro. The IMU measurements are pre-integrated and the residuals are computed for rotation, velocity and position. The inertial residuals are then used to compute the reprojection error. The reprojection error for inertial are combined with the reprojection error for the visual and solved as a minimization problem. More details on the IMU integration and ORB-SLAM3 in general can be found in (Campos et al., 2020).

Chapter 4

Sensors

4.1 Infrared imaging

Infrared vision in general refers to ways that animals or machines can gain a visual perception by detection of infrared radiation. Many variations of subdivisions for the infrared spectrum exist, but the convention used here is summarized in Table 4.1 (Byrnes, 2009). While human vision is limited to a small portion of the electromagnetic spectrum referred to as visible light which has wavelengths between about $0.38 - 0.74\mu$m, the infrared spectrum spans (depending on author) approximately $0.75 - 1000\mu$m. Thermal imaging, or *thermography*, is a subdivision of infrared vision which spans about $3 - 15\mu$m, but most commonly refers to the portion called long-wavelength (LWIR) division between about $8 - 15\mu$m. The long-wavelength division roughly coincides with the $7.5 - 13.5\mu$m spectral range of the FLIR Boson camera used on Milliampere and the FLIR Tau 2 camera used for capturing images used in the new vocabulary. Note that the term thermal imaging is also often applied to the mid-wavelength (MWIR) division between about $3 - 8\mu$m, but less commonly.

Infrared radiation is emitted from all black bodies with temperature above zero kelvin. LWIR is particularly useful for creating thermograms in the Earth's climate of for example terrain, water bodies, vehicles and life. This can be seen intuitively from the wavelengths

Subdivision	**Abbr.**	**Wavelength (μm)**	**Temp (K)**	**Temp (°C)**
Near infrared	NIR	$0.75 - 1.4$	$2070 - 3864$	1797 to 3591
Short-wavelength	SWIR	$1.4 - 3$	$966 - 2070$	693 to 1797
Mid-wavelength	MWIR	$3 - 8$	$362 - 966$	89 to 693
Long-wavelength	LWIR	$8 - 15$	$193 - 362$	-80 to 89
Far infrared	FIR	$15 - 1000$	$3 - 193$	-270.15 to -80.15

Table 4.1: Subdivisions of infrared light.

of LWIR coinciding with peak spectral wavelengths of black body surfaces having temperatures between -80 and 89 °C, according to Wien's displacement law. This makes LWIR thermography suitable for observing many enviroments on the Earth, independent of any external illumination being present or not.

A brief overview of different subdivisions of infrared and their typical uses are:

- **Near infrared**: Used in image intensifiers, such as in night-vision devices, and naturally in some animals (note: some snakes also have *thermal* vision). NIR is also used in near-infrared spectroscopy for medical purposes, and in fiber optic telecommunication systems.
- **Short-wavelength**: Used for optical long distance telecommunication.
- **Mid-wavelength**: Referred to as one of the two types of thermal infrared. MWIR cameras are known for being the type of thermal imaging capable of achieving the longest range detection, and are used in heat seeking missiles to detect hot exhausts.
- **Long-wavelength**: Also referred to as thermal infrared. Many civilian and military applications, such as identification and surveillance.
- **Far infrared**: Used in spectroscopy and astronomy. Examples include: Far-infrared lasers, which can be used in infrared spectroscopy to identify chemical substances, and passive infrared sensors for detecting general human movement.

Thermal imaging uses passive sensors, which means that they are eye safe and can operate undetected. It maps surface temperatures, and is good at detecting objects with different temperatures than their surroundings. Thermal imaging has proven particularly useful for operation of vehicles in fog or during the darkness of the night, and can be used as an alternative or in combination with traditional image intensifier night vision devices.

The term electro-optical sensor (EO) can broadly refer to any type of optoelectronic sensor, and therefore any sensor that senses either visible light or even invisible light such as infrared or ultraviolet radiation. It should be noted, however, that the company FLIR Systems (which has delivered the Boson 640 thermal video cameras used in this project) istead uses EO to specifically refer to image sensors which operate in the visual spectrum, and IR to specifically refer to sensors that operate in the thermal imaging spectrum.

Thermal images can either be constructed radiometric (thermographic) or non-radiometric. In a radiometric thermogram, every pixel contains information of the temperature on the corresponding point of the observed surface. Non-radiometric images on the other hand merely contains a visualization of the relative temperature field. The FLIR Systems cameras used in this thesis have options of outputting either 14-bit radiometric or 8-bit non-radiometric images. The 8-bit images are automatically adjusted and normalized to provide good contrast.

Thermal cameras, and especially uncooled thermal cameras as used in this project, have a time constant due to the time it takes for every pixel (every small thermometer) in the image plane to update their temperature reading. Fast movement or panning of the thermal camera can therefore result in motion blur. This effect can be somewhat reminiscent of the lagging which can be seen when using long exposure times on common visual spectrum video or still cameras, for example when shooting in low light situations. Visual spectrum cameras suffer less from this problem if there is sufficient light. Thermal cameras are however not dependent on lighting, and the motion blir effect will therefore be approximately the same on day and night.

For the thermal camera calibration procedure and resulting data used as a basis for thermal camera parameters in this thesis, see (Hølland, 2019).

4.2 IMU

An inertial measurement unit (IMU) commonly refers to electronic devices which use accelerometers and gyroscopes to calculate pose (position and orientation) and velocity (speed and direction). IMUs are often part of an inertial navigation system (INS). INS systems use dead reckoning, which provides the relative position since the start of the measurement. Dead reckoning is susceptible to noise and calibration errors, and the deviation from the true position can grow significantly over time. Various methods can be used to correct for drift, for example by using GNSS or stereo VSLAM. Good noise models and careful calibration can help to achieve good IMU performance.

The advantage of an inertial navigation system is that it in principle does not have to rely on any external information or references such as from a GNSS, a camera or other types of sensors. It is therefore perhaps a common belief that IMUs are more robust against jamming than for example GNSS based systems. In the recent years however, there have been several demonstrations on how sensors can be jammed or hacked for example using acoustics, radio frequency (RF) or lasers to disturb or even control outputs from sensors (Yan et al., 2020). For example, resonant acoustic attacks have been demonstrated on capacitive MEMS accelerometers, enabling attackers to spoof systems who blindly trust sensors outputs (Trippel et al., 2017).

Chapter 5

Implementation

This chapter presents the implementation for the ORB-SLAM algorithm based on thermal infrared data. This includes ORB-SLAM2 as well as ORB-SLAM3 with and without IMU, all implemented with ROS. The hardware and software setup used can be found in this chapter, as well as calibration parameters for the thermal cameras. Implementation details for the augmentations performed on the algorithm, cropping of the video as well as the attempt to train a new bag of words more suited to thermal data, are also presented in this chapter. This chapter also explains how the custom thermal dataset was created. Lastly, this chapter presents details on how the plots comparing the ORB-SLAM trajectory with the GNSS trajectory was made.

5.1 The video files

The test dataset consisted of 3 rosbag-files recorded in Brattøra, Trondheim, on 11 November 2019 during daytime between 10:00 and 14:00 in slightly overcast weather. Each of the files consisted of rosbag topics containing thermal videos covering 5 different angles from the center of the boat, as well as data for GNSS, IMU, radar, lidar and more. The boat is non-stationary in all of the selected videos. Table 5.1 shows the filenames as well as a numbering scheme which is used here for easier reference to each of the files. The scenery in the harbor environment contains quays, other stationary boats, office buildings, a pier with a daymark, and persons walking onshore.

Video	**Filename**
1	2019-11-07-13-51-02.bag
2	multitarget_2019-11-07-11-59-39.bag
3	multitarget_2019-11-07-12-11-49.bag
4	merged_cbf_s1_2020-12-02-12-16-31.bag
5	merged_cbf_s2_2020-12-02-12-23-52.bag
6	merged_cr2_2020-12-02-10-24-06.bag
7	merged_cr2_2020-12-02-10-32-12.bag
8	merged_cl2_2020-12-02-10-27-47.bag
9	merged_cl2_2020-12-02-10-37-46.bag
10	merged_f1_2020-12-02-10-41-25.bag

Table 5.1: Video files overview.

5.2 Hardware and software used

The hardware components in the personal computer relevant to this project and software installed are summarized in Table 5.2. Description of the equipment used on Milliampere, such as the thermal camera, GNSS and IMU, can be found in 5.3. Additionally, a description of the thermal camera used for creating our own thermal data set can be found in 5.4.

CPU	AMD Ryzen 5 3600, 6-core, 12-Thread, 3.6 GHz, 65 W
RAM	Corsair Vengeance LPX, 16 GB DDR4 (2x8 GB), 3600 MHz, PC4-28800, CL18, XMP 2.0
Motherboard	ASUS ROG Strix B550-I
GPU	MSI GeForce GTX 1660 SUPER VENTUS XS OC, 6GB GDDR6, Turing
Operating system	Ubuntu 20.04.1 LTS, 64-bit.
ORB-SLAM2	Downloaded and used ORB-SLAM2 from GitHub user raulmur. Version f2e6f51, 11 Oct 2017. Several bugs needed to be fixed due to version incompatibilities.
ORB-SLAM3	Downloaded and used ORB-SLAM3 from GitHub user richard-elvira. Version ef97841on, 7 Sep 2020. Several bugs needed to be fixed due to version incompatibilities.
Pangolin	Version 86eb497 on 2020 May 19
OpenCV	OpenCV 4.2.0
Eigen	3.3.7
ROS 1	Robot operating system (ROS) for Ubuntu, Noetic version. (Open Source Robotics Foundation, 2018).
ROS 1 packages	ROS vision_msgs package (Open Source Robotics Foundation, 2013).

Table 5.2: Hardware and software on personal computer.

Thermal camera	FLIR Boson 640, 4.9 mm focal length (EFL, 95° (HFOV)).
GNSS	Vector VS330.
IMU	Xsens MTi 10-series (MTI-20-2A5G4-DK). This is an Xsens 4th generation MTi-20 vertical reference unit (VRU).)

Table 5.3: Hardware used on Milliampere.

Thermal camera	FLIR Tau 2 640, f/1.25, 19 mm focal length, 32x26 deg FoV (h x v), 0.895 mrad IFOV, 153 mm minimum focus distance.
Interface	ThermalCapture Grabber USB interface.
Laptop	Macbook Pro 13 Mid 2012, A1278, 4 GB RAM, 120 GB SSD.

Table 5.4: Hardware used for capturing vocabulary dataset.

5.3 Camera calibration

The camera calibration parameters used are based on the work of (Hølland, 2019) which had shown better results with passive calibration than with active calibration. The passive calibration values are summarized in Table 5.5a, 5.5b, 5.5c, 5.5d, and 5.5e, where Tan denotes tangential distortion, Rad denotes radial distortion, K is the intrinsic camera matrix, and err denotes the error.

	Parameters	**Values**
Rad	$\begin{bmatrix} k1 & k2 \end{bmatrix}$	$\begin{bmatrix} -0.3527 & 0.1081 \end{bmatrix}$
Tan	$\begin{bmatrix} p1 & p2 \end{bmatrix}$	$\begin{bmatrix} 7.5873 \cdot 10^{-4} & -9.9092 \cdot 10^{-4} \end{bmatrix}$
K	$\begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 403.5068 & 0 & 320.6654 \\ 0 & 403.5167 & 248.2110 \\ 0 & 0 & 1 \end{bmatrix}$
err	e	0.2620

(a) Front camera (F).

	Parameters	**Values**
Rad	$\begin{bmatrix} k1 & k2 \end{bmatrix}$	$\begin{bmatrix} -0.3601 & 0.1206 \end{bmatrix}$
Tan	$\begin{bmatrix} p1 & p2 \end{bmatrix}$	$\begin{bmatrix} 2.1968 \cdot 10^{-4} & 0.0015 \end{bmatrix}$
K	$\begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 404.5495 & 0 & 320.3003 \\ 0 & 404.5017 & 255.8629 \\ 0 & 0 & 1 \end{bmatrix}$
err	e	0.2401

(b) Front left camera (FL).

	Parameters	**Values**
Rad	$\begin{bmatrix} k1 & k2 \end{bmatrix}$	$\begin{bmatrix} -0.3633 & 0.1160 \end{bmatrix}$
Tan	$\begin{bmatrix} p1 & p2 \end{bmatrix}$	$\begin{bmatrix} -6.5629 \cdot 10^{-4} & -8.3030 \cdot 10^{-4} \end{bmatrix}$
K	$\begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 406.4848 & 0 & 315.7331 \\ 0 & 406.3131 & 249.5483 \\ 0 & 0 & 1 \end{bmatrix}$
err	e	0.2081

(c) Front right camera (FR).

	Parameters	**Values**
Rad	$\begin{bmatrix} k1 & k2 \end{bmatrix}$	$\begin{bmatrix} -0.3773 & 0.1527 \end{bmatrix}$
Tan	$\begin{bmatrix} p1 & p2 \end{bmatrix}$	$\begin{bmatrix} 4.4258 \cdot 10^{-4} & 0.0026 \end{bmatrix}$
K	$\begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 404.5471 & 0 & 309.1175 \\ 0 & 404.9005 & 259.6884 \\ 0 & 0 & 1 \end{bmatrix}$
err	e	0.2446

(d) Rear left camera (RL).

	Parameters	**Values**
Rad	$\begin{bmatrix} k1 & k2 \end{bmatrix}$	$\begin{bmatrix} -0.3582 & 0.1159 \end{bmatrix}$
Tan	$\begin{bmatrix} p1 & p2 \end{bmatrix}$	$\begin{bmatrix} 0.0013 & 2.2999 \cdot 10^{-4} \end{bmatrix}$
K	$\begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 403.9308 & 0 & 319.8940 \\ 0 & 404.0541 & 251.5129 \\ 0 & 0 & 1 \end{bmatrix}$
err	e	0.2467

(e) Rear right camera (RR).

Table 5.5: Passive camera calibration based on (Hølland, 2019).

5.4 Setting up ORB-SLAM2 with ROS

ORB-SLAM2 was set up by following the instructions in the README.md file provided in the GitHub repositrory found in (Raul Mur-Artal and Juan D. Tardos and J. M. M. Montiel and Dorian Galvez-Lopez, 2016). Prerequisites such as Pangolin, OpenCV, Eigen and ROS were installed with versions as specified in table 5.2. For some of the packages, newer versions than those specified by ORB-SLAM2 had to be used due to the older versions not being available for the newer Ubuntu 20.04 operating system. The ORB-SLAM2 repository was then cloned and compiled by running the build.sh script. Then the ROS package path were set to be the ROS folder inside ORB_SLAM2/Examples in .bashrc. As the last part of configuration, ORB-SLAM2 was built with ROS by running the build_ros.sh script. The monocular ORB-SLAM2 program could then be started with the following terminal command:

```
rosrun ORB_SLAM2 Mono PATH_TO_VOCABULARY PATH_TO_SETTINGS_FILE
```

The monocular ORB-SLAM program listens for input video on a topic called ”camera/image_raw”. Therefore when the images are published to another topic, like in the provided rosbag files, one would need to do a mapping. An example of this mapping from rostopic /ir/RR to /camera/image_raw is shown below for video files 1 to 3 in Table 5.1:

```
$ rosbag play 2019-11-07-13-51-02.bag /ir/RR:=/camera/image_raw
```

For video files 4 to 10 in Table 5.1, the bag files instead came with thermal images stored on the following format, shown here with an example for the rear right camera: $/infrared/driver/rr/image\_raw$. The mapping therefore changes not only with the chosen camera, but also with the convention used in the bag file. The ORB-SLAM algorithm provides support for usage with ROS. The default method in ORB-SLAM subscribes to a topic where messages of the type $sensor\_msgs :: Image$ are expected to be received, which is where the video files number 4 to 10 publishes their messages as expected. However, in the other rosbag files provided, images are published as the message type $vision\_msgs :: Detection2D$. Therefore the ROS mono C++ file needed to be modified to accept the $visionmsgs :: Detection2D$ message type instead of the $sensormsgs :: Image$. The code is provided in Appendix A so that others wanting to use $vision\_msgs$ instead of $sensor\_msgs$ can find an already available solution.

Additionally, the ORB extractor parameters are set in the yaml settings file. The ORB parameters used in this implementation can be found in Table 5.6. The camera frame rate should also be specified in the settings file. The camera frame rate used in the settings file was 9.0 frames per second (fps), which corresponds to the actual frame rate used in the video recordings.

Parameter	**Value**
Number of ORB features per image	1000
Scale factor between levels in the scale pyramid	1.2
Number of levels in the scale pyramid	8
ORB extractor	Fast threshold
iniThFAST	20
minThFAST	7

Table 5.6: ORB parameters used in our implementation of ORB-SLAM2.

5.5 Cropping of input video

There was a need to suppress the influence of the upper part of the image in the provided thermal video recordings. The input image was therefore cropped such that the upper one third of the image was removed. The cropping was done due to a vast amount of object detections in the skies, and this cropping was performed in the author's specialization project as well. This solution was satisfactory in the way that it moved the ORB-SLAM algorithm's focus away from the skies, and was therefore used in this master's thesis as well. The modifications to implement this was written in the ROS mono file, as well as in the ROS mono inertial file in the case of ORB-SLAM3 with IMU. These modifications are detailed in Appendix A. To account for the cropping and make sure the the principal point stays in the center of the image, the principal point camera parameter in the y-direction was therefore correspondingly set to two thirds of its original value. In this way the principal point stays in the center of the image.

5.6 Training of new vocabularies based on thermal data

5.6.1 Method 1

New vocabularies were created using DBoW2 (Gálvez-López and Tardós, 2012). The default vocabularies bundled with ORB-SLAM2 and 3 are also based on DBoW2, but the data representation is slightly modified. A complete code for creation of a vocabulary on the format used in ORB-SLAM has not been published. In the author's specialization project, the bone stock DBoW2 was method was used to generate new bag of words vocabularies, and the data was then cleaned manually to get a format which could be used by ORB-SLAM. The same procedure as used in the specialization project has been automated, and is from here refered to as ”method 1”. The script used in method 1 is new in this masters' thesis, and can be found in Appendix B.

The procedure for method 1 was based on the demo file in the DBoW2 package. In order to create a vocabulary with the DBoW2 package, the first step will be to copy the images one would like to use for training into the ”images” folder. The images to be used should follow a naming scheme of ”image1.png”, ”image2.png”, ..., ”image[n].png”, etc. Additionally, the demo.cpp file should be modified such that the variable for number of images is set to the wanted amount. Also in the demo.cpp file, it is possible to modify the branching factor k and depth L to get the desired tree structure. The outputted vocabulary from DBoW2 when running *demo.cpp*, is however as mentioned, on a quite different format compared to the built-in ORB-SLAM vocabulary. Therefore it was necessary to convert from the DBoW2 format to the ORB-SLAM format. No explicit information on the conversion was found, so the suggested approach for method 1 in Appendix B was based on assumptions by comparing the DBoW2 vocabulary format and the ORB-SLAM vocabulary format by visual inspection.

The DBoW2 vocabulary was outputted as a yaml file, which contains information about the tree structure, a list of node IDs, their corresponding parent ID and descriptor, and lastly the word ID's for each node ID was given. To convert to the ORB-SLAM vocabulary format, the newly generated vocabulary was cleaned, which included selection of relevant rows, removal of superfluous text, rearrangement of said rows, and export of the file using the correct type of delimiters for row separation. This resulted in the following row format, which had been determined by visual inspection of the bundled ORB-SLAM vocabulary:

parent_id	0/1	descriptor	weight

Table 5.7: ORB-SLAM vocabulary format.

The second column was believed to be zero if all nodes belonging to the same parent had zero weights, otherwise it was set to one. There was also included one additional row in the top of the text file with four space separated numbers corresponding with the number of branches and layers in the tree, as well as scoring type and weighting type, respectively.

5.6.2 Method 2

The second method for generating vocabularies was created later when it was discovered that the files in the thirdparty/DBoW2 folder of ORB-SLAM was modified by the ORB-SLAM authors. The fileset in the ORB-SLAM folder was however not complete, and method 2 therefore involved manual cloning of files to make a complete fileset for generation of vocabularies. The fileset in method 2 was completed by once again cloning the DBoW2 repository, but this time replacing the following files with the corresponding ones found in the ORB-SLAM2 repository:

```
BowVector.cpp
BowVector.h
FClass.h
FORB.cpp
FORB.h
FeatureVector.cpp
FeatureVector.h
```

```
ScoringObject.cpp
ScoringObject.h
TemplatedVocabulary.h
```

Additionally, some modifications were needed in order to be able to compile with these replaced files since ORB-SLAM uses files from the $DUtils$ class from the $DLib$ package (Dorian Galvez-Lopez and Avatar Javier Hidalgo-Carrió and Anup Parikh, 2016). The following files were needed:

```
Random.cpp
Random.h
Timestamp.cpp
Timestamp.h
```

The header files $Random.h$ and $Timestamp.h$ were copied to the $include/DBoW2$ folder, while the source files $Random.cpp$ and $Timestamp.cpp$ were copied to the src folder. Then their paths were added in the $CMakeLists.txt$ file in the same manner as the files already included from the DBoW2 library.

5.6.3 Vocabularies created

An overview of the vocabularies created can be found in Table 5.8, where the built in ORB-SLAM vocabulary also is included for reference. Short descriptions of the different vocabularies will be presented in the following paragraphs:

At first, the vocabulary bundled with ORB-SLAM (which is based on Dorian DBoW2 and optimized for visual spectrum video) was tested on thermal video. This is the largest vocabulary used in this report, and contains approximately 1 000 000 words generated from 10 000 images (Mur-Artal and Tardós, 2014). The images used for training this vocabulary have been specified to be selected from the Bovisa dataset dated 2008-09-01 (which contains various images taken both outdoors and indoors), but which of those specific images that was used has not been specified.

The BoWs trained in this thesis are denoted as ”Custom thermal” with suffixed numbers indicating branching factor and depth. The method used for generation (method 1 or 2) is also indicated. Most of the selected vocabularies were trained on a set of 10 000 images taken by the author in Trondheim. The computing time needed to generate a vocabulary depended heavily on the structure. For example, 9^3 vocabulary took about 2 hours to train, while 10^6 took 2 days.

An additional 10^5 vocabulary was trained on 20 000 images (a mix of images from Stavanger and Trondheim), a training which took 4 days. A final 10^5 vocabulary was trained on 5000 images (also a mix of images from Stavanger and Trondheim).

Name, branching factor, depth	Number of images	Number of words
ORB-SLAM bundled 10 6	10 000	971 814
Custom thermal 9_3 (method 1)	10 000	728
Custom thermal 10_6 (method 1)	10 000	902 273
Custom thermal 9_3 (method 2)	10 000	729
Custom thermal 9_4 (method 2)	10 000	6 561
Custom thermal 9_5 (method 2)	10 000	59 043
Custom thermal 9_6 (method 2)	10 000	511 638
Custom thermal 10_4 (method 2)	10 000	10 000
Custom thermal 10_5 (method 2)	10 000	99 975
Custom thermal 10_6 (method 2)	10 000	899 595
Custom thermal 10_5 (method 2)	20 000	99 974
Custom thermal 10_5 (method 2)	5 000	99 608

Table 5.8: Vocabularies overview.

5.7 Setting up thermal camera for camera data collection

In order to collect camera data from the FLIR Tau 2 camera, the ThermalCapture Grabber software development kit (TCG_SDK_2018_02_14) was downloaded. The thermalgrabber example code was modified in order to save the thermal images. The images were saved using the OpenCV library. Also, the images were saved with a filename on the format "YYYY-MM-DDThh:mm:ss.mmm+01:00" which is based on the ISO 8601 with the only modification being mmm near the end indicating milliseconds. The string variable called "path" was modified in order to save the images in the desired folder. Additionally, the desired time length of the capturing sequence was adjusted. The complete code can be found in Appendix C.

Compilation was done by the running the following command:

```
g++ -std=c++11 main.cpp -o main -Ilibthermalgrabber/inc
-Iusr/include/opencv4/opencv2 -Llibthermalgrabber/lib -lthermalgrabber
`pkg-config opencv4 --cflags --libs`
```

Then the program was run with the following command:

```
LD_PRELOAD=libthermalgrabber/lib/libthermalgrabber.so.1 ./main
```

5.8 ORB-SLAM3 setup

The implementation procedure for ORB-SLAM3 was very similar to the one for ORB-SLAM2. The dependencies were the same, using Pangolin, Eigen and OpenCV. ROS 1 Noetic was also used for ORB-SLAM3. The camera parameters used were the same, and the ORB parameters were almost the same except that the default for ORB-SLAM3 is to extract 1500 features per image instead of 1000. The ORB parameters used are summarized in Table 5.9:

Parameter	Value
Number of ORB features per image	1500
Scale factor between levels in the scale pyramid	1.2
Number of levels in the scale pyramid	8
ORB Extractor	Fast threshold
iniThFAST	20
minThFAST	7

Table 5.9: ORB parameters used with ORB-SLAM3.

Build errors of course depend on configuration, but for this configuration the following build error was encountered:

```
static assertion failed: std::map must have the same value_type
as its allocator.
```

The compilation error was fixed by following the tips from the (non-merged) pull request number 25 in ORB-SLAM3 github repository. The following line:

```
typedef map<KeyFrame*,g2o::Sim3, std::less<KeyFrame*>,
        Eigen::aligned_allocator<std::pair<const KeyFrame*,
            g2o::Sim3> > > KeyFrameAndPose;
```

in the file *LoopClosing.h* was replaced with:

```
typedef map<KeyFrame*,g2o::Sim3, std::less<KeyFrame*>,
        Eigen::aligned_allocator<std::pair<KeyFrame *const,
            g2o::Sim3> > > KeyFrameAndPose;
```

Another important step is to change the ROS_PACKAGE PATH, which can be set in the .bashrc script. The .bashrc script is found in the home directory and runs when a new terminal is opened. The ROS package path was earlier set to the ROS folder inside ORB-SLAM2, and therefore needed to be changed to ORB-SLAM3 when switching to ORB-SLAM3 (and vice versa). The following command can be used in one terminal to set the package path in only this terminal, or alternatively the following line can be added to .bashrc such that it will be set for new terminals from that point on:

```
export ROS_PACKAGE_PATH=\${ROS_PACKAGE_PATH}:/home/ORB_SLAM3
/Examples/ROS
```

Additionally, include statements were updated to make the ORB-SLAM able to find the Eigen files. ”Eigen” was replaced with ”eigen3/Eigen” in the files that were causing compilation error. For example, ”#include ¡Eigen/Core¿” was replaced by ”#include ¡eigen3/Eigen/Core¿” in some files.

The monocular node in ORB-SLAM3 subscribes to the same /camera/image_raw topic as ORB-SLAM2, and is run by the same formatted terminal command as in ORB-SLAM2 stated below: (Note that the settings file is somewhat modified between the two ORB-SLAM versions as described above.)

```
rosrun ORB_SLAM3 Mono PATH_TO_VOCABULARY PATH_TO_SETTINGS_FILE
```

5.8.1 IMU

ORB-SLAM3 provides built-in support for IMU. The ORB-SLAM $Mono\_Inertial$ node subscribes to $/camera/image\_raw$ and $/imu$ for IMU data. In general the ORB-SLAM3 $Mono\_Inertial$ can be run by executing the following command:

```
$ rosrun ORB_SLAM3 Mono_Inertial PATH_TO_VOCABULARY
PATH_TO_SETTINGS_FILE [EQUALIZATION]
```

More specifically three terminals should be started, one for each of the commands listed below. Where $settingfiles$, $vocabularyfiles$, $rostopics$ and $rosbag$ should be replaced for different scenarios.

```
$ roscore

$ rosrun ORB_SLAM3 Mono_Inertial Vocabulary/ORBVoc.txt
Examples/Monocular-Inertial/settingRR_IMU.yaml

$ rosbag play 2019-11-07-13-51-02.bag /xsens/imu:=/imu
/ir/RR:=/camera/image_raw
```

In the $settingsfile$, the transformation between the camera and the IMU is required for the ORB-SLAM3 algorithm to work. One $settingsfile$ for each camera should be used. The different transformations for each camera; front, front right, front left, rear right and rear left, are listed below:

F:

$$T = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.2 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & -2.76 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{5.1}$$

FR:

$$T = \begin{bmatrix} 0.30901699 & -0.95105652 & 0.0 & 0.0618034 \\ 0.95105652 & 0.30901699 & 0.0 & 0.1902113 \\ 0.0 & 0.0 & 1.0 & -2.76 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{5.2}$$

FL:

$$T = \begin{bmatrix} 0.30901699 & 0.95105652 & 0.0 & 0.0618034 \\ -0.95105652 & 0.30901699 & 0.0 & -0.1902113 \\ 0.0 & 0.0 & 1.0 & -2.76 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{5.3}$$

RR:

$$T = \begin{bmatrix} -0.80901699 & -0.58778525 & 0.0 & 0.1618034 \\ 0.58778525 & -0.80901699 & 0.0 & 0.11755705 \\ 0.0 & 0.0 & 1.0 & -2.76 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{5.4}$$

RL:

$$T = \begin{bmatrix} -0.80901699 & 0.58778525 & 0.0 & -0.1618034 \\ -0.58778525 & -0.80901699 & 0.0 & -0.11755705 \\ 0.0 & 0.0 & 1.0 & -2.76 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{5.5}$$

IMU noise parameters

Finding good IMU noise parameters can be challenging, but is important since it impacts how much the IMU measurements are trusted by ORB-SLAM. IMU datasheets usually provide the gyroscope white noise term and the accelerometer white noise term, which are shown in shown in Table 5.10.

White noise terms				
Physical quantity	**ORB-SLAM code parameter**	**GTSAM code parameter**	**SI unit**	**Equivalent unit**
Gyroscope white noise	IMU.NoiseGyro	gyro_noise_sigma	$rad/s^{0.5}$	$rad/s/\sqrt{Hz}$
Accelerometer white noise	IMU.NoiseAcc	accel_noise_sigma	$m/s^{1.5}$	$m/s^2/\sqrt{Hz}$

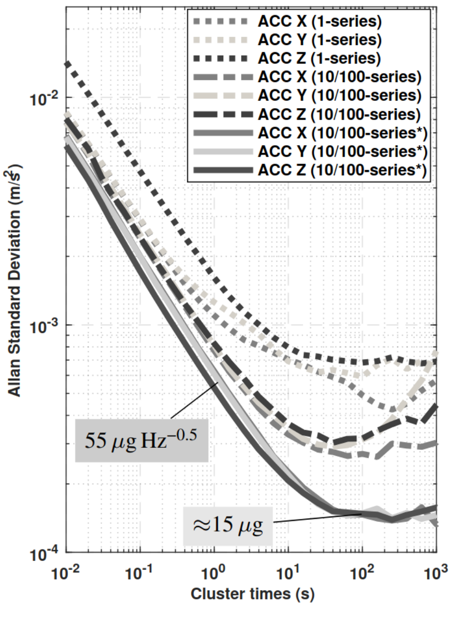
Table 5.10: Common units and notations for IMU noise densities.

Bias terms				
Physical quantity	**ORB-SLAM code parameter**	**GTSAM code parameter**	**SI unit**	**Equivalent unit**
Gyroscope bias random walk	IMU.GyroWalk	gyro_bias_rw_sigma	$rad/s^{1.5}$	$rad\sqrt{Hz}/s$
Accelerometer bias random walk	IMU.AccWalk	accel_bias_rw_sigma	$m/s^{2.5}$	$m\sqrt{Hz}/s^2$

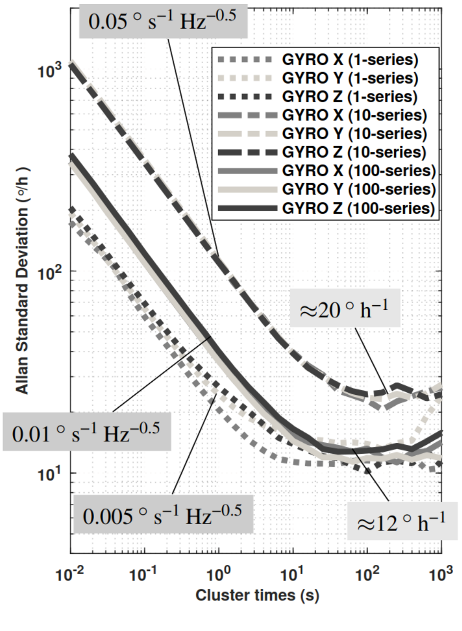
Table 5.11: Common units and notations for IMU bias random walk terms.

However, IMU datasheets usually do not directly provide the random walk terms (which are shown in shown Table 5.11), but instead usually provides their corresponding in-run bias stabilities, and leave it up to the engineer to decide on the random walk terms based on their specific use case. For the gyro, this means that datasheets usually provide the *gyro bias stability* (unit: rad/s) instead of *gyroscope bias random walk* (unit: $rad\sqrt{Hz}/s$) which we need. Likewise for the accelerometer, datasheets usually provide the *accelerometer bias stability* (unit: m/s^2) instead of *accelerometer bias random walk* (unit: $m\sqrt{Hz}/s^2$). One recognized method for calculation of the bias terms is by graphical reading from a plot created using the Allan deviation. In our case for XSens MTi 10-series IMU, the Allan deviation plot for the accelerometer and gyro can be seen in the left and right part of Figure 5.1, respectively.

The angular randow walk line is sketched into both the gyro and accelerometer diagrams see Figure 5.2 and 5.3 respectively. The lines are drawn with a slope of -1/2 in the log-log diagrams and values should be read when T=1. The line for rate randow walk would be drawn in a similar manner in the log-log diagram, but with a slope +1/2 instead and be read when T=3. In the case of the gyro, it is important to be aware the units on the axes as the y-axis has unit degrees per hour while the x-axis is given in seconds. For more details on how to do conversion from the AD values to bias terms, interpretation and Allan deviation in general see (IEEE, 1998) and (El-Sheimy et al., 2008).



(a) Allan variance curves of accelerometers



(b) Allan variance curves of gyroscopes

Figure 5.1: Image from (Vydhyanathan and Bellusci, 2018): Allan deviation plot from an XSens MTi 10-series whitepaper. Our sensor is a 4th gen MTi 10, indicated by the medium dashed lines. (The solid lines indicated by the star (*) shows values for the later 5th gen MTi 10-series).

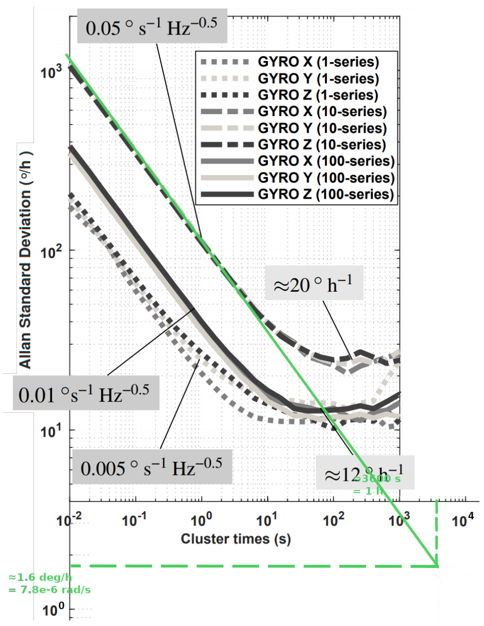


Figure 5.2: Manually traced Allan plot for gyro.

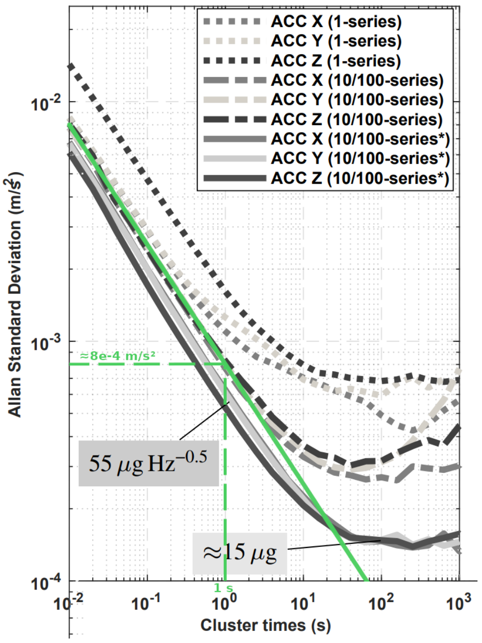


Figure 5.3: Manually traced Allan plot for accelerometer.

In this thesis, it was decided to use the parameters below which were provided and are based on previous work with a roughly similar XSens MTi used with GTSAM:

```
IMU.NoiseGyro:  1.7500e-04  # rad/s^0.5
IMU.NoiseAcc:   7.8480e-04  # m/s^1.5
IMU.GyroWalk:   0.0035      # rad/s^1.5
IMU.AccWalk:    0.03        # m/s^2.5
IMU.Frequency:  100         # Hz
```

As can be seen in Table 5.10 and 5.11, the units of the parameters used in GTSAM have been checked, and have the same physical quantities as the parameters used in ORB-SLAM.

In general, more accurate parameters for the specific IMU setup used will give better performance: A noisy IMU should be described with noisy parameters. In theory, better and more precisely synced IMU hardware allows us to use smaller IMU noise parameters which will give better performance. The time synchronisation between camera frames and IMU measurements is important here, and is handled by the rosbag files in this project. In reality there are also other noise terms, and for example temperature changes are not accounted for. Some would call this an optimistic model, and to capture other noise parameters, the Kalibr GitHub wiki suggests increasing the noise parameters (user rehderj, 2016). Kalibr therefore suggests increasing the noise parameters with a factor up to 10 when using very low cost sensors. What is considered as a ”cheap” or ”expensive” sensor might vary, and what can be regarded as a low performance or high performance sensor depends on several factors. XSens market their MTi 1-series as low cost sensors, the MTi 10-series as industrial grade sensors, and the MTi 100-series as higher grade. Despite its industrial grade rating, the provided parameters listed above were modified by multiplying them with factors of 10 and 100.

Another important IMU parameter is the frequency. To obtain the frequency of the measurements, the command *$ rostopic hz /imu* was used, which returned how frequently IMU messages were published. The ”IMU.Frequency” parameter in ORB-SLAM was therefore set to 100 Hz.

5.9 Plotting of SLAM and GNSS trajectories

The evo Python package (Grupp, 2017) was used in order to plot the ORB-SLAM and GNSS trajectories. ORB-SLAM outputs the keyframe trajectory on the TUM format, which is accepted by this package. For ground truth and reference, the GPS trajectory published by the $/TF$ rostopic was used. The message published is the transformation between the GPS antenna on milliampere and a fixed location at the pier. The information on the TF rostopic were exported to the TUM format. The following terminal command can be used for exporting the GPS reference for video file 1:

```
$ evo_traj bag 2019-11-07-13-51-02.bag /tf:piren.gps_antenna
```

```
−−save_as_tum
```

The TUM format is on the following form:

timestamp $x\ y\ z\ q_x\ q_y\ q_z\ q_w$

where x, y and z is the position and q_x, q_y, q_z and q_w are quaternions representing rotation.

In order to align and scale trajectories by the first n poses, the argument used is $--n\_to\_align\,N$. To align with respect to all poses the argument $-a$ can be used, while the argument $-s$ can be used for scaling. The evo package uses Umeyama alignment to achieve this. Poses in the SLAM trajectories with more than $0.01s$ in difference to any GPS pose will not be used, the threshold can be modified, but in this implementation it was set to the default value.

The evo package was installed using python3 and pip3. With this method, "/home/hal/.local/bin" needed to be exported to path in the terminal where the evo package wanted to be used in order to locate the package. The following command can be used to accomplish this:

```
$ export PATH="/home/hal/.local/bin"
```

where x, y and z are the position and q_x, q_y, q_z and q_w are quaternions representing rotation. In order to align trajectories by the first 3 poses, the argument used is –n_to_align. To align with respect to all poses the argument -a can be used, while the argument -s can be used for scaling. The evo package uses Umeyama alignment to achieve this. Poses in the SLAM trajectories with more than $0.01s$ in difference to any GPS pose will not be used, the threshold can be modified, but in this implementation it was set to the default value. Note that in this implementation only keyframes were used to plot the SLAM trajectories. In order to obtain more poses all frames could have been used, but then these frames should be computed relative to a keyframe, since only keyframes are optimized as part of bundle adjustment.

The number of poses used for alignment can have a huge impact. For example, if only using the 3 first poses, and those are of poor quality, you can risk getting a flipped map (mirror image), which was experienced. Therefore, at least 3 to 10 poses were used for most of the plots. Alignment using even more poses could also be used, but may mask the drift of the SLAM method. This is particularly of interest with un-aided monocular SLAM.

Chapter 6

Results and discussion

6.1 Vocabulary trained on Bovisa dataset

First, a new vocabulary was trained on the Bovisa dataset. This was done in order to verify that vocabularies were trained in the correct manner by attempting to replicate the built-in vocabulary to ORB-SLAM. This study was also motivated by results in the author's specialization project where several self-trained vocabularies seemed to even struggle to initialize at all and at that time the Bovisa dataset was not available for download. The Bovisa dataset was possible to download during the time this thesis was written and therefore this verification of training step was now possible to do. The results were expected to be similar, but not identical, as there are some uncertainties in some steps of the training as well as some randomness in the ORB-SLAM method from one run to the next.

For training, two methods were tried. The second method suggested in section 5.6 was likely the most similar to the ORB-SLAM approach and therefore used for this comparison. Both vocabularies used the same tree structure, branching factor 10 and tree depth 6, and was trained on the same number of images. For comparison the rear left camera for video file 1 is used. The built-in vocabulary was able to initialize after 30 seconds, while the one self-trained on the Bovisa dataset initialized one minute later. Then both keep track for the rest of the video which in total lasts 13 minutes and 40 seconds. Therefore in these cases there was no opportunity for relocalization, neither loop closure for that matter as there was no loops in the path. Initial thoughts on whether the training method might be correct are positives as in the author's specialization project several of the vocabularies struggled to initialize at all.

Another example was also used to verify the second training method. Also, this time the first video file was used, but with the rear right camera. ORB-SLAM with the self-trained vocabulary based on the Bovisa dataset lost track before the tunnel (5:15 min), while the built in ORB-SLAM vocabulary has been run four times for this video, two times it lost track in the tunnel as well, one time in a large and quick turn (3:30 min) and one time it didn't lose track at all. ORB-SLAM with built-in vocabulary initialized quicker also this time, after 15 seconds compared to 35 seconds for the self-trained. This example also gave confirmation that the training method might be correct as the performance was similar. The fact that performance can vary from run to run was taken into account when looking at the results, both in this section, but also for the rest of this chapter. The algorithm is non-deterministic, but there were outcomes that were more likely than others. For example often initialization happened in one out of a few locations were sideways translation was good, while track was typically lost under quick rotations or in the tunnel, like mentioned in this example.

The first method presented in 5.6 was also explored. A vocabulary trained by using this method and a tree structure with branching factor 10 and depth 6, did not initialize at all in video file 1 with rear left camera. Both the vocabulary trained with the second method and the built-in ORB vocabulary were able to initialized relatively early as mentioned above. This vocabulary therefore did not appear very promising as even though the environment was challenging for SLAM, the video was quite long and there were situations with good sideways translation motion. However, when training in the same manner a much smaller vocabulary with tree structure $k = 9$ and $L = 3$ the performance was a lot better. The vocabulary initialized at 1 minute for the same video and camera angle as mentioned above and kept track the whole video except when the agent rotated quickly and it lost track briefly before re-localizing. This motivates the later study, found in section 6.4, of the affect of different vocabulary size.

By comparing how the ORB SLAM vocabulary was assumed created at first (method 1) with how it was more likely created (method 2), it was seen that the vocabulary files were created in a similar manner. The most uncertainty when creating method one, was a binary column which was believed to be ”1” if the node itself or any of the siblings had non-zero weights, while ORB-SLAM used ”1” if leaf node and ”0” if non-leaf node. These statements are believed to be almost equivalent as only words have weights. However, if all the leaf nodes to one parent had zero weights the first method would provide the wrong result. Due to this and other modifications done to DBoW2 by ORB-SLAM, the focus will forward on by on method 2.

Through this initial investigation the confirmation sought after was obtained for the second method. The differences in performance could be explained by random elements in the ORB-SLAM algorithm as well as it is unknown which 10 000 images ORB-SLAM used from the Bovisa 2008-09-01 dataset, as it in this case were used 10 000 random ones from the 16 000 in the front camera torrent. Also, investigating the effect of different vocabulary sizes was motivated and will be presented later in this report.

6.2 Camera parameters

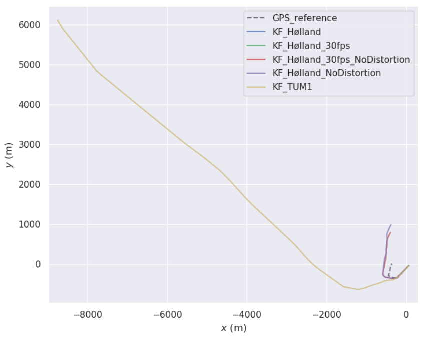
By coincidence the ORB-SLAM2 algorithm was tested with a settings file corresponding to TUM1 datasets and the keyframetrajectory appeared in the map view at first glance to be more reasonable than the ones calibrated in particular for the camera used, found in

(Hølland, 2019). This was of course a surprising result, but interesting and something that needed to investigate further as also in the author's specialization project there was some discrepancies in the creation of plot, and camera parameters could be the reason for it. Assuming that the calibration was correct, focal length and principal point were kept the same. Focal length would probably be most influencing as the principal point would give more of a bias error. However, the radial and tangential distortion parameters could be possibly overestimated, so an attempt was made to run ORB-SLAM with distortion parameters set to zero. Another difference between the setting files was the frame per seconds parameter, in the TUM case it was 30 fps, while in the Milliampere case it was 9 fps. Therefore the fps parameter was also adjusted to 30 fps in order to investigate its effect.

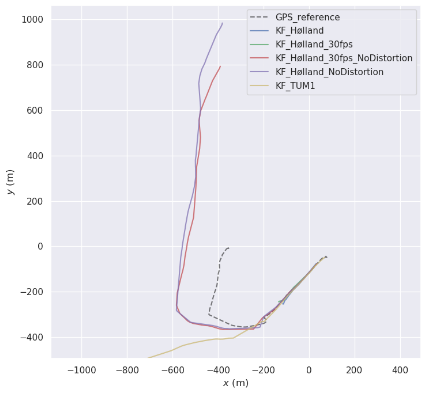
In order to see the effect of varying distortion parameters and fps, keyframe trajectories were plotted together with GPS reference. The trajectories were aligned and scaled with respect to the GPS reference. The alignment and scaling were done with respect to the first three poses. The plots looked similar also when aligning and scaling with respect to the for comparison 10 poses as well. In Figure 6.1 the keyframe trajectory for calibrated camera parameters found by Hølland and 9 fps were plotted in blue, in green the same parameters were used only fps was changed to 30, in red the distortion parameters were additionally set to zero, in purple only distortion parameters were set to zero and in yellow the settings file for the TUM1 dataset was used. The GPS reference is plotted in dashed grey.

In Figure 6.1 the TUM1 stood out due to its large drifting scale. It made sense that the TUM1 drifted more in scale as the camera parameters were in fact wrong. However, from the plot it was understandable that this TUM1 appeared at first look to make more sense than the one with correct parameters (KF_Hølland) in blue. The KF_Hølland trajectory kept track just as long, but seemed to drift in scale as well. This can be seen in Figure 6.2 as the number of keyframes is almost tripled from the first frame, but the map looked very similar. Also the drift can be seen since map points were visible (colored red) before the tunnel, which was correct, but after the tunnel, then visible map points (in red) were almost not visible at the shown scale. One can see some red close to the current keyframe (green) in the last frame, this could indicate that maybe very many map points have relative scale much smaller, and therefore at this scale lie upon each other, just as the keyframes lie upon each other and the trajectory in figure 6.1 blue appears to stop much sooner than the others. Both the blue and green trajectory was a bit difficult to see in the plot, but is visible in the zoomed in version of the xy-plot.

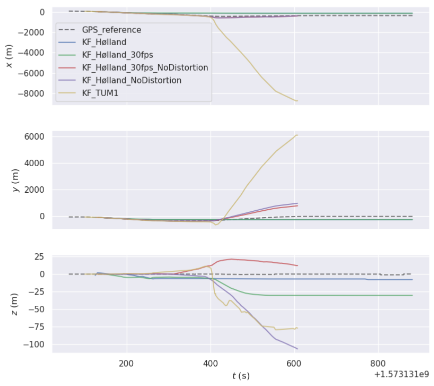
Other observations based on Figure 6.1 were that the ones with no distortion seemed to appear very similar shaped as the GPS reference, even though they are drifting in scale as well, but in the opposite manner, the scale has increased. Adjusting between 9 fps and 30 fps seemed to effect the trajectories to some extent, however, one does not seem to be clearly better than the other based on these plots. Looking at 6.1c there seems to be some event happening around 400 seconds that make the position drift. The frame taken approximately at this time, around 400 seconds, is shown in Figure 6.3c. There were some non-ideal dynamic features detected on the sea surface, however, there was more difficult scenes with less stable features before this frame. A closer look at Figure 6.1c indicates



(a) xy-plot.



(b) xy-plot zoomed.

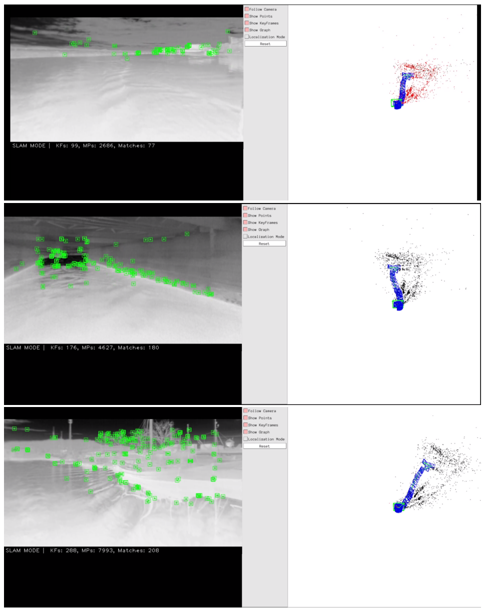


(c) x(t), y(t) and z(t).

Figure 6.1: Comparison of GPS reference and SLAM trajectories when the first three poses were aligned and scaled.

that the z-estimates for some of the trajectories start to drift before this time. By looking at what happens around this time, there are more features on the water around 360 seconds, frame shown in Figure 6.3b and before that around 330 seconds the challenging tracking through the tunnel with little distinct features took place. These scenarios could possibly have lead to the scale starting to drifting.

Another example from video file 3 using the rear left camera is shown in Figure 6.4. The trajectories with no distortion parameter keep track the longest and shows a clearly resembling shape. The trajectory with TUM parameters keep track only a short period, and the trajectories with distortion parameters keep track for a shorter period, but longer than the run with TUM parameters. By looking at the time series plot for positions, it was seen some resemblance. It is also worth noting that the scales on the axis were varying so for the z-position the prediction was less than 4 meters off for the no distortion and 30 fps one



```
SLAM MODE |  KFs: 99, MPs: 2686, Matches: 77
SLAM MODE |  KFs: 176, MPs: 4627, Matches: 180
SLAM MODE |  KFs: 288, MPs: 7993, Matches: 208
```

Figure 6.2: Scale drift map appears to still.

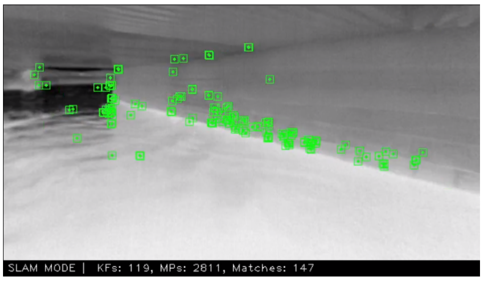
(red) before it improved. The no distortion one (purple) was also less than 4 meters off in z-position, but seemed to be drifting when track was lost.

Based on looking at these two samples it was concluded to forward on set the distortion parameters to zero, the framerate will be kept as 9 fps. These samples were chosen as they have different trajectories and tracking were good in both cases relative to other cases, making more keyframes available for comparison. The change of setting camera parameters to zero will make the algorithm lose more information in the edges as the data here will not fit the model and therefore be thrown away and not used. However, the approximation is good, as it would be worse to operate with wrong distortion parameters like in the examples above. Overtraining/overfitting can often be the case when using too many parameters. In our case three parameters for radial distortion and two for tangential distortion appeared to be too many.

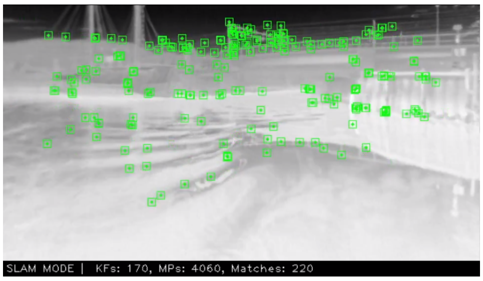
6.3 Camera data collection

The thermal images captured as part of this thesis consists of two datasets, with one dataset taken in Trondheim and in Stavanger. Each dataset had a size of 10 000 images.

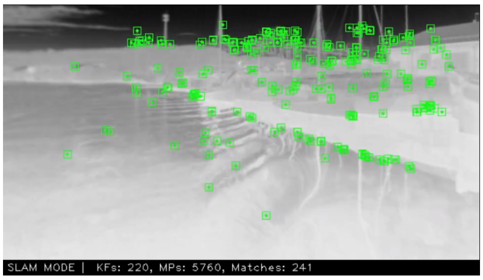
The dataset captured in Trondheim used images both from indoors and outdoors around the Gløshaugen campus of NTNU, as well as outdoors in the area around Brattøra quay taken in a radius of approximately 1-2 kilometers. The images were taken one evening in November without rainfall, and mostly in movement while walking. The dataset captured



(a) Frame that occurred at 330 seconds.



(b) Frame that occurred at 360 seconds.

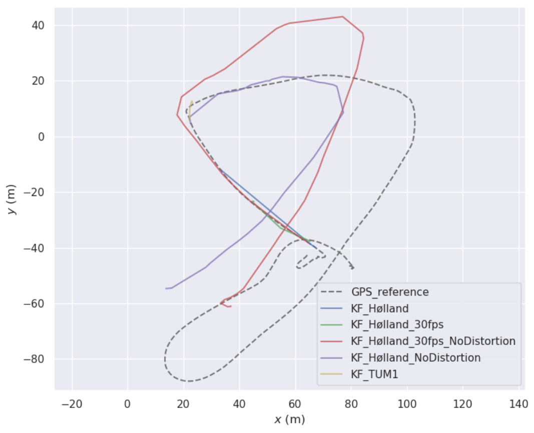


(c) Frame that occurred at 400 seconds.

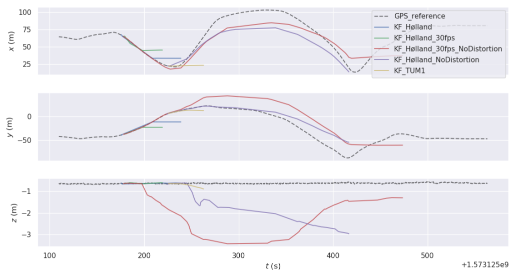
Figure 6.3: Frames where drift occurs.

in Stavanger was mostly taken while driving, and therefore the images therefore possibly contain more motion blur due to higher speeds. The car window had to be rolled down since the thermal camera could not see through the glass. The Stavanger dataset was also created in November, there were some light drizzle precipitation outside when taking the images. The images were mostly taken around the Stavanger city center. Both data sets were captured as 8-bit images.

Some example images from these datasets are shown in Figure 6.5. The images appeared to contain some noise. It was speculated whether this could be ".jpg" noise, but editing the code and saving the images as the filetype ".png" instead of ".jpg" by using OpenCVs $imwrite$ function gave the same appearance.



(a) xy-plot.



(b) x(t), y(t) and z(t).

Figure 6.4: Comparison of GPS reference and SLAM trajectories when the first three poses were aligned and scaled.

Figure 6.6 shows two images taken seconds apart. The whole image exposure changed, likely due to an automatic normalization in the FLIR Tau2 camera. In our case this did not present at practical problem, as the images were used to train vocabularies and it was actually desired to have different thermal signatures, but this could be a nuisance in some other scenarios. Furthermore the camera focus was set automatically through our image capture program. A clicking or gearing sound sometimes came from the camera (especially when the lens cap was on), and this was probably the autofocus.

In Figure 6.7 it can be seen that the thermal camera was not able to see through the car window, and that reflections (a mirrored image) can be seen when taking a thermal image of the laptop screen.

6.3.1 Thermal camera performance in snowy weather

It is common knowledge that thermal camera have good performance in dust, smoke and fog, but it was interesting to see its performance on snow.

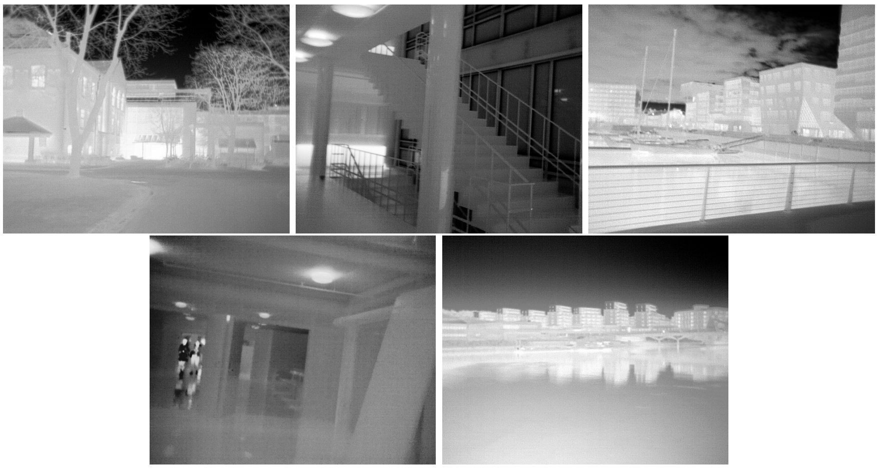


Figure 6.5: Example images self-created dataset, from upper left: Gløhaugen campus (outdoor and indoor), Brattøra, inddor Gløshaugen again, and Stavanger.

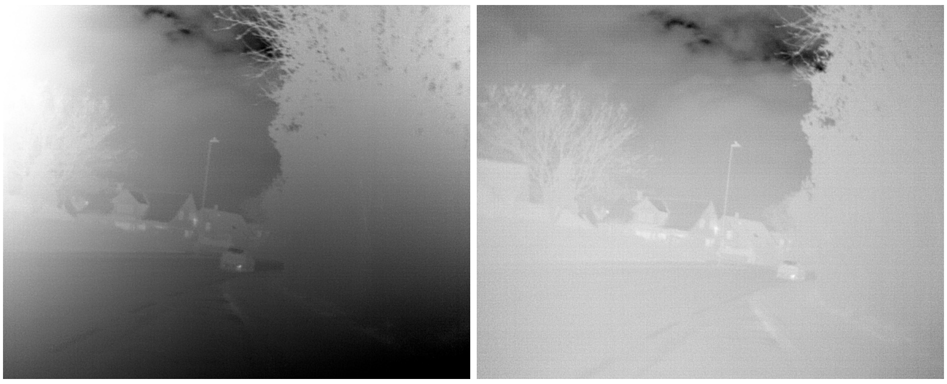


Figure 6.6: Large exposure changes: Two images of the same scene, showing one instance where the FLIR camera struggled to provide good contrast during auto adjusting.

To compare this, one experiment was conducted where images were taken of a house on a clear night when there was no snow on the ground, as well as from under an umbrella during snowfall and finally after the snowfall had ended. Qualitatively it can be seen from Figure 6.8 that the images that visibility was somewhat decreased during the snowfall. The image also lacked some of the previous texture on the ground after the snowfall had laid its temperature-wise monotone cover. Other details in the image which were still not covered by snow were also affected, probably due to the in-built normalization of the camera, such that less details were visible after the end of the snowfall. The previous images taken during the clear night without any snow showed better contrast and more details.

Another experiment consisted of taking images of a cold and stationary car in cold, sta-

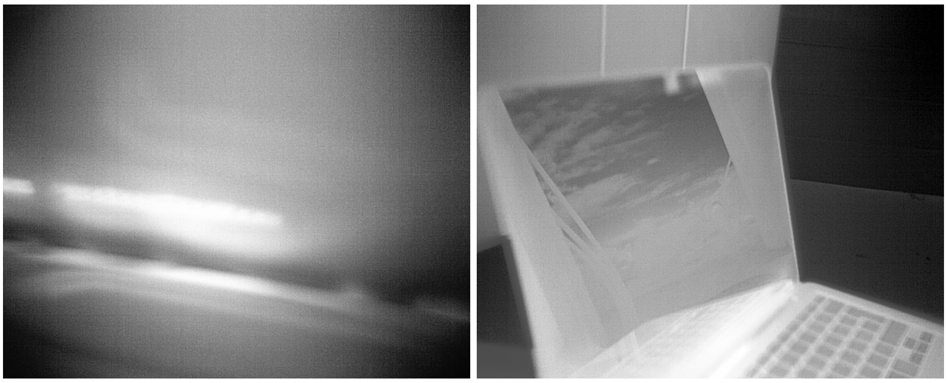


Figure 6.7: Left: Thermal camera not able to "see" through car window. Right: Thermal camera captures mirror reflections from computer screen.

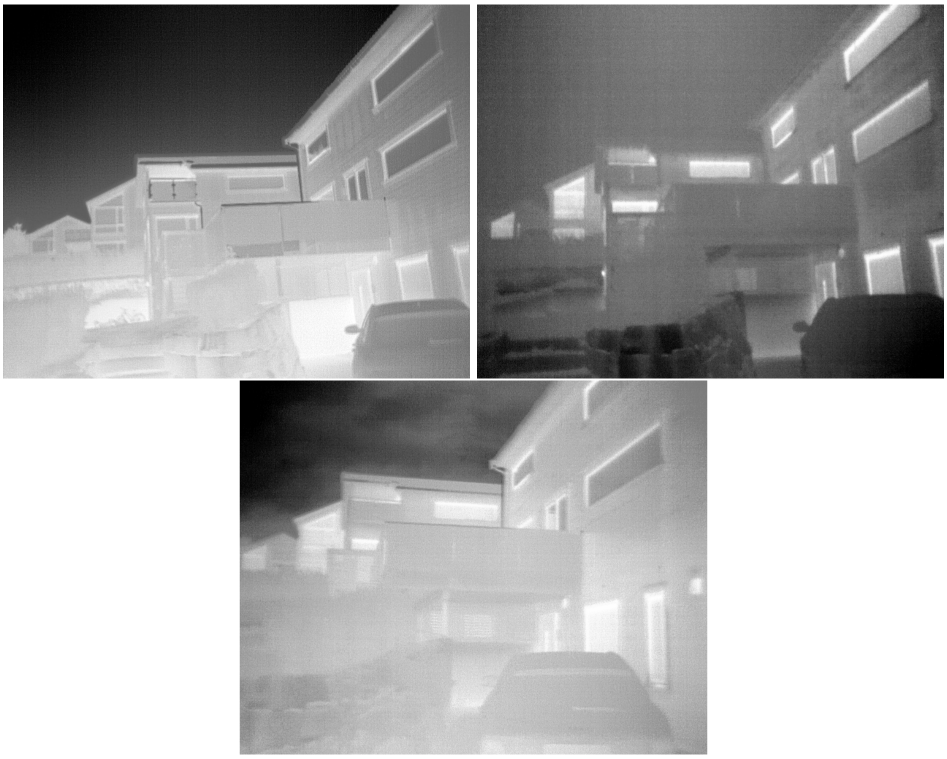


Figure 6.8: From upper left: Before snow, during snowfall and after the snowfall has ended. The thermal camera provides decreased image quality during the snowfall. It improves somewhat after the snow has fallen, but the scene still looks "flatter" with less texture after it has stopped snowing.

ble weather conditions during a cold night. As can be seen in Figure 6.9, the car was highly visible in the thermal images before the snowfall, but almost invisible after an approximately 2 cm thick snow layer had fallen on the car. The lines disappeared, and it was almost impossible to see the car on the thermal camera, even on close-up images. The image in general also has a matte appearance, which may indicate that the automatic normalization struggles to highlight contrasts.

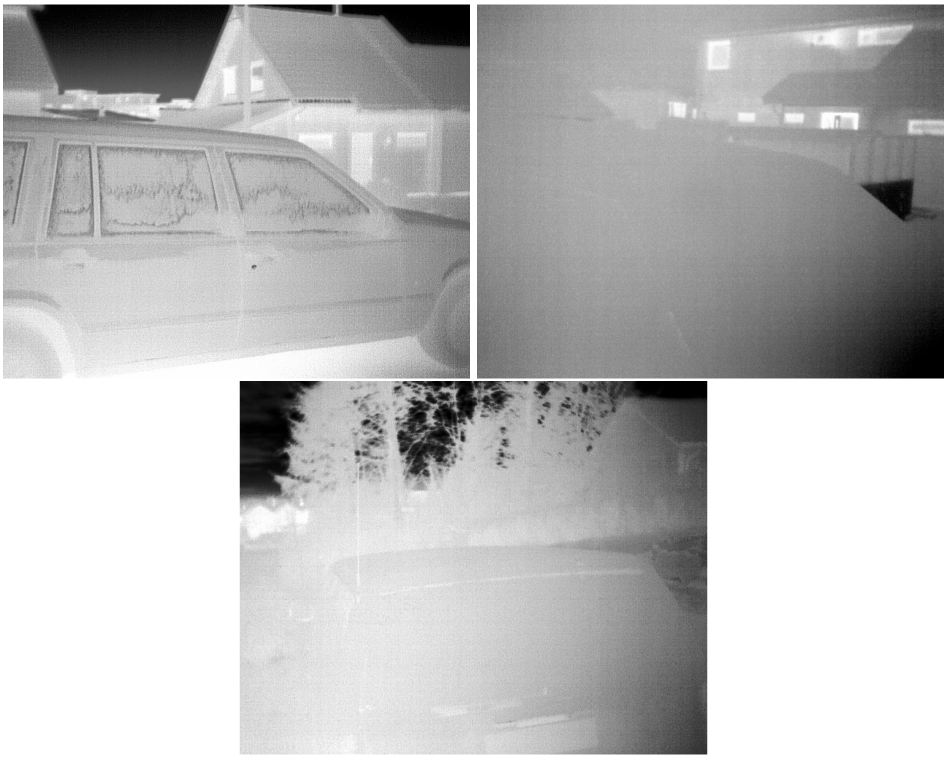


Figure 6.9: A cold car, with approximately the same temperature as its surroundings. The upper left image is before snowfall, while the following two images are after snowfall has laid an approximately 2 cm coat with very even temperature, making the car almost invisible in the thermal images.

6.3.2 Thermal camera performance when blinded by lights

One area where thermal cameras excel is in dynamic lighting conditions, such as when being blinded by lights. The visual spectrum images below are taken with an iPhone 6s.

Figure 6.10 indicates that it for example might be easier to count ongoing cars and gain other relevant forms of situational awareness, a situation where visual spectrum cameras or even human vision will be disturbed the oncoming headlights. Figure 6.11 is another example, showing that how the sun can potentially blind drivers or visual spectrum camera, while the thermal camera provides stable performance during day and night. A thermal camera therefore might be considered a good complement to visual spectum cameras in some situations.

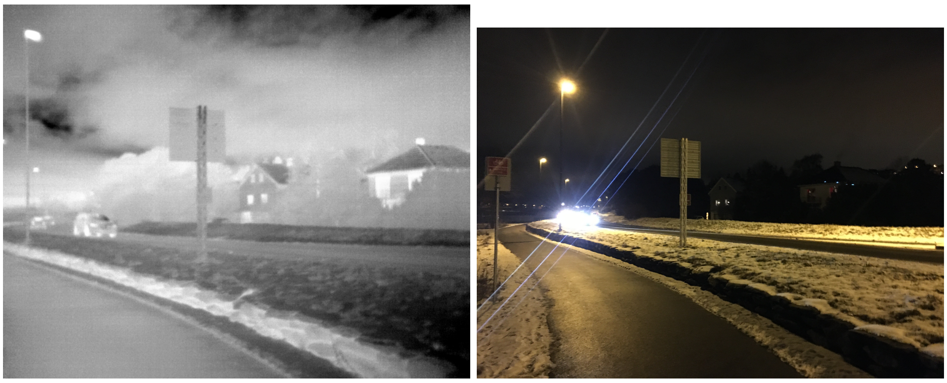


Figure 6.10: How many cars? The thermal camera is not blinded by the car lights, making it easier to spot and count the cars.

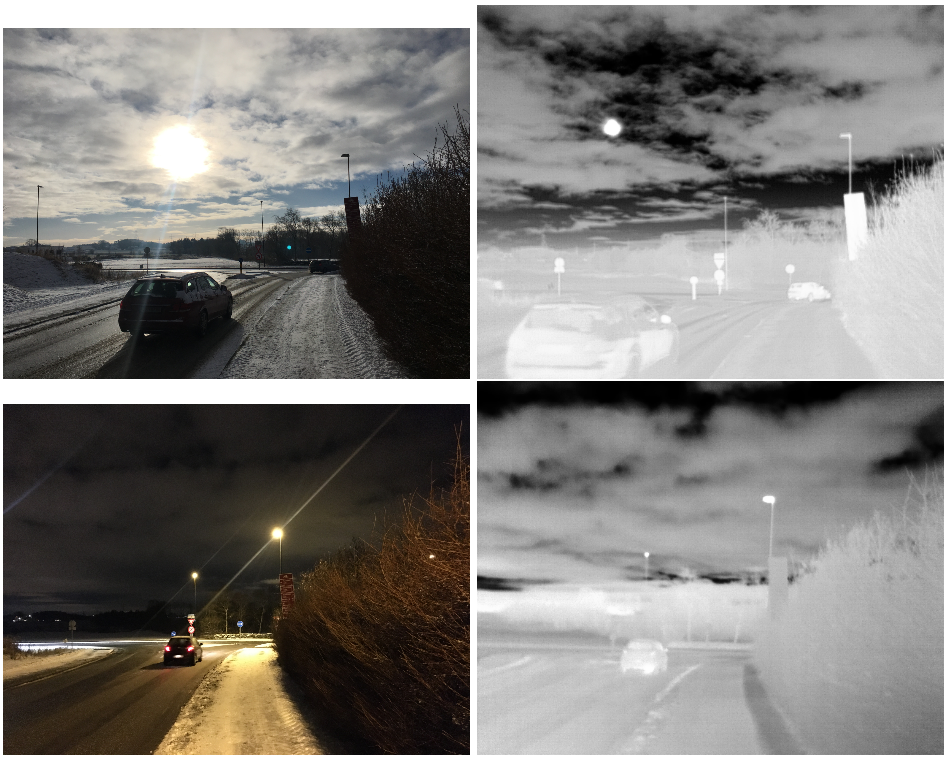


Figure 6.11: Well lit intersection. The thermal camera provides very similar images both day and night, and is not blinded by the sun.

6.3.3 Thermal camera motion blur

The thermal camera has some motion blur, but is very stable regardless of the amount of lighting. In comparison, visual spectrum cameras suffer much more from motion blur in low light situations due to longer exposure times. Image 6.12 show an example of superior motion blur performance of our thermal camera during night, even though the street is well lit. The visual spectrum images are taken with an iPhone 6s.

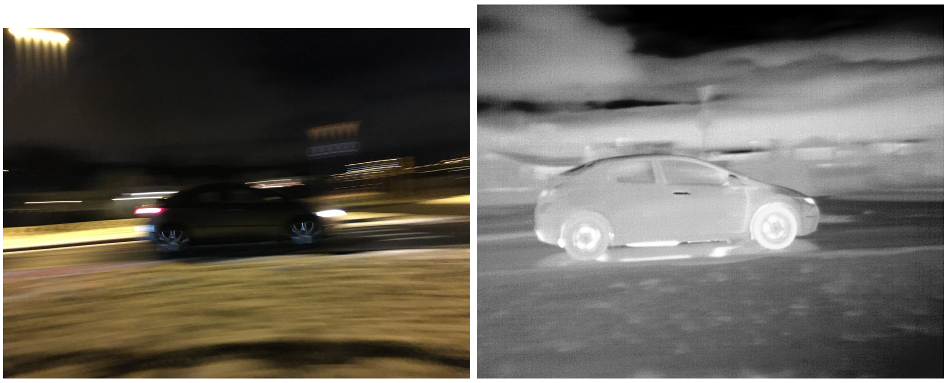


Figure 6.12: Motion blur: The visual camera suffers from severe motion blur during night, while the thermal camera provides similar performance as during daytime.

6.4 BoW trained on self-collected dataset

The goal of this section is to test different tree structures for vocabularies, and the effect of using different images as well as different amounts of images. The comparison is done by looking at keyframe trajectories and considering challenging place recognition scenarios.

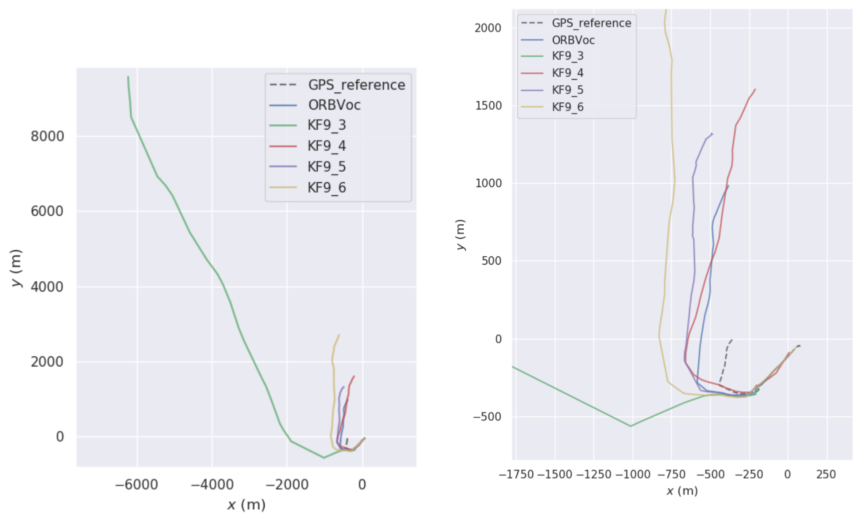
6.4.1 Comparison tree structure

In this section the branching factor k and tree depth L is varied in order to analyze their affect on vocabulary performance. The tree structures (k_L) tested were 9_3, 9_4, 9_5, 9_6, 10_4, 10_5 and 10_6. The built-in ORB-SLAM vocabulary was also included, to see how well the custom trained vocabularies compared. The built-in ORB-SLAM vocabulary has, as mentioned before, tree structure 10_6.

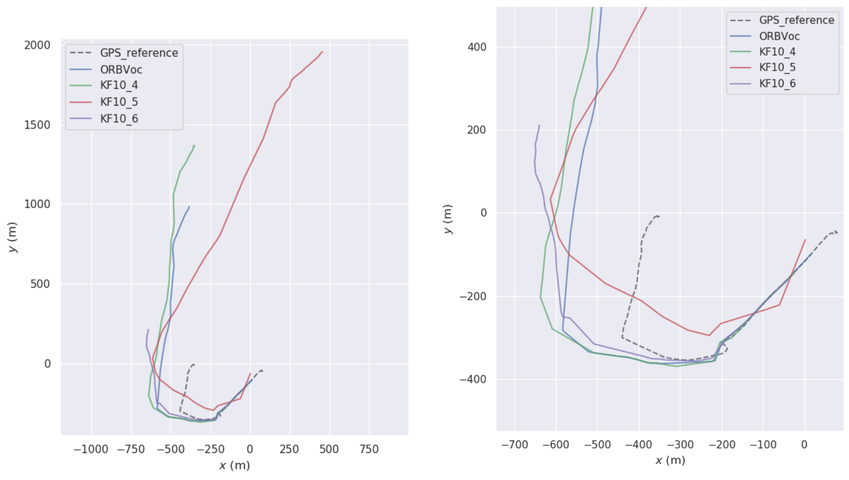
For comparing tree structures, first an example with the rear left camera on video file 1 is shown. The keyframe trajectories were scaled and aligned with respect to the first five poses. The resulting trajectories are shown in Figure 6.13. The trajectory created with the built-in ORB-SLAM vocabulary as well as the GPS reference are also included. The trajectories were divided in two groups in order to make the plots less crowded. In Figure 6.13a the trajectories with branching factor set to 9 was included, while the ones with branching factor 10 was included in Figure 6.13b, zoomed versions of the xy-plot was included as well. By inspecting the resulting trajectories it was seen that all have a similar shape as the GPS reference. This result was not obvious as previous experience showed that poorly trained vocabularies can have trouble initializing at all or quickly lose track. It can further be seen that some trajectories drifted in scale quicker than others for example 9_3, 9_6 and 10_5 drifted the most in scale. The custom thermal vocabulary with size 10_6 had the most resembling trajectory in terms of scale. However, there were no clear patterns that for example the smaller vocabularies performed worse than the larger ones or vice versa.

The same video file, but this time the front right camera was used as another example. Also this time the first 5 poses were aligned and scaled with respect to the GPS reference. The keyframe trajectories was also included for this example, see Figure 6.14. Also this time performance was similar for the different tree structures, the shapes were resembling one another. By inspection of the run it was seen that ORB-SLAM lost track at approximately the same time for all the vocabularies, around the point (-200,-300) in the xy-plot. However, it can be seen that the trajectories did not reach close to this point in the xy-plot. During the run it appeared like the agent had no translation, only rotation, after some time. This was believed due to the repeating pattern of the pier. It is possible that the algorithm believed it was seeing the same scene and therefore rotated backwards, which was the camera motion observed. In Figure 6.15a it is possible to see in the map view that the algorithm believed that the camera looked back on the trajectory it had moved. The map view was set to follow the camera, and it therefore showed what the algorithm believed that the camera saw. However, this view would better correspond to the back camera, not the front right as there was no rotation at this point. It is also possible to see in the map view that the keyframes are stacking upon each other and as mentioned there are more rotational movement at this point than translation movement which would better fit the reality.

Another observation from the run mentioned above is that the track was lost in the frames after these repeating pier frames, even though they were more distinct and had more distinct objects like the tower on the pier and the small house looking building shown in Figure 6.15b. The fact that track was lost in these better tracking conditions was believed due to that keyframes and map points prior were created erroneously and therefore made it harder to create new ones fitting with the current algorithm ”hypothesis” or ”world-view”. It was interesting that these events were similar for all vocabularies and therefore none stood out as the better choice. It is worth noting that the ”driving along the pier” scenario is extremely difficult for the SLAM in general as the frames are very similar in this part of the harbour. The 10_6 tree structure with the most resembling shape in the previous example, had the least resembling shape this time. The behaviour using this vocabulary is strange like for the others, with a lot of rotation and keyframes stacked upon each other, but it happens somewhat sooner when using this vocabulary compared to the others. However, it did not rule out the vocabulary as more runs could have made the vocabulary keep track a bit longer than the other. As mentioned before, the ORB-SLAM is non-deterministic and running several times could give very different results. The results for this example



(a) xy-plot, original and zoomed, for vocabularies with $k = 9$.

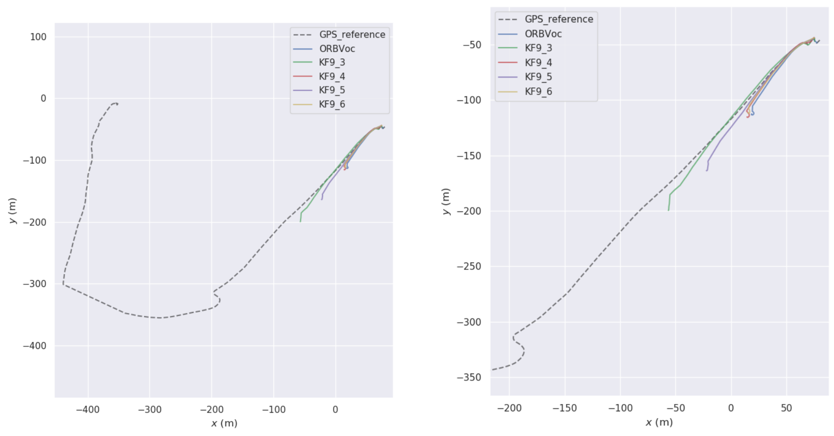


(b) xy-plot, original and zoomed, for vocabularies with $k = 10$.

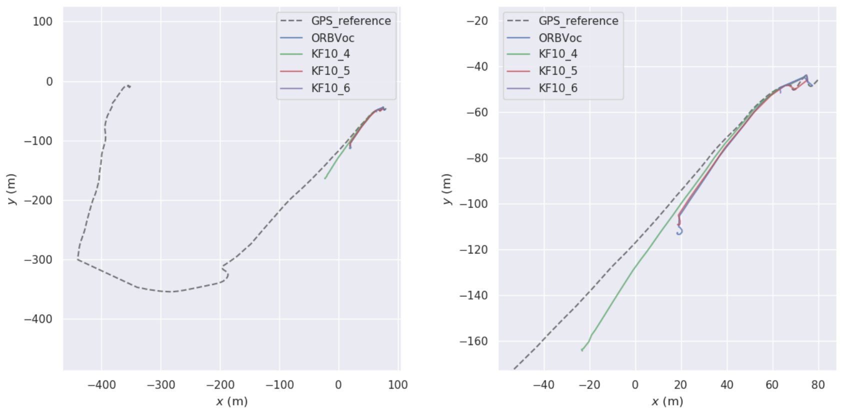
Figure 6.13: Trajectories created where vocabularies with different tree structures were used.

gave the same experience as the previous example, in that the vocabularies provide similar shaped trajectories and similar tracking.

In Figure 6.16, it is possible to see that the trajectories for different tree structures all have the same shape this time as well. This time the rear right in video file 3 was used. The first five poses were used to align and scale in Figure 6.16a, while in Figure 6.16b the first ten poses were used, as using only five resulted in trajectories associated with vocabular-



(a) xy-plot, original and zoomed, for vocabularies with $k = 9$.

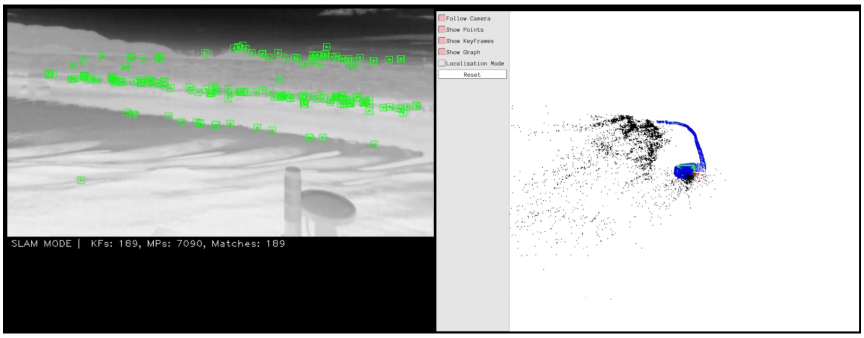


(b) xy-plot, original and zoomed, for vocabularies with $k = 10$.

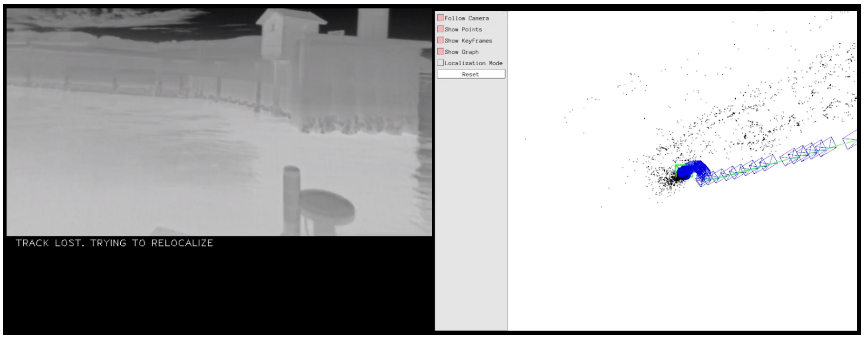
Figure 6.14: Trajectories created where vocabularies with different tree structures were used.

ies 10_4 and 10_5 were mirrored. It can be seen by looking at the GPS reference that the ferry drives in a loop, this example was therefore included to check if loop closure was detected. However, as one can see from the GPS trajectory the viewpoint was different when arriving at the same place, making loop closure based on camera difficult. The custom thermal 10_6 vocabulary kept track for the longest period, the closest frames in terms of loop closure was the frames shown in figure 6.17, but as one can see it was not a good loop closure candidate, even though some of the same objects were present, as the viewpoints and locations were too different giving the frames very different appearances.

In Figure 6.18, the GPS trajectory for video file 4 is shown. There can be seen a loop in the end of the trajectory. However, also this time the viewpoints were different at the



(a) Perceived rotation.



(b) Frame where track was lost.

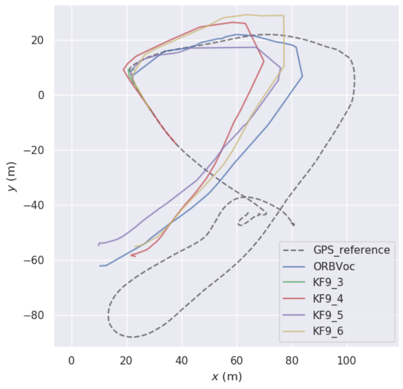
Figure 6.15: Perceived rotation and frame where track was lost.

start of the loop compared to the end of the loop, making loop closure difficult. In additional the loop was mostly rotation, resulting in track being lost before being able to finish the loop at all in the tested cases. It is also worth to mention that ORB-SLAM with the built-in vocabulary initialized in this case quicker than the custom thermal ones, using this vocabulary the initialization time was 18 seconds. Most of the ones using custom thermal vocabularies initialized at the exact same time $2:40$ seconds. Using the 9_3 vocabulary initialization was $1:30$, while for the one using 10_4 initializes in the last seconds of the 5 minutes long video.

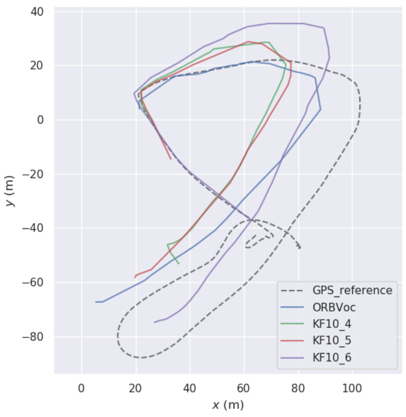
6.4.2 Comparison with different amount of training images

In this section the effect of using different amounts of training images was analyzed. The three different amounts of training images were $5\,000$, $10\,000$ and $20\,000$. All three used the vocabulary structure 10_5.

The comparison was done with the front right camera on the first video, as used previously as well. The trajectories created using 5k and 20k images were captured twice. The two runs for the 5k training images looked very similar to the one corresponding to the 10k



(a) xy-plot for vocabularies with $k = 9$.



(b) xy-plot for vocabularies with $k = 10$.

Figure 6.16: Trajectories created where vocabularies with different tree structures were used.

training images. For the case of the 20k training images, the two runs were different, the first was the longest while the second resembled the run for the 10_6 tree structure above.

Two additional runs and comparisons were made, these results built up under the same impression that there did not appear to be any immediate performance difference between the different vocabulary sizes. A smaller vocabulary is more lightweight, which can be preferred in some cases. However, a larger vocabulary has the ability to describe images in more detail. It was therefore decided to focus on comparison of the bundled 10^6 vocabulary form and our custom thermal 10^6 vocabulary generated with method 2.

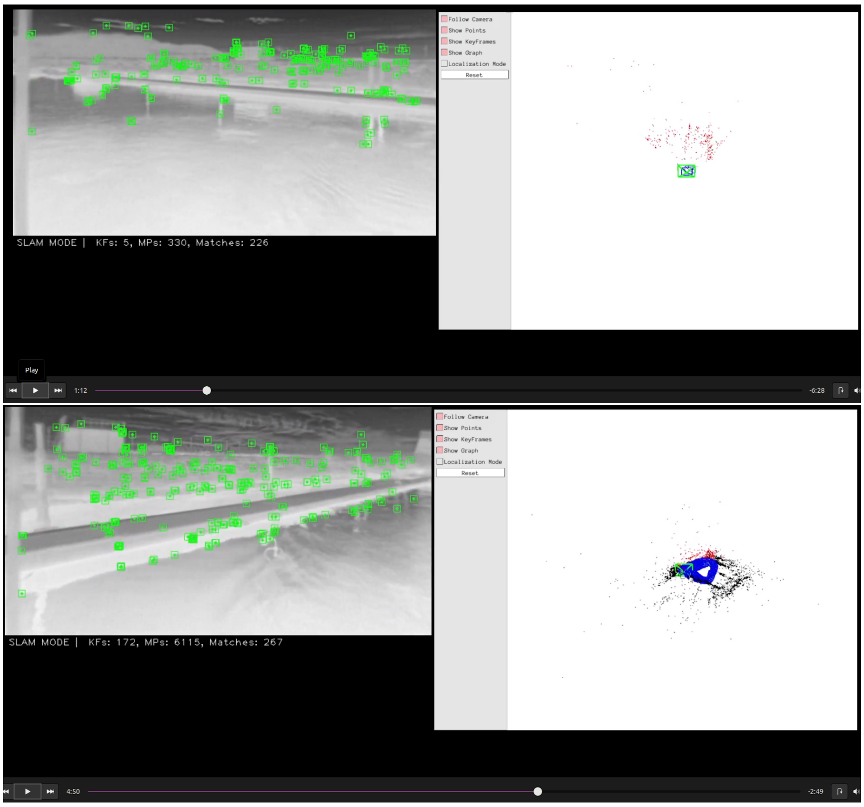


Figure 6.17: Similar scene, but different viewpoint.

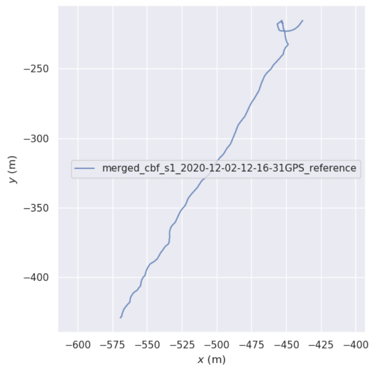


Figure 6.18: GPS trajectory for video file 4

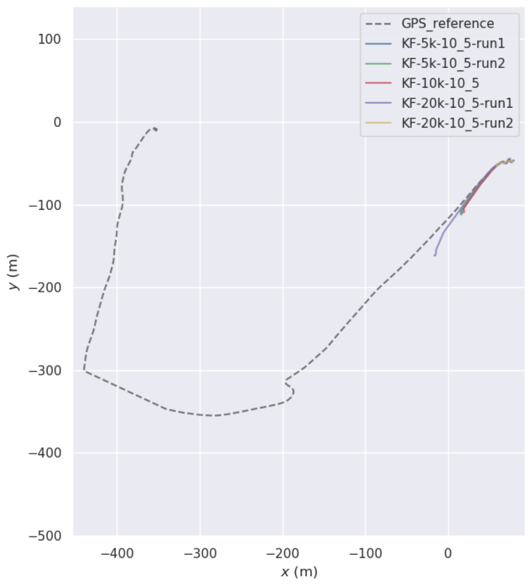


Figure 6.19: Amount of training images comparison.

6.4.3 Case study driving back and forth

The video file number 10 in Table 5.1 shows Milliampere sailing the same route back and forth five and a half times during 33 minutes, which is a realistic scenario for a ferry such as Milliampere. Figure 6.20 shows the corresponding GPS trajectory (xy-plot) as well as three timeseries plots along the x, y and z. The xy-plot shows overlaying trajectories, as the same path were taken several times. From the x axis timeseries plot we can see 5 rising edges and 6 falling edges (and the opposite for the y axis timeseries), which shows that five complete round trips and one half trip has been sailed.

Figure 6.21 and 6.22 both show pictures of the video and map windows during the 33 minute ferry run, taken at 5 and 30 minutes respectively. The image at 5 minutes is taken after the first complete round trip (back and forth), while the picture after 30 minutes is taken after 5 round trips.

- In Figure 6.21, the bundled ORB vocabulary was used, and we can see that the number of keyframes steadily increased from 22 to 123. The number of map points has increased as well from 1094 to 3348.

- In Figure 6.22, the custom 10_6 type thermal vocabulary was used, and the number of keyframes kept relatively stable at 20 and 24, and the number of map points also kept relatively stable at 1026 and 978, respectively.

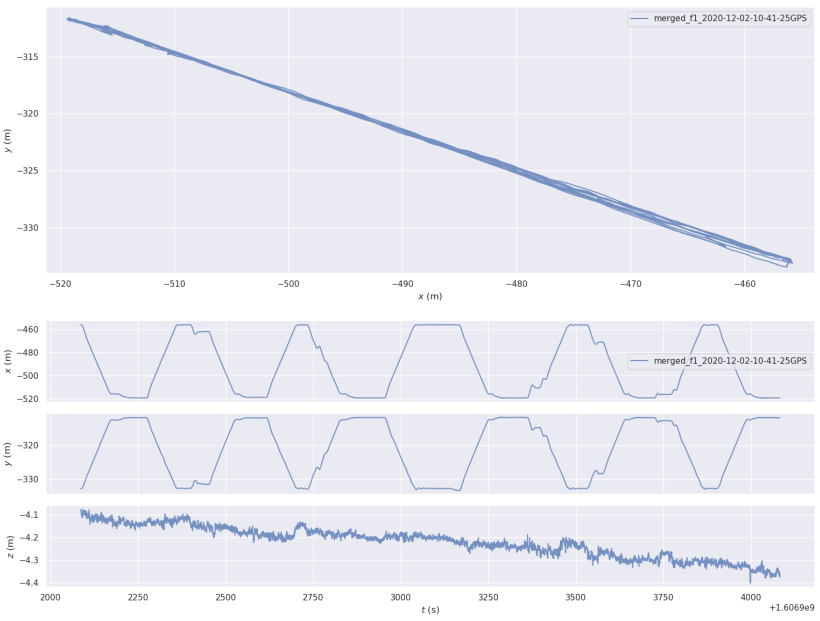
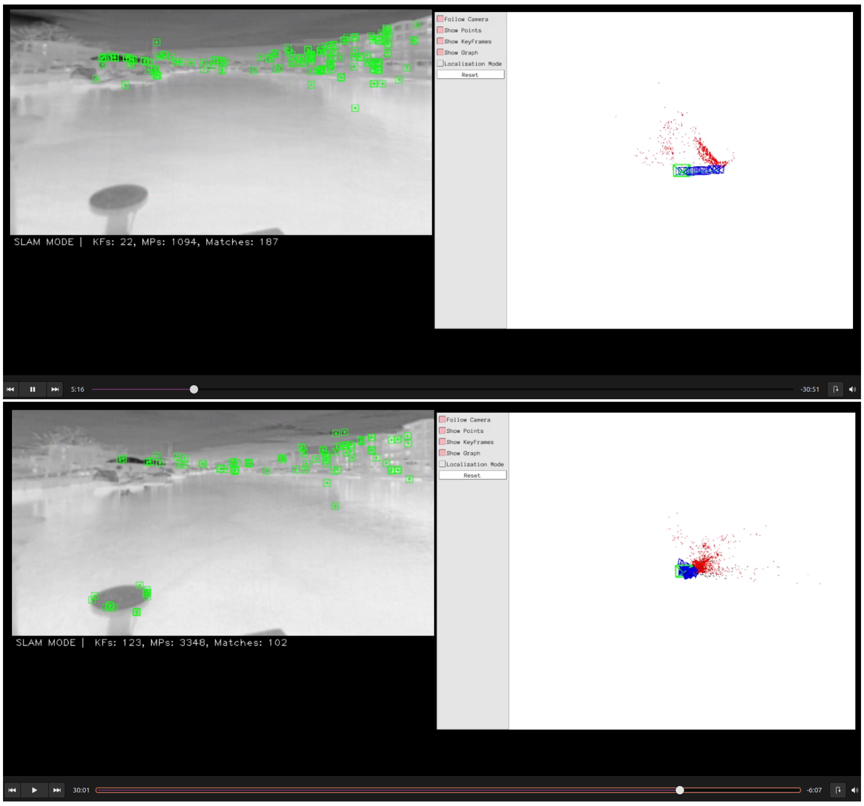


Figure 6.20: Video file 10 GPS trajectory and timeseries plot.

The scenario in Figure 6.21 and 6.22 was run again a total of three times as a form of verification, and the same tendency was shown using the custom thermal vocabulary. The results were approximately the same and varied very little, with only only one or two keyframes and map points. On the other hand, using the built-in ORB-SLAM vocabulary on the two next runs resulted in similar performance as the custom thermal.

These runs in 3/3 cases using the custom thermal vocabulary and 2/3 cases using the built-in ORB-SLAM were exciting. These results conforms to the ORB-SLAM theory, which has a survival of the fittest strategy with culling of redundant keyframes. In comparison, the number of keyframes kept growing in the first out of three runs when using the bundled ORB-SLAM vocabulary built on visual data. The result was exciting because the problem was kept at a manageable size of only around 20 keyframes. Typically some keyframes were added while the ferry was driving, but they were culled away after some time, keeping the problem at a constant size. This was reasonable as the same route is driven every time and the SLAM algorithm therefore does not get any new information. It was also a good result because the scenario was realistic, the ferry moves slower back and forth with no abrupt rotations and the method kept track for the full 33 minute period. The performance in this scenario also handled some faulty frames appearing on several times throughout the video. An example of the faulty frames causing the method to lose track can be seen in the middle image in Figure 6.24. For the purpose of SLAM these faulty frames can be comparable to inserting a black frame which will cause the algorithm to lose track and therefore it can be used to test short-term relocalization capabilities. The method



SLAM MODE | KFs: 22, MPs: 1094, Matches: 187

SLAM MODE | KFs: 123, MPs: 3348, Matches: 102

Figure 6.21: Ferry run (back and forth) using the bundled ORB vocabulary.

(with both vocabularies) handles the faulty frames very well and relocalize quickly. The method also showed some long-term relocalization capabilities as the method for every ferry round trip was able to recognize that the keyframe it had already have the information as the new ones.

The runs for the first runs with the front left camera as well as for the other cameras are summarized in Table 6.1. The front camera was not able to initialize for either of the vocabularies. This result was experienced several times, as the front camera typically show the same scene when driving, and therefore it does not get the sideways translation needed to initialize. All cameras except the front right camera result in the problem being bounded which is desired. The reason why using the front right camera in this case performed worse could be more features in the water and on the ferry. In general the custom thermal vocabulary has less keyframes and map points.

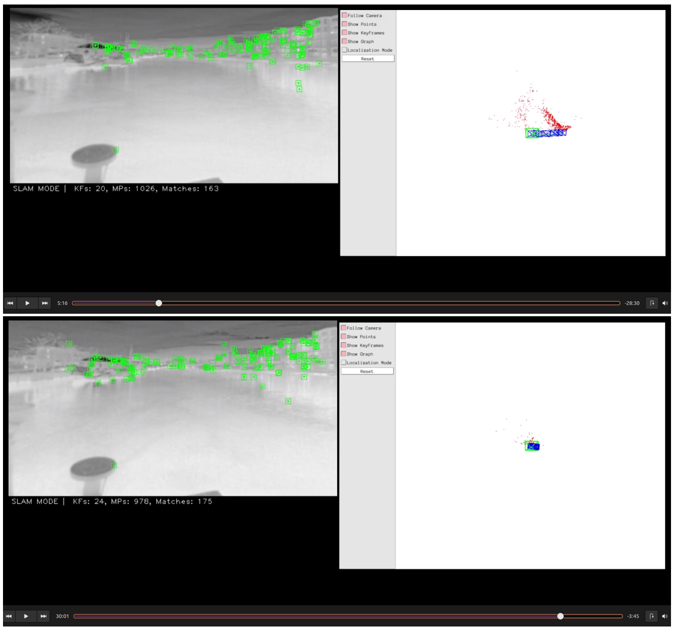

```
SLAM MODE |  KFs: 20, MPs: 1026, Matches: 163
```
```
SLAM MODE |  KFs: 24, MPs: 978, Matches: 175
```

Figure 6.22: Ferry run (back and forth) using the custom thermal vocabulary.

ORB-SLAM2						
Run	**Vocabulary**	**Camera**	**Key-frames**	**Map points**	**Initialization**	**Comment**
1	Custom thermal 10 6	FL	18	825	00:40	Much better performance. Bounded
2	ORB 10 6	FL	121	3184	00:49	Not bounded.
3	Custom thermal 10 6	FR	84	1728	03:14	Better performance. Not bounded.
4	ORB 10 6	FR	124	2420	08:55	Not bounded.
5	Custom thermal 10 6	RL	17	548	00:51	Better performance. Bounded.
6	ORB 10 6	RL	30	897	01:18	Bounded.
7	Custom thermal 10 6	RR	33	149	23:17	Slower init, but fewer map points. Similar number of KFs. Bounded.
8	ORB 10 6	RR	31	817	18:03	Bounded.
9	Custom thermal 10 6	F				Not able to initialize.
10	ORB 10 6	F				Not able to initialize.

Table 6.1: Case studies driving back and forth with ORB-SLAM2.

6.4.4 Choice of vocabulary

The vocabularies did perform surprisingly similar as they are very different for example does the 9_3 vocabulary only contain up to 729 words while the structure 10_6 contains up to 1 million words. The trajectory shapes and performance were similar. Forward it was therefore decided to use the custom 10_6 due to belief that a larger vocabulary could be more flexible and able to describe more situations and it has the same tree structure as the built-in ORB-SLAM vocabulary. It was done some random tests in later stages as well using the built-in ORB-SLAM vocabulary to verify that this decision was not wrong.

6.5 Comparing ORB-SLAM2 and ORB-SLAM3

The upgraded version of ORB-SLAM2, ORB-SLAM3 was implemented as described in section 5.8. A comparison to the results obtained with ORB-SLAM2 is here presented, both with and without IMU measurements.

6.5.1 Without IMU measurements

The first experiment was made using video file 1 and the rear right (RR) camera. In Figure 6.23. As mentioned in the theory ORB SLAM3 creates new maps when track is lost. In this example a new map was created after only a few seconds after the track was lost and the gap between part 1 and 2 was therefore in reality smaller than it appears in the figure. The two part could be connected by using a constant velocity assumption. Compared to ORB-SLAM2 this result was a lot better, as ORB-SLAM3 is able to continue tracking after the tunnel (which is where the tracking was lost). Also the ORB-SLAM3 trajectory follows the GPS reference very well.

Case study driving back and forth

Considering the same case study as for ORB-SLAM2 when the ferry was driving back and forth found in video 10. The results were a bit different. First, ORB-SLAM3 starts initializing a new map straight away when these faulty frames mentioned above appears. This results in several maps being created. However, the method was very good at merging these maps in this particular scenario. The scenario found in video file 10 was the only scenario were merging of maps have been observed during this thesis. Merging of maps indicates good performance of the vocabularies, as it was seen in the terminal output while running how the BoW scores are calculated, compared and matches are found. From the output log to ORB-SLAM3 it was possible to see how ORB-SLAM3 calculated the frames with highest bag of words score, how many meters they were separated by and the rotation between the keyframes. When a good enough candidate was found the maps were fused. Observing that the map fusion happened, over and over again, in this scenario put trust in the vocabularies capabilities to recognize previously visited places. This example was a good test for place recognition as the path was driven repeatedly. The viewpoints and location was very similar from run to run. The test succeeds as place recognition happened and maps were merged. In Figure 6.24 it can be seen in the first image that five maps have been created by looking in the text beneath the video playing or by the different colored keyframes which represent one map each. The second image show an example of a faulty frame while the third image show later in the video when all maps were merged to one.

Another observation that distinguished the ORB-SLAM3 performance from the ORB-SLAM2 performance in this scenario was how the number of keyframes and map points steadily increased and unfortunately was not bounded as in the ORB-SLAM2 cases. This was the case for all of the cameras, except the front camera which did not initialize. It was switched to the built-in ORB-SLAM vocabulary and tested for two cameras, but it did have the same result as the custom thermal vocabulary. In conclusion ORB-SLAM2 seemed to perform better than ORB-SLAM3 in this particular scenario.

6.5.2 With IMU measurements

ORB-SLAM3 has built-in support for IMU measurements, in this section the results of using ORB-SLAM3 with IMU is shown.

Exploring IMU noise parameters

Often it is difficult to take all noise into account, the noise model here used incorporated white noise and bias noise terms. Noise parameters are also computed in different conditions than the tested scenarios, causing the noise terms to in reality be higher. Therefore one rule of thumb is to multiply noise terms with a factor of ten. The noise terms were also multiplied with a factor of 100 as the trajectories were better without IMU. This multiplication was done in order to try to put less trust on the IMU measurements and rely more on the information from the camera sensor which has provided good results earlier. Example of trajectories created with different noise parameters can be seen in Figure 6.25. The trajectory using provided noise parameters increases a lot in scale after initialization. The noise parameters multiplied with a factor of 100 seemed to not reduce the influence of the IMU either. The noise parameters multiplied with a factor of 10 performed best, but the trajectory was still not impressive.

Case study driving back and forth

The case study was only tested for one video as the IMU integration did not work well. The result was increasing number of keyframes and map points. Also the trajectories seemed to constantly move to new place, and it was not visible that the ferry was moving back and forth by looking at the map.

Conclusion: Tracking with IMU

Tracking with the use of IMU appeared to initialize very quick in the tested scenarios. Also IMU kept track when there was a lot of quick rotations and in flat environments like the tunnel. In other words it kept track when ORB-SLAM2 and ORB-SLAM3 without IMU lost track. However, trajectories were quite off and a lot worse compared to the ones obtained without it. The noise parameters should ideally be calculated from measurements were the IMU would lay still on for example a table for a longer time period for example two hours with even temperature preferably the same as in operating conditions. Then an Alan deviation plot could be created and noise parameters more specific to our case could be calculated. This could possibly provide better result.

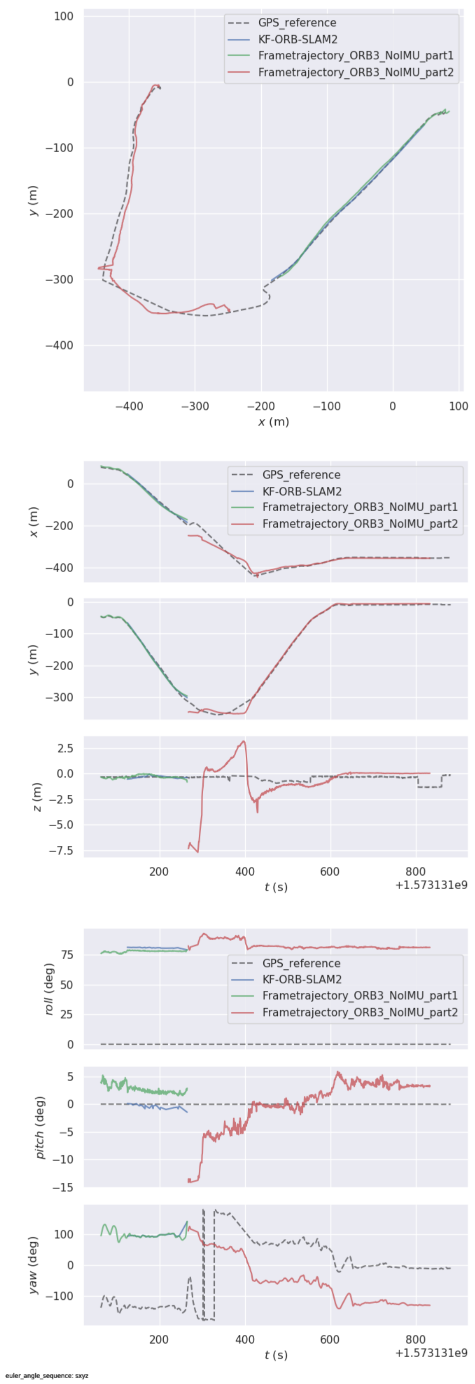
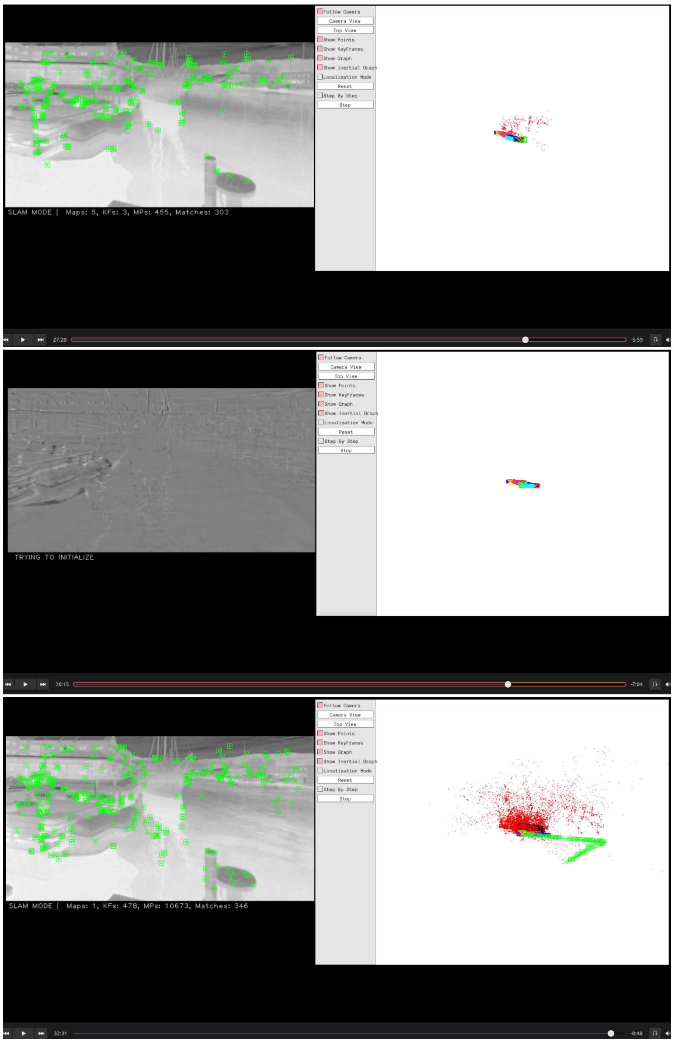


Figure 6.23: ORB-SLAM2 vs ORB-SLAM3 without IMU.

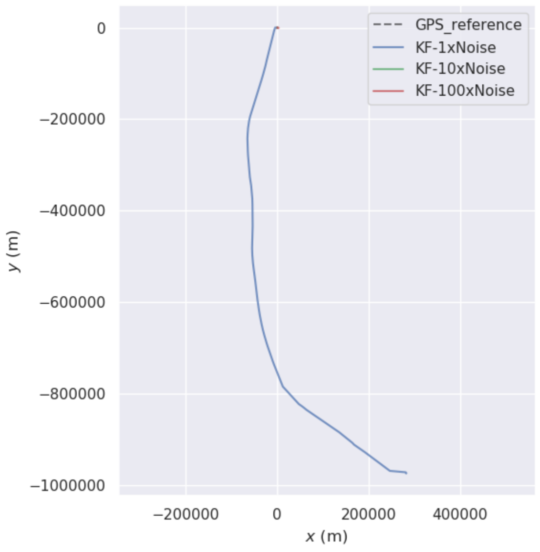


```
SLAM MODE |  Maps: 5, KFs: 3, MPs: 455, Matches: 303
```

```
TRYING TO INITIALIZE
```

```
SLAM MODE |  Maps: 1, KFs: 478, MPs: 10673, Matches: 346
```

Figure 6.24: ORB-SLAM3 tracking with multiple maps and faulty frames.



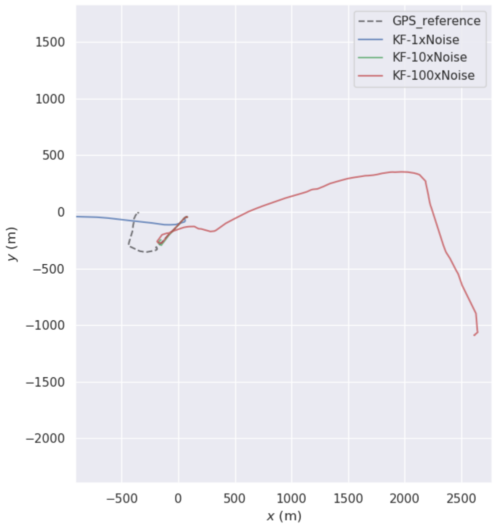


Figure 6.25: ORB-SLAM3 trajectories created with IMU.

Chapter 7

Conclusion and future work

7.1 Conclusion

The discovery of how much the distortion parameters affected the ORB-SLAM method was a big step in this thesis as it led to more reasonable and improved results. Using all five of the distortion parameters when doing camera calibration led to an overestimated camera model which showed poorer performance than when using none. Camera calibration with one or two distortion parameters could work as well.

In this thesis vocabularies with similar performance as the one built-in for ORB-SLAM has been created. A particular result of interest was the case study with the ferry driving back and forth where the custom thermal vocabulary as well as the built-in ORB-SLAM vocabulary kept the amount of keyframes bounded. The results in this thesis are more in a case study manner and qualitative rather than quantitative. Either way the thermal ORB-SLAM performed well for this ferry driving back and forth scenario which is realistic, no abrupt turns and slower motion and we saw example of ORB-SLAM with custom-thermal vocabulary being able to keep the number of keyframes below 30 during half an hour of tracking, showing good relocalizing abilities.

The upgrade to ORB-SLAM3 gave improved tracking. The creation of several maps in one run provided useful, in the cases were ORB-SLAM2 would have lost track for example in the tunnel. ORB-SLAM3 could create a map before and after the tunnel. ORB-SLAM3 with IMU were capable to keep track in scenarios with quick rotations and flat areas like the tunnel, but with the settings here used it went on expense of the accuracy of the trajectory.

7.2 Future work

RDS-SLAM was released January 2021, but is under optimization and therefore has not yet been released as open-source at the time of writing. This could be implemented at a later time to add semantics to ORB-SLAM3. RDS-SLAM is currently only available for RGB-D camera, but according to their paper, the plan is to extend the system to work for monocular camera (as in our case), as well as stereo cameras. Semantics can for example be used to distinguish features of dynamic objects (like for example persons), whose features may be handled differently for the purposes of for tracking, mapping and collision avoidance.

14-bit radiometric thermal images is an interesting alternative with potential for better performance, but would require modifications to the pipeline of ROS and ORB-SLAM. Shin and Kim (2019) have proposed a 14-bit thermal SLAM method which might work well in this scenrio, and would be interesting to implement in the future as an alternative to ORB-SLAM.

True scale can be important for navigation, and it would be interesting to implement ORB-SLAM with stereo thermal video, which ORB-SLAM has support for. This would however require a adaptations to the sensor rig.

Since thermal images often appear to show large surfaces with gradual transitions of object's temperatures, often leading to less sharp and clearly defined corners and edges in some situations compared to visual images, it would be interesting to implement a dense thermal VSLAM method.

ORB-SLAM3 with IMU did not perform as well as desired, and more fine tuning of IMU noise parameters may be needed. Integration with other sensors like optical camera and lidar can also be explored in the future for improved tracking and mapping.

Bibliography

Anderson, S.E., . Bit twiddling hacks. URL: `http://graphics.stanford.edu/~seander/bithacks.html`.

Arthur, D., Vassilvitskii, S., 2007. K-means++: The advantages of careful seeding, pp. 1027–1035. doi:`10.1145/1283383.1283494`.

Borges, P.V.K., Vidas, S., 2016. Practical infrared visual odometry. IEEE Transactions on Intelligent Transportation Systems 17, 2205–2213. doi:`10.1109/TITS.2016.2515625`.

Byrnes, J., 2009. Unexploded Ordnance Detection and Mitigation. Springer. Pages 21–22.

Calonder, M., Lepetit, V., Strecha, C., Fua, P., 2010. Brief: Binary robust independent elementary features, in: Daniilidis, K., Maragos, P., Paragios, N. (Eds.), Computer Vision – ECCV 2010, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 778–792.

Campos, C., Elvira, R., Rodríguez, J.J.G., Montiel, J.M.M., Tardós, J.D., 2020. Orb-slam3: An accurate open-source library for visual, visual-inertial and multi-map slam.

Dorian Galvez-Lopez and Avatar Javier Hidalgo-Carrió and Anup Parikh, 2016. DLib. `https://github.com/dorian3d/DLib`. Published by GitHub user dorian3d.

El-Sheimy, N., Hou, H., Niu, X., 2008. Analysis and modeling of inertial sensors using allan variance. IEEE Transactions on Instrumentation and Measurement 57, 140–149. doi:`10.1109/TIM.2007.908635`.

Engel, J., Koltun, V., Cremers, D., 2016. Direct sparse odometry. CoRR abs/1607.02565. URL: `http://arxiv.org/abs/1607.02565`, `arXiv:1607.02565`.

Engel, J., Schöps, T., Cremers, D., 2014. Lsd-slam: Large-scale direct monocular slam, in: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (Eds.), Computer Vision – ECCV 2014, Springer International Publishing, Cham. pp. 834–849.

Engel, J., Stückler, J., Cremers, D., 2015. Large-scale direct slam with stereo cameras, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1935–1942. doi:`10.1109/IROS.2015.7353631`.

Faugeras, O., Lustman, F., 1988. Motion and structure from motion in a piecewise planar environment. International Journal of Pattern Recognition and Artificial Intelligence - IJPRAI 02. doi:10.1142/S0218001488000285.

Forster, C., Pizzoli, M., Scaramuzza, D., 2014. Svo: Fast semi-direct monocular visual odometry, in: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp. 15–22. doi:10.1109/ICRA.2014.6906584.

Fu, Z., Quo, Y., Lin, Z., An, W., 2017. Fsvo: Semi-direct monocular visual odometry using fixed maps, in: 2017 IEEE International Conference on Image Processing (ICIP), pp. 2553–2557. doi:10.1109/ICIP.2017.8296743.

Gálvez-López, D., Tardós, J.D., 2012. Bags of binary words for fast place recognition in image sequences. IEEE Transactions on Robotics 28, 1188–1197.

Gao, X., Wang, R., Demmel, N., Cremers, D., 2018. Ldso: Direct sparse odometry with loop closure, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2198–2204. doi:10.1109/IROS.2018.8593376.

Gonzalez, M.F., Furlong, P.M., Dille, M., Wong, U.Y., 2019. Fusion of visible and thermal-infrared imagery for slam for landing on icy moons. https://ntrs.nasa.gov/citations/20190027478.

Grupp, M., 2017. evo: Python package for the evaluation of odometry and slam. https://github.com/MichaelGrupp/evo.

Harris, C., Stephens, M., 1988. A combined corner and edge detector, in: Alvey Vision Conference.

Hølland, E.H., 2019. Maritime object detection using infrared cameras. Master's thesis. NTNU.

IEEE, 1998. Ieee standard specification format guide and test procedure for single-axis interferometric fiber optic gyros. IEEE Std 952-1997 , 1–84doi:10.1109/IEEESTD.1998.86153.

Khattak, S., Papachristos, C., Alexis, K., 2019. Keyframe-based direct thermal-inertial odometry. CoRR abs/1903.00798. URL: http://arxiv.org/abs/1903.00798, arXiv:1903.00798.

Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., Burgard, W., 2011. G2o: A general framework for graph optimization, pp. 3607 – 3613.

Leutenegger, S., Furgale, P., Rabaud, V., Chli, M., Konolige, K., Siegwart, R., 2013. Keyframe-based visual-inertial slam using nonlinear optimization. doi:10.15607/RSS.2013.IX.037.

Liu, Y., Miura, J., 2021. Rds-slam: Real-time dynamic slam using semantic segmentation methods. IEEE Access , 1–1doi:10.1109/ACCESS.2021.3050617.

Loo, S.Y., Amiri, A.J., Mashohor, S., Tang, S.H., Zhang, H., 2019. Cnn-svo: Improving the mapping in semi-direct visual odometry using single-image depth prediction, in: 2019 International Conference on Robotics and Automation (ICRA), pp. 5218–5223. doi:10.1109/ICRA.2019.8794425.

Ma, P., Bai, Y., Zhu, J., Wang, C., Peng, C., 2019. Dsod: Dso in dynamic environments. IEEE Access 7, 178300–178309. doi:10.1109/ACCESS.2019.2958374.

Mordvintsev, A., Abid, K., 2013. OpenCV-Python Tutorials. Revision 43532856.

Mur-Artal, R., Montiel, J.M.M., Tardós, J.D., 2015. ORB-SLAM: a Versatile and Accurate Monocular SLAM System. CoRR, Computing Research Repository .

Mur-Artal, R., Tardós, J.D., 2016. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras .

Mur-Artal, R., Tardós, J.D., 2014. Fast relocalisation and loop closing in keyframe-based slam, in: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp. 846–853. doi:10.1109/ICRA.2014.6906953.

Open Source Robotics Foundation, 2013. ROS vision_msgs Package Summary at wiki.ros.org.

Open Source Robotics Foundation, 2018. ROS Melodic version documentation at wiki.ros.org.

Raul Mur-Artal and Juan D. Tardos and J. M. M. Montiel and Dorian Galvez-Lopez, 2016. ORB-SLAM2, Real-Time SLAM for Monocular, Stereo and RGB-D Cameras, with Loop Detection and Relocalization Capabilities. Published by GitHub user raulmur.

user rehderj, G., 2016. The kalibr visual-inertial calibration toolbox - github wiki - imu noise model. https://github.com/ethz-asl/kalibr/wiki/imu-noise-model.

Rosten, E., Porter, R., Drummond, T., 2010. Faster and Better: A Machine Learning Approach to Corner Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence 32, 105 –119.

Saputra, M.R.U., de Gusmao, P.P.B., Lu, C.X., Almalioglu, Y., Rosa, S., Chen, C., Wahlström, J., Wang, W., Markham, A., Trigoni, N., 2019. Deeptio: A deep thermal-inertial odometry with visual hallucination. CoRR abs/1909.07231. URL: http://arxiv.org/abs/1909.07231, arXiv:1909.07231.

Shin, Y., Kim, A., 2019. Sparse depth enhanced direct thermal-infrared slam beyond the visible spectrum. IEEE Robotics and Automation Letters 4, 2918–2925. doi:10.1109/LRA.2019.2923381.

Trippel, T., Weisse, O., Xu, W., Honeyman, P., Fu, K., 2017. Walnut: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks, in: 2017 IEEE European Symposium on Security and Privacy (EuroS P), pp. 3–18. doi:10.1109/EuroSP.2017.42.

Vidas, S., Sridharan, S., 2012. Hand-held monocular slam in thermal-infrared, in: 2012 12th International Conference on Control Automation Robotics Vision (ICARCV), pp. 859–864. doi:10.1109/ICARCV.2012.6485270.

Vydhyanathan, A., Bellusci, G., 2018. The next generation xsens motion trackers for industrial applications. https://www.xsens.com/hubfs/Downloads/Whitepapers/MTi_whitepaper.pdf.

Yan, C., Shin, H., Bolton, C., Xu, W., Kim, Y., Fu, K., 2020. Sok: A minimalist approach to formalizing analog sensor security, in: 2020 IEEE Symposium on Security and Privacy (SP), pp. 233–248. doi:10.1109/SP40000.2020.00026.

Appendix A

ROS mono C++ file modifications

This appendix describes necessary modifications to the ROS mono C++ file in order to subscribe to vision_msgs messages instead of the default sensor_msgs. Changes necessary for cropping are also included.

```
/**
* This file is part of ORB-SLAM2.
*
* Copyright (C) 2014-2016 Raul Mur-Artal <raulmur at unizar dot es>
* (University of Zaragoza)
* For more information see <https://github.com/raulmur/ORB_SLAM2>
*
* ORB-SLAM2 is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* ORB-SLAM2 is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with ORB-SLAM2. If not, see <http://www.gnu.org/licenses/>.
*/


#include<iostream>
#include<algorithm>
#include<fstream>
```

```
#include<chrono>
#include <math.h> //round

#include<ros/ros.h>
#include <cv_bridge/cv_bridge.h>

#include<opencv2/core/core.hpp>
#include"../../../include/System.h"
#include <vision_msgs/Detection2D.h>


using namespace std;

class ImageGrabber
{
public:
    ImageGrabber(ORB_SLAM2::System* pSLAM):mpSLAM(pSLAM){}

    void GrabImage(const vision_msgs::Detection2DConstPtr& msg);

    ORB_SLAM2::System* mpSLAM;
};

int main(int argc, char **argv)
{
    ros::init(argc, argv, "Mono");
    ros::start();

    if(argc != 3)
    {
        cerr << endl << "Usage: rosrun ORB_SLAM2 Mono
        path_to_vocabulary path_to_settings" << endl;
        ros::shutdown();
        return 1;
    }

    ORB_SLAM2::System SLAM(argv[1],argv[2],ORB_SLAM2::System
    ::MONOCULAR,true);

    ImageGrabber igb(&SLAM);

    ros::NodeHandle nodeHandler;

    ros::Subscriber sub = nodeHandler.subscribe("/camera/image_raw"
    , 1, &ImageGrabber::GrabImage,&igb);
    ros::spin();

    // Stop all threads
    SLAM.Shutdown();

    // Save camera trajectory
    SLAM.SaveKeyFrameTrajectoryTUM("KeyFrameTrajectory.txt");

    ros::shutdown();

    return 0;
}

 void ImageGrabber::GrabImage(const
 vision_msgs::Detection2DConstPtr& msg)
{
    // Copy the ros image message to cv::Mat.
    cv_bridge::CvImagePtr cv_ptr;
    try
    {
                // Crop image to remove upper 1/3 of image
                cv_ptr = cv_bridge::toCvCopy(msg->source_img);

                int width = (int) round(cv_ptr->image.size().width);
                int height = (int) round(cv_ptr->image.size().height);
                int y = (int) round(height/3);

                cv::Rect2d r(0,y,width,height-y);
                cv_ptr->image = cv_ptr->image(r);
                // Cropping completed

                // Add timestamp
                cv_ptr->header.stamp.sec = msg->header.stamp.sec ;
                cv_ptr->header.stamp.nsec = msg->header.stamp.nsec ;

    }
    catch (cv_bridge::Exception& e)
    {
        ROS_ERROR("cv_bridge exception: %s", e.what());
        return;
    }
    mpSLAM->TrackMonocular(cv_ptr->image, cv_ptr->header.stamp.toSec());
}
```

Appendix B

Code used for converting a DBOW2 vocabulary to an ORB-SLAM vocabulary.

```
# This file converts a DBoW2 vocabulary to an ORB–SLAM vocabulary.
# Note that NodeID column and all wordID rows are removed as well as
# all strings (like ”parentId”, ”weights) and symbols
# (like brackets ”{}” and quotation marks ”:”).

file_input_name = ”bovisa8000.yml”
file_output_name = ”bovisa8000_cleaned.txt”

f = open(file_input_name, ”r”)
out = open(file_output_name, ”w”)

# Make a temporary text file, where the last part of the text
# (after ”words:”) is removed.
temp1 = open(”temp1.txt”, ”w”)

# Detect the line containing ”words:”
current_weight = 0
for line in f:
        if ”words:” in line:
                break

        elif (”%YAML:1.0” in line) or (”−−−” in line) or
        (”vocabulary:” in line) or (”␣␣␣nodes:” in line):
                # do nothing
        elif (”␣␣␣k:␣” in line):
                txt = line.split()
                temp1.write(txt[−1]+”␣”)

        elif (”␣␣␣L:␣” in line):
                txt = line.split()
                temp1.write(txt[−1]+”␣␣”)

        elif (”␣␣␣scoringType:␣” in line):
                txt = line.split()
                temp1.write(txt[−1]+”␣”)

        elif (”␣␣␣weightingType:␣” in line):
                txt = line.split()
                temp1.write(txt[−1]+”\n”)

        else:
                if (line.split()[0]==”−”):
                        x = line.split()
                        nodeId   =(x[2].split(”:”)[1].split(”,”)[0])
                        parentId=(x[3].split(”:”)[1].split(”,”)[0])
                        temp1.write(parentId +”␣”)
                        current_weight  = (x[4].split(”:”)[1]
                        .split(”,”)[0])
                        if current_weight == ”0.”:
                                current_weight = ”0”
                        else:
                                current_weight = str(float(”{:.5f}”
                                .format(float(current_weight))))

                else:
                        line.split(’”’)[1]
                        temp1.write(line.split(’”’)[1] + ”␣” +
                        current_weight + ”\n”)

temp1.close()
f.close()
f  = open(”temp1.txt”, ”r”)
f2 = open(”temp1.txt”, ”r”)

# Handle first line
line = f.readline()
line = f2.readline()
out.write(line)
```

```
# init
line = f.readline()
elements = (line).split()
countChildrenNodes = 1
previous = elements[0]
column2 = '0'
if elements[-1]!='0':
        column2 = '1'

for line in f:
        elements = (line).split()
        if elements[0]==previous:
                countChildrenNodes = countChildrenNodes + 1
                if elements[-1]!='0':
                        column2 = '1'
        else:
                for i in range(countChildrenNodes):
                        line2 = f2.readline()
                        elements2 = (line2).split()
                        for x in range(len(elements2)):
                                if x == 0:
                                        out.write(elements2[x] +
                                        "␣" + column2)
                                elif x == len(elements2)-1:
                                        out.write("␣␣" +
                                        elements2[x] + "\n")
                                else:
                                        out.write("␣" + elements2[x])
                previous = elements[0]
                if elements[-1]=='0':
                        column2 = '0'
                else:
                        column2 = '1'
                countChildrenNodes=1


f.close()
f2.close()
out.close()
```

Appendix C

Code used for thermal image capturing

This appendix describes modifications to a thermalgrabber example code in order to save thermal images. The images are saved by the use of OpenCV. The string variable called ”path” can be modified in order to save the images in the desired folder. The images are saved with a filename on the format ”YYYY-MM-DDThh:mm:ss.mmm+01:00” which is based on the ISO 8601 with the only modification being mmm at the end indicating milliseconds.

```
#include "thermalgrabber.h"

#include <stdbool.h>
#include <cstdlib>
#include <iostream>
#include <chrono>
#include <thread>
#include <cstdint>
#include <cstring>
#include <string>
#include <ctime>

#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>

using namespace cv;
using namespace std;
using namespace std::chrono;
```

```
ThermalGrabber* tGr;
string filename_start = "";
string path = "/home/";

class Test
{
public:
    void test();
    };

void callbackTauImage(TauRawBitmap& tauRawBitmap, void* caller)
{
    // Save thermal image as a OpenCV Mat object
    Mat image = Mat(tauRawBitmap.height, tauRawBitmap.width, CV_8UC1);
    for (int row = 0; row < tauRawBitmap.height ; row++) {
        for (int col = 0; col < tauRawBitmap.width ; col++) {
            image.at<unsigned char>(row,col) =
            tauRawBitmap.data[col + row*tauRawBitmap.width];
        }
    }


    time_t now = time(0);
    tm *ltm = localtime(&now);

    string min = to_string(ltm->tm_min);
    string sec = to_string(ltm->tm_sec);


    int64_t ms = duration_cast< milliseconds >(system_clock::now()
    .time_since_epoch()).count()%1000;
    string millisec = to_string(ms);
    if (millisec.length() == 1){
        millisec = "00" + millisec;
    }
        if (millisec.length() == 2){
            millisec = "0" + millisec;
        }


    // Write the image to a defined location as JPEG
    bool check = imwrite(path + filename_start + min + ':' + sec +
    ':' + millisec + "+01:00" +".jpg", image);
```

```
}

void Test::test()
{

    tGr = new ThermalGrabber(callbackTauImage, this);

    unsigned int mWidth = tGr->getResolutionWidth();
    unsigned int mHeight = tGr->getResolutionHeight();

    // run demo for 20 seconds
    std::this_thread::sleep_for(milliseconds(20000));
    delete tGr;
}

void createFilenameWithTimeAndDate()
{
    time_t now = time(0);
    tm *ltm = localtime(&now);

    string year = to_string(1900 + ltm->tm_year);
    string month = to_string(1+ ltm->tm_mon);
    string day = to_string(ltm->tm_mday);
    string hour = to_string(ltm->tm_hour -1);
    string min = to_string(ltm->tm_min);
    string sec = to_string(ltm->tm_sec);

    // Add "0" to keep consistent format
    if (month.length() == 1){
        month = "0" + month;
    }
    if (day.length() == 1){
        day = "0" + day;
    }
    if (hour.length() == 1){
        hour = "0" + hour;
    }
    if (min.length() == 1){
        min = "0" + min;
    }
    if (sec.length() == 1){
        sec = "0" + sec;
    }
    filename_start = year + '-' + month + '-' + day + 'T'
        + hour + ':';

}
int main()
{
    createFilenameWithTimeAndDate();
    {
        Test* t = new Test();
        t->test();
        delete t;
    }
    return 0;
}
```

Hallvard Fosso

NTNU
Norwegian University of
Science and Technology